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

Adam Wierman[1] Takayuki Osogami[2] Jörgen Olsén[3]

May 2003

CMU-CS-03-133

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a general analytical framework for the modeling and analysis of TCP variations. The framework is quite comprehensive and allows the modeling of multiple variations of TCP, i.e. TCP-Vegas, TCP-SACK, and TCP-Reno, under very general network situations. In particular, the framework allows us to propose the first analytical model of TCP-Vegas under on-off traffic – all existing analytical models of TCP-Vegas assume bulk transfer only. All TCP models are validated against event driven simulations (*ns*-2) and existing state-of-the-art analytical models. Finally, 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]Computer Science Department, Carnegie Mellon University. Email: acw@cs.cmu.edu
[2]Computer Science Department, Carnegie Mellon University. Email: osogami@cs.cmu.edu
[3]Department of Mathematics, Uppsala University, Sweden. Email: jorgen@math.uu.se

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, an NSF Graduate Research Fellowship, and a project grant from the Swedish Foundation for Strategic Research. This report is also published as U.U.D.M Report 2003:12 ISSN 1101-3591 at Uppsala University.

Keywords: Internetworking Protocol and Performance; Performance Analysis and Validation; TCP-Vegas; TCP-SACK; TCP-Reno

1 Introduction

Much of today's Internet traffic is carried using the Transmission Control Protocol (TCP), and consequently there has been a significant amount of research toward modeling and understanding the impact that this protocol has on file transmission times and network utilization. TCP has been and will continue to be an evolving protocol, and as such, one important task of these models is to facilitate comparisons between the different flavors of TCP. Because TCP has gone through incremental refinements, the protocol itself has grown increasingly complex, which makes analytical modeling quite difficult. As a result, much of the evaluation of TCP variations has been done using event driven simulators, such as *ns*-2, a network simulator developed at Lawrence Berkeley National Laboratory [32]. More recently, research has moved toward analytical modeling of the performance of TCP. This movement has been triggered by the limitations inherent in event driven simulations, which can be quite time consuming for fast networks and force researchers to use a packet level granularity even when investigating large, complex networks. Also, when designed carefully, analytical models help researchers understand the effectiveness of new mechanisms being added to the protocol. As a result, many papers have been written introducing new, analytical models of the performance of TCP. However, most of the 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 of the dependence of TCP-Vegas on the delay experienced by packets in the network. Specifically, TCP-Vegas uses observed delay to detect an incipient stage of congestion and tries to adjust the sending rate before packets are lost. Thus, unlike TCP-Reno, TCP-Vegas attempts to determine the correct sending rate without relying on packet losses. 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 and how TCP-Vegas should be incrementally deployed.

The goal of this paper is to propose a unified framework for analytical modeling of TCP variations, including TCP-Vegas, TCP-Reno, and TCP-SACK. Our framework allows modeling the behavior of TCP senders under very general network settings: it allows mixtures of multiple senders, where each sender can use a different flavor of TCP and carry out a different type of communication (such as HTTP connections or FTP connections). It also allows the network to have a general topology, and hence each sender can observe a different round trip time (RTT) and a different loss rate.

A wide variety of techniques have been applied to the problem of TCP modeling with a fair amount of success. These techniques range over fluid models [31], Markov chains [20], and renewal theory [19]. However, until the recent, seminal work of papers such as [9, 10, 13, 24] no modeling technique has been able to mimic both the structure of a TCP source and the interaction a source has with the network. The framework proposed by Casetti and Meo [9, 10] takes the novel approach of separating the modeling of network behavior from the modeling of the behavior within a TCP source and then allowing the two to interact via feedback. The parameters of the TCP source model and the network model are tuned by iterative analysis with feedback from each other. The TCP source model is analyzed to obtain the aggregate network traffic, which is then used as an input to the network model. Next, the network model is analyzed to obtain the loss rate and expected queueing delay, which are then used as the parameters in source model. The source model is analyzed again, and the procedure is continued until a stable solution is obtained. Given accurate source and network models, the fixed point of this iteration matches the operating point of the true network. Using this approach Casetti and Meo are able to accurately model the performance of TCP-Reno connections in [10] and a simplified TCP-Tahoe protocol in [9]. Garetto, Cigno, Meo and Marsan have since refined the TCP models in [9] and [10], and have considered both persistent connections and transfers of files coming from arbitrary file-length distributions [14, 16, 15]. The TCP models, however, are still limited to TCP-Tahoe and TCP-Reno.

In this paper, we generalize the framework of Casetti and Meo in order to improve their model of TCP-Reno, as well as model TCP-Vegas and TCP-SACK connections. We extend their framework to include the exponential backoff mechanism of TCP-Reno and the selective acknowledgment mechanism used in TCP-SACK. In addition, we present a model of the novel slow-start and congestion avoidance mechanisms in TCP-Vegas. 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 is nontrivial. In order to apply the framework, we need to 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.

In addition to modeling multiple variations of TCP, a second focus of this work is on validating possible network models. In [10], a $M/M/1/B$ queue is used to model networks with bottleneck topology; however there is intuition that other, simple queueing models might provide better analysis of these networks. For instance, an $M/D/1/B$ model takes into account the fact that packet sizes are likely to have a distribution with low variability; and an $M^r/M/1/B$ batch arrival model describes the fact that TCP traffic is likely to be quite bursty due to synchronized loss events that are experienced by multiple users. In this work, we analyze the effectiveness of these three possible queueing systems for use in modeling a network with a bottleneck topology.

Related work

Before delving into our model results, it is useful to understand where our model sits among previous research in this area. We start by describing the prior work on the analytical modeling of TCP-Vegas and then move to previous techniques for the analytical modeling of TCP-Reno. In particular, all of the prior work on TCP-Vegas is limited to a single sender of bulk transfer and most of the prior work on both TCP-Vegas and TCP-Reno is limited to the analysis of the throughput of a single sender as a function of the loss rate in the network.

The model of TCP-Vegas presented in this work overcomes two limitations of previous TCP models. First, our model is the first model of TCP-Vegas sources sending on-off traffic; all previous work only allowed bulk transfers [5, 6, 17, 21, 22, 25, 33]. Bulk transfer models are usually associated with FTP traffic, where a user sends a large amount of data without ever closing the connection. On-off traffic provides a more general model, one that actually includes bulk transfer as a special case (when the on periods are very 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. Second, our model is the first model of TCP-Vegas that accurately predicts the operating point of the network in terms of throughput and loss rate for on-off traffic. Most of the prior work gives the throughput only under loss-free operation [5, 6, 17, 21, 22, 25]. Samios and Vernon [33] recently proposed the first analytical model to incorporate loss rate. Based on a renewal approach, they give a closed-form formula for the throughput achieved by a bulk transfer of a single TCP-Vegas sender as a function of the loss rate and the round trip time (RTT) of the network. However, since their derived closed form is dependent on knowledge of the achieved loss rate, they are not able to predict the full operating point of the network.

We now briefly describe prior work on developing analytical models for TCP-Reno. Roy et al. [31] have investigated a fluid model of individual simplified TCP-Reno connections and derived expressions for the congestion window size and queue length evolution over time of the connection. Altman et al. [3] have investigated a more detailed TCP fluid model under the assumptions of both independent and correlated losses. Another style of approaches uses stochastic analysis of the TCP window size and renewal theory to again provide an expression for the throughput of a single connection [2, 18, 19, 23, 28]. Yet another technique derives a Markov chain for the evolution of the TCP congestion window size to calculate the throughput of a single connection [20]. Unfortunately, though these approaches accurately predict performance metrics such as the throughput of a single persistent TCP connection given the loss rate of a network, they are not applicable to more general settings including mixtures of various TCP senders and general network topologies. Our work belongs to a line of research that fills this gap by combining detailed models of TCP sources and queuing models of general network topologies [4, 8, 9, 10, 22, 30].

Summary of results

In this paper, we propose the first analytical model of TCP-Vegas for on-off traffic. The framework of our analysis allows for applicability 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:

- *We illustrate a general framework for modeling variations of TCP.* We extend the framework of Casetti and Meo [9, 10] to allow modeling of performance metrics such as the throughput, loss rate, and queueing delay, of various TCP flavors including TCP-Reno, TCP-SACK, and TCP-Vegas, under very general network environments: heterogeneous TCP sources with general on-off traffic connected to a network with a general topology.

- *We propose a novel analytical model for TCP-Vegas under on-off traffic, and the accuracy of the model is validated against both ns-2 simulations and existing analytical models.* Previous work on modeling TCP-Vegas all focused on persistent connections (i.e. bulk transfers), and hence our model is the first that allows the analysis of on-off traffic. We validate the model against *ns*-2 simulations and find that the model is accurate in very general situations. We also compare our novel model of TCP-Vegas to the current state-of-the-art model in the literature [33]. Since all the prior work on modeling TCP-Vegas is limited to persistent connections, we show that our model is at least as accurate as the state-of-the-art in networks with persistent connections while verifying the model against *ns*-2 for more general network setups.

- *We extend the analytical model for TCP-Reno proposed in [10] to include the exponential backoff mechanism and a minimum timeout duration. In addition, we extend the framework to model the selective acknowledgements used in TCP-SACK.* The extensions to the model for TCP-Reno improve the accuracy of the model significantly in high loss environments. Additionally, selective acknowledgements allow us to model TCP-SACK, the accuracy of which is validated against ns simulations.

- *We extensively test the applicability of standard queueing models for approximating the performance of a bottleneck link shared among TCP sources and find, surprisingly, that an $M/M/1/B$ queue is a good approximation.* We test various queueing models, including $M/M/1/B$, $M/D/1/B$, and $M^r/M/1/B$, and find that the loss rate predicted by an $M/M/1/B$ queue is a good approximation of the simulated bottleneck link shared among TCP sources under any traffic load. In addition, the queueing model allows our complete model to predict the operating point of a TCP network within five percent accuracy. This is surprising since the actual packet size distribution in the network has low variability and the arrival process is not necessarily well approximated by a Poisson process in simulation. We provide intuition for this surprising observation.

- *We investigate the effectiveness of the new mechanisms introduced in TCP-SACK and TCP-Vegas and conclude that the selective acknowledgment mechanism introduced in TCP-SACK is effective at avoiding timeouts and that the modified slow-start mechanism introduced in TCP-Vegas does not help reduce packet loss but does result in wasting more time in the slow-start phase.* Our framework allows investigating information such as the fraction of time that a TCP sender spends in each TCP state, which we find difficult to obtain in event driven simulations such as *ns*-2. This information allows us to draw conclusions about design decisions made in TCP-Vegas and TCP-SACK.

2 Methodology

In this section, we describe our framework of modeling the performance of TCP flavors. This framework was first introduced by Casetti and Meo [9, 10]. We extend their framework so that the performance of TCP-Vegas can be modeled. This extension also enables the modeling of more general network and sender behavior.

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 [9, 10], only the loss rate is analyzed, but we require the delay distribution for modeling TCP-Vegas. Aspects such as network topology, queueing capacity, link capacity, packet arrival pattern, queue management scheme (DropTail, RED, class-based RED, etc.), and the network environment (fixed links or Wireless transmission) are taken into account within the queuing model.

Although the two components are analyzed separately, they interact with each other via feedback. The parameters in the source models and the network model 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

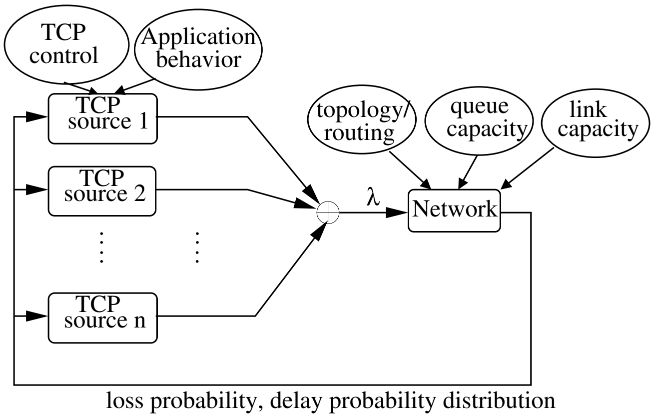


Figure 1: *TCP analysis methodology*

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 stable 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 stable solution is usually found within $10 - 15$ iterations. In the rest of this section, we describe the source model and the network model in 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. Note that many of the transitions and states are omitted for clarity.

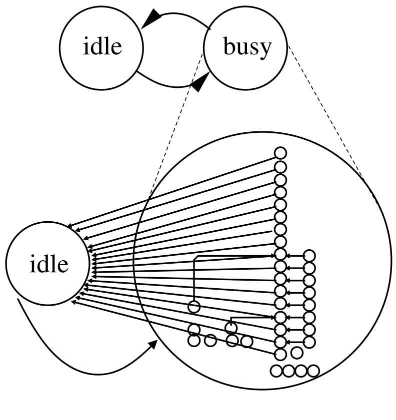


Figure 2: *Model of a TCP source*

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.

A few extensions to this application level model follow. Though the length of each busy and idle duration is assumed to be exponentially distributed in [9, 10], we note that the length of the busy and idle duration can be easily generalized to any phase-type (PH) distributions. This enables us to model the highly variable file lengths observed in real world Internet traffic.[1] Although these extensions provide interesting topics for future work, the focus of the current article is on extending the TCP level and network level models; so we have chosen to work with the simple application model described.

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, assuming that a TCP source keeps sending segments at a speed determined by its congestion window size.

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 aggregated throughput λ 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 λ_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 [10] 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 3.2 is novel and shows a nontrivial extension of the original model of TCP-Reno. In Section 3.5, we validate our model using *ns*-2 simulations.

[1] Feldmann and Whitt have proposed an algorithm for fitting heavy tailed distributions by PH distributions in [12].

Before describing the details of the model of 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 other information such as the slow-start threshold and whether we need to recover from loss or not. 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. Throughout this paper, we approximate the RTT, or the duration of staying at each state, by an exponential distribution (by matching the mean), but this approximation can be generalized to any PH distribution. 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.

We now describe the TCP-Reno source model introduced in [10] 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

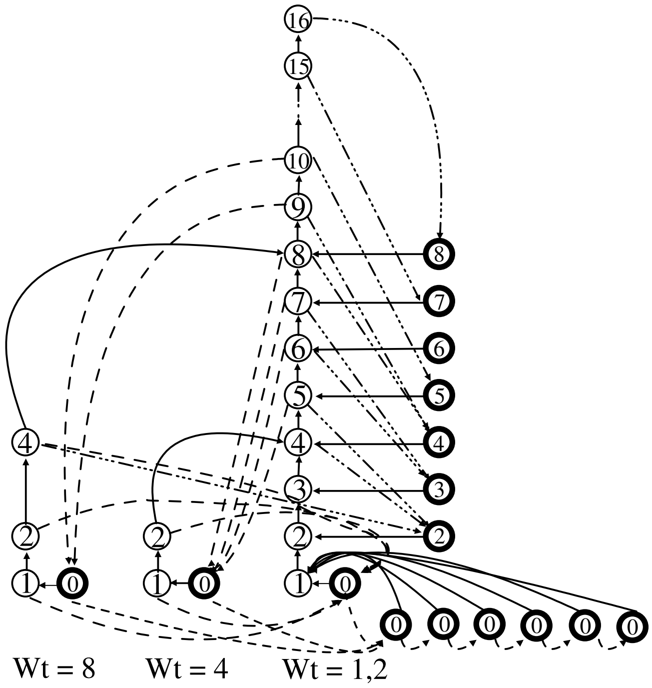


Figure 3: *Model of a TCP-Reno window model for* $W_M = 16$.

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 [10]. The first extension is the addition of the exponential backoff mechanism in TCP. The second extension is the addition of the 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 extension is shown in Figure 6 (Section 3.5). Also, although the TCP models in this paper restart each busy period with the same initial window size of one packet, the model is easily extendable to allow the connections to maintain both the window size and slow-start threshold between busy periods to better reflect actual TCP behavior for traffic with short off-times. In fact, this extension was performed in the case of TCP-Tahoe in [9].

We now define the transition intensities of the TCP-Reno Markov chain. The transition intensities are determined by the loss rate, P, and the mean RTT, θ, based on the TCP protocol specification. For states where transitions can take place to many other states, the transition intensities 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 intensity 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 intensity 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 intensity 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 intensity $P^R_{\text{fr/fr}}\,\theta^{-1}$, and to timeout $(0,1,1)$ with intensity $P^R_{\text{to}}\,\theta^{-1}$, where $P^R_{\text{fr/fr}}$ is the probability of fast retransmit/fast recovery and P^R_{to} 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 intensity θ^{-1}, from exponential back-off state k ($k = 0$ corresponds to single timeout) back to slow-start with intensity $P_1(0)\,(2^k T)^{-1}$, and from exponential back-off to the next exponential back-off state with intensity $(1 - P_1(0))\,(2^k T)^{-1}$.

Finally, we describe how to obtain the three parameters: the probability of fast retransmit/fast recovery $P^R_{\text{fr/fr}}$, the probability of timeout P^R_{to}, and the mean duration of timeout T. Using the study in [11], $P^R_{\text{fr/fr}}$ and P^R_{to} can be approximately quantified for different congestion window sizes w:

$$P^{\text{R}}_{\text{to}} = \begin{cases} \sum_{i=3}^{w} P_w(i) & \text{if } w \geq 10 \\ \sum_{i=2}^{w} P_w(i) & \text{if } 4 \leq w < 10 \\ 1 - P_w(0) & \text{if } w < 4 \end{cases} \quad \text{and} \quad P^{\text{R}}_{\text{fr/fr}} = \begin{cases} P_w(1) + P_w(2) & \text{if } w \geq 10 \\ P_w(1) & \text{if } 4 \leq w < 10 \\ 0 & \text{if } w < 4. \end{cases}$$

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, θ, its standard deviation, σ, 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 [11], 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 a larger number of packets are lost, TCP-Reno will be forced to timeout and transit will be to the appropriate timeout state.

Therefore, the structure of the Markov chain for TCP-SACK remains the same as the Markov chain for TCP-Reno, and only the transition intensities differ as follows. 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

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

and the probability for fast retransmit/fast recovery as

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

Hence, the transition intensities for the TCP-SACK Markov chain are obtained by modifying the transitions to fast retransmit/fast recovery to $P^{S}_{\mathrm{fr/fr}}(w)\,\theta^{-1}$, and to timeout with intensity $P^{S}_{\mathrm{to}}(w)\,\theta^{-1}$, for all $w \geq 4$.

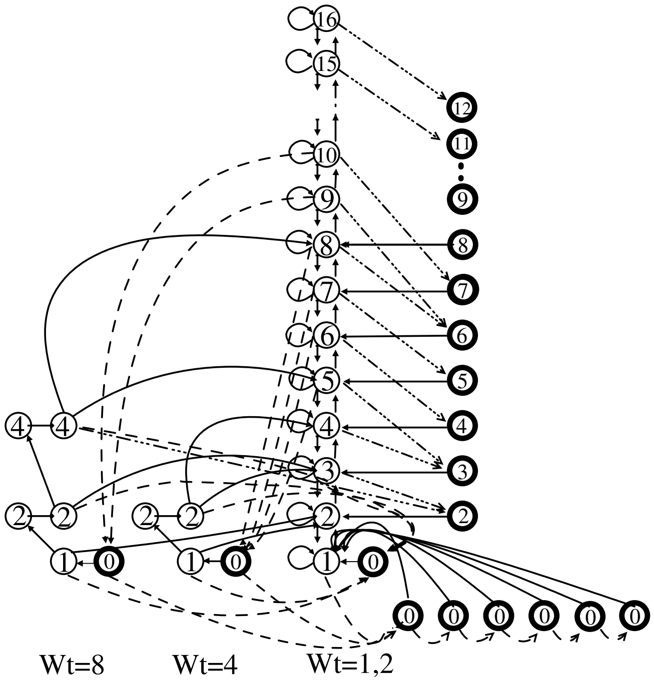


Figure 4: *TCP-Vegas window size transition for* $W_M = 16$.

3.3 Modeling TCP-Vegas

The core features introduced by TCP-Vegas in [7] 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.

3.3.1 Novel TCP-Vegas mechanisms

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, Θ is the round trip time, and Θ_{min} is the smallest RTT seen over the connection. Notice that $\Theta - \Theta_{\min}$ is the total queueing delay and w/Θ 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.[2] First, TCP-Vegas tries to keep the estimated number, N_b, of backlogged packets between two thresholds, α and β, where $\alpha \leq \beta$. If $N_b < \alpha$, TCP-Vegas concludes that there are *too few* packets at the bottleneck queue, and 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 to observe the delay seen by packets at each window size. 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.

3.3.2 TCP-Vegas Markov chain

A Markov chain for TCP-Vegas is shown in Figure 4. This chain represents a large extension to the work in [10] 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 intensity during slow-start are added. Second, the possibility for TCP-Vegas to exit slow start early if the queried network delay is too high are 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 Θ_{min} by the propagation delay D_p and Θ 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

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

The transition intensities 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 intensities from the slow-start and congestion avoidance states to the fast retransmit/fast recovery and

[2] TCP-Vegas is sometimes defined to use a large initial window size of two packets, instead of the standard one packet. We leave the initial window size as one in order to maintain comparability to the models of TCP-SACK and TCP-Reno described in the previous sections. However, modeling this increased initial window size is a trivial extension to the model described.

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 intensity $P(N_b \leq \gamma)\, P_w(0)\, \theta^{-1}$, from which it transitions to the state with double window size with intensity $P(N_b \leq \gamma)\, P_w(0)\, \theta^{-1}$. When the window size is greater than $\lfloor \frac{W_t}{2} \rfloor$, TCP-Vegas transitions to an intermediate state with intensity $P(N_b > \gamma)\, P_w(0)\, \theta^{-1}$, from which it transitions to the congestion avoidance phase (window size $w+1$) with intensity $P(N_b > \gamma)\, P_w(0)\, \theta^{-1}$. During congestion avoidance, for congestion window sizes $1 < w < W_M$, the delay based congestion avoidance mechanism leads to window increase with intensity $P(N_b < \alpha)\, P_w(0)\, \theta^{-1}$, to window decrease with intensity $P(N_b > \beta)\, P_w(0)\, \theta^{-1}$ and remains in the current state with intensity $P(\alpha \leq N_b \leq \beta)\, P_w(0)\, \theta^{-1}$. 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 intensity $P(N_b \leq \beta)\, P_{W_M}(0)\, \theta^{-1}$ and decreases the window size with intensity $P(N_b > \beta)\, P_{W_M}(0)\, \theta^{-1}$. For the minimum window size $w = 1$, TCP-Vegas remains in the current state with intensity $P(N_b > \alpha)\, P_1(0)\, \theta^{-1}$ and increases the window size with intensity $P(N_b < \alpha)\, P_1(0)\, \theta^{-1}$.

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 [26].[3] Using $\vec{\pi}$, the intensity λ of the throughput is calculated as:

$$\lambda = \sum_{i:\text{all active states}} w(i) \vec{\pi}(i),$$

where $w(i)$ is the window size of state i, and all active states consist of all slow-start states and congestion avoidance states.

The size of the state space directly affects the computational complexity of solving the Markov chain. In the Markov chains for TCP-Reno and TCP-SACK, the number of states, $N_{\text{Reno}}(W_M)$ and $N_{\text{Sack}}(W_M)$, including the idle state, are:

$$N_{\text{Reno}}(W_M) = N_{\text{Sack}}(W_M) = W_M + \lfloor \frac{W_M}{2} \rfloor + \frac{1}{2} \lceil \log_2 W_M \rceil \left(\lceil \log_2 W_M \rceil + 1 \right) + 5,$$

and the number of states, $N_{\text{Vegas}}(W_M)$, in the Markov chain for TCP-Vegas is:

$$N_{\text{Vegas}}(W_M) = 2W_M - \lfloor \frac{W_M}{4} \rfloor + \lceil \log_2 W_M \rceil (\lceil \log_2 W_M \rceil - 1) + 6.$$

In particular, when $W_M = 16$, $N_{\text{Reno}}(16) = 38$ and $N_{\text{Vegas}}(16) = 45$, and when $W_M = 21$, $W_M = 21$, $N_{\text{Reno}}(21) = 50$ and $N_{\text{Vegas}}(21) = 62$. These state spaces are small enough and allow us to evaluate the Markov chain iteratively in our framework.

[3] It should be noted that this procedure is more complex when the loss rate is very low. When the loss rate is very low, some of the transition intensities (such as transition to the timeout) become 0 due to numerical precision. These states must be identified and removed before solving the Markov chain. This is accomplished by finding the largest strongly connected component of the chain that is connected to the idle state, and then reducing the Markov state space to include only states in this component.

3.5 Validation of source models

The separation between the network and the source provided by this framework allows an independent validation of our source models. This is accomplished by comparing with *ns*-2 results in the following manner. First, an *ns*-2 simulation is run to calculate the loss rate and throughput under a particular network setting. Then, using the loss rate calculated by *ns*-2 as an input to our source model, our source model provides an estimated throughput, which we can verify with ns independently of the details of the network model. Note that in the validation of our source models no iteration is necessary.

In this manner, we validate the TCP-Reno and TCP-SACK models (TCP-Vegas requires the delay distribution) and observe that they give very accurate predictions when the loss rate of the network is known. Due to constrained space, we do not include these plots in this report; they are superseded by the validation of the complete model in Section 5.2, where we show that our model is capable of accurately predicting the operating point of the network in terms of throughput and loss rate.

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 relative merits. We focus on a single bottleneck network with a DropTail queue. In analytical modeling, it is quite 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 $M/D/1/B$. Thus, 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 [29], although it does underestimate the delay seen on the link. In addition, Casetti and Meo use an $M/M/1/B$ queue as their network model in the case of TCP-Reno [10].

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.

We include formulas for the mean queueing delay, the loss rate, and the delay distribution under each of these queueing models in Appendix A.

4.1 Validation of network models

The network models can be validated independently of the source models in a similar way to the validation of the source models in Section 3.5. This is accomplished by comparing with *ns*-2 results in the following manner. First, an *ns*-2 simulation is run to calculate the loss rate and throughput under a particular setting. Then, using the throughput calculated by *ns*-2 as an input to our network model, we can verify the performance of the network model independently of the details of the source model. 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 [27], which investigates the applicability of various queueing models both for heterogeneous and homogeneous TCP-NewReno sources under similar settings to this paper. It is shown that under high loss ($> 5\%$) environments, any of these queueing models predict the loss rate equally well. However, under low loss ($< 5\%$) environments, the best queueing model depends on the type of transfers by TCP sources, i.e. persistent or transient. It is shown that $M/D/1/B$ queues best predict the loss rate for transient sources. However, it is also shown that for sources with a slightly longer on and off periods, $M/M/1/B$ queues best predict the loss rate, and for (homogeneous) persistent sources, $M^r/M/1/B$ queues best predict the loss rate. An intuition behind this observation is that the packet size distribution is best approximated by a deterministic distribution, and the arrival process is well approximated by Poisson process under transient connections. However, under persistent connections, the arrival process is better approximated by batch Poisson process due to the traffic burstiness stemming from the TCP slow-start and source synchronization effect.

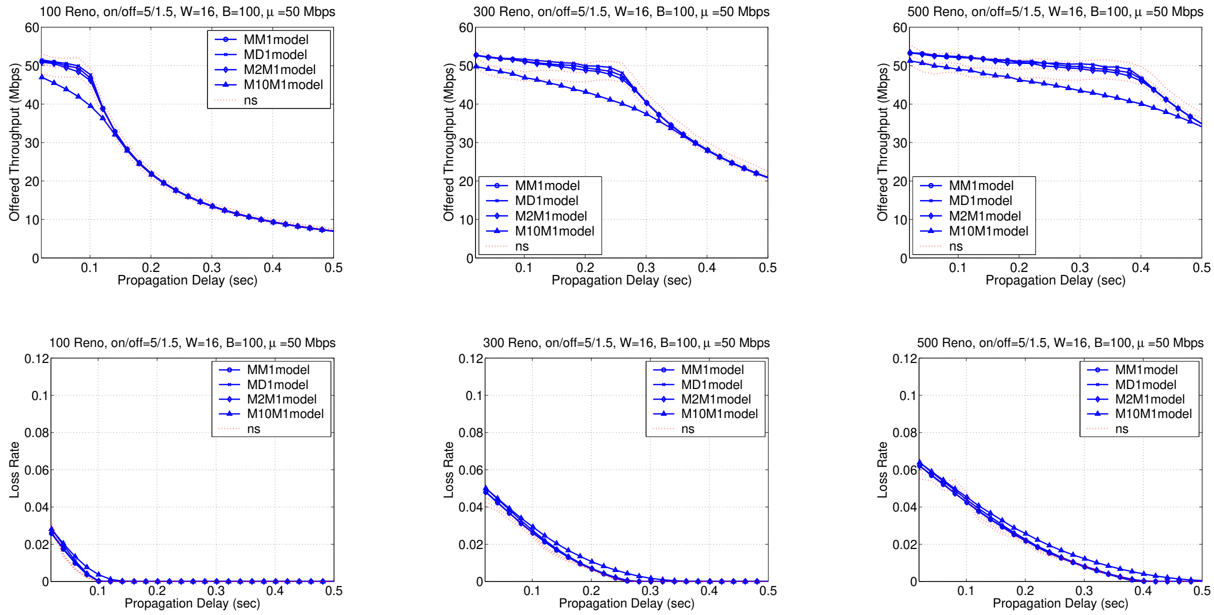
5 Full model validation

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.

5.1 Validation setup

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

Validation against *ns*-2 simulation: TCP-Reno



(a) 100 sources (b) 300 sources (c) 500 sources

Figure 5: *Comparison of ns-2 and Markov model for TCP-RENO with respect to offered traffic (row 1) and loss rate (row 2) when TCP sources with transient traffic (On/Off =* $5/1.5$*) share* 50 *Mbps core network. (a)* 100 *TCP sources; (b)* 300 *TCP sources; (c)* 500 *TCP sources.*

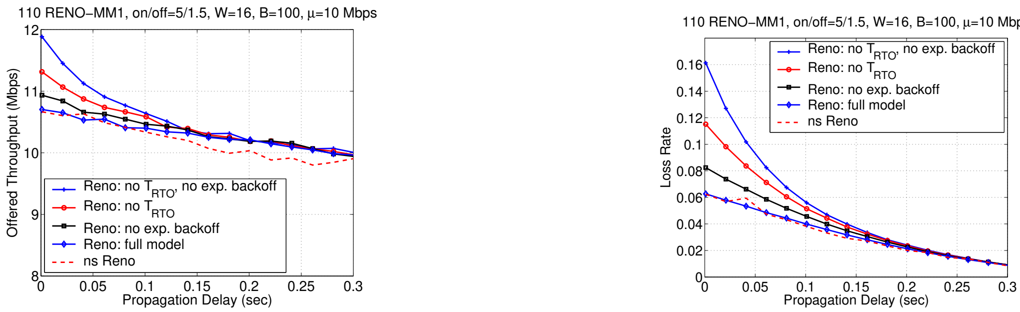
range of propagation delays, capacities, and buffer sizes are tested for the bottleneck link. Each source uses exponential duration both for the on state (mean T_{on}), and the off state (mean T_{off}). For each on period, the sources start with an initial window size of 1 packet, a slow-start threshold 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 μ 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. In particular, when the bottleneck capacity is 10 Mbps and the buffer size is 100, varying propagation delay between 0.02 and 0.50 seconds results in varying the maximum queueing delay between $8\ \%$ and $66\ \%$ of the total round trip time. 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: $T_{\text{on}}/T_{\text{off}}$ periods of $1.5/0.5$ and $5/1.5$ seconds), semi-persistent sources ($T_{\text{on}}/T_{\text{off}}$ periods of $50/5$ seconds), and persistent sources ($T_{\text{on}}/T_{\text{off}}$ periods of $10^6/1$ 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.

5.2 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. We will evaluate all queueing models in the TCP-Reno validation in Section 5.2.1. 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 model in Section 5.2.2 and the TCP-Vegas model in Section 5.2.3 using the $M/M/1/B$.

Improvement of the model due to exponential backoff and T_{RTO}



(a) throughput

(b) loss rate

Figure 6: *The effect of adding exponential backoff and the* T_{RTO} *parameter is analyzed. Comparison of Markov models for TCP-Reno with respect to (a) offered traffic and (b) loss rate.* 110 *TCP sources with transient traffic (On/Off* = $5/1.5$*) share* 10 *Mbps core network.*

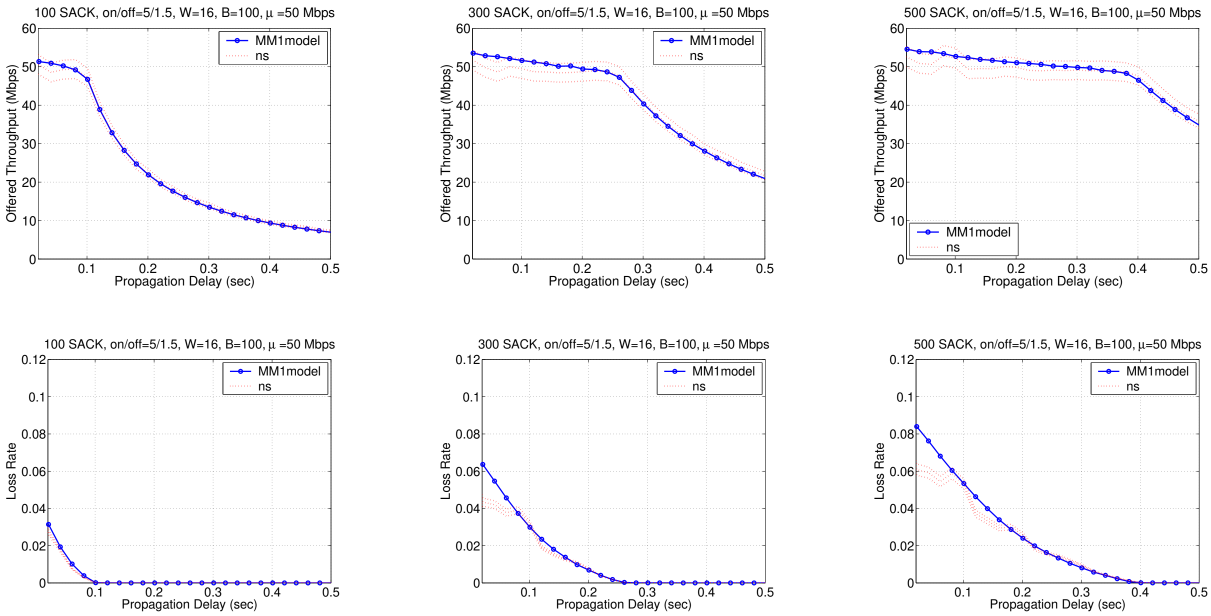
5.2.1 TCP-Reno validation

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 an intermediate number of TCP sources: a 50 Mbps bottleneck link with buffer size 100 shared by 300 transient TCP-Reno sources. Column (c) shows the results under a large number of TCP sources: a 50 Mbps bottleneck link with buffer size 100 shared by 500 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 ± 5 percent interval around the simulated values. The solid lines correspond to the $M/M/1/B$, $M/D/1/B$, $M^2/M/1/B$ 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 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 [10]. 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 under high loss scenarios, but the model with these extensions accurately predicts both the throughput and the loss rate under any scenarios.

Validation against *ns*-2 simulation: TCP-SACK



(a) 100 sources (b) 300 sources (c) 500 sources

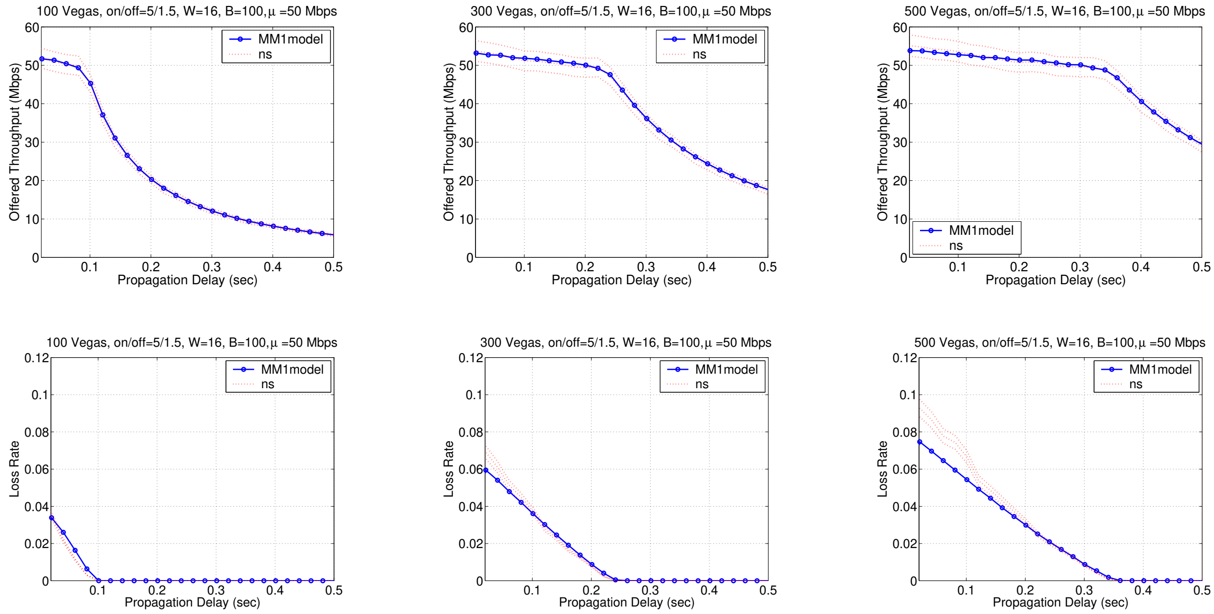
Figure 7: *Comparison of ns-2 and Markov model for TCP-SACK with respect to offered traffic (row 1) and loss rate (row 2) when TCP sources with transient traffic (On/Off =* $5/1.5$*) share* 50 *Mbps core network. (a)* 100 *TCP sources; (b)* 300 *TCP sources; (c)* 500 *TCP sources.*

5.2.2 TCP-SACK validation

Figure 7 validates the TCP-SACK model against *ns*-2 simulation. 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-SACK sources. Column (b) shows the results under an intermediate number of sources: a 50 Mbps bottleneck link with buffer size 100 shared by 300 transient TCP-SACK sources. Column (c) shows the results under a large number of sources: a 50 Mbps bottleneck link with buffer size 100 shared by 500 transient TCP-SACK sources. The model estimation of throughput and loss rate is within the ± 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.

Validation against *ns*-2 simulation: TCP-Vegas — Transient sources: On/Off = $5/1.5$



(a) 100 sources (b) 300 sources (c) 500 sources

Figure 8: *Comparison of ns-2 and Markov model for TCP-Vegas with respect to offered traffic (row 1) and loss rate (row 2) when TCP sources with transient traffic (On/Off =* $5/1.5$*) share* 50 *Mbps core network. (a)* 100 *TCP sources; (b)* 300 *TCP sources; (c)* 500 *TCP sources.*

5.2.3 TCP-Vegas validation

Now we validate the TCP-Vegas model and compare it to an existing analytical model. Following the recommendations in [33], the TCP-Vegas parameters are configured to $\alpha = \beta = \gamma = 2$ which improves fairness.[4]

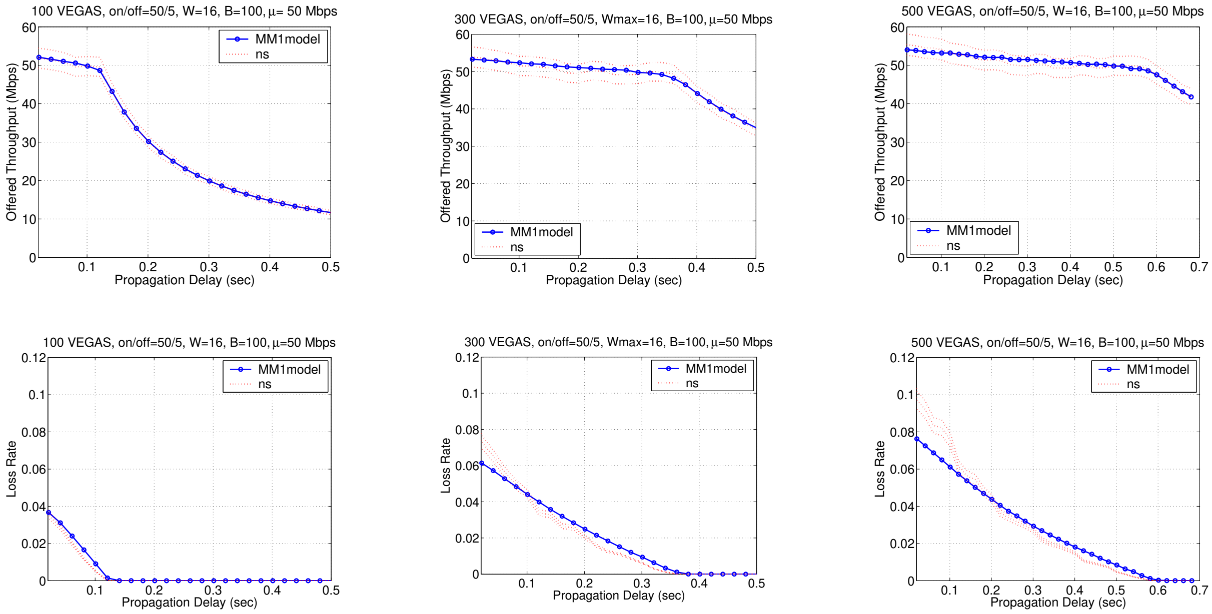
Figures 8 and 9 validate the TCP-Vegas model against *ns*-2 simulation. Figure 8 shows the results under transient sources ($T_{\text{on}} = 5$ and $T_{\text{off}} = 1.5$), and Figure 8 shows the results under semi-persistent sources ($T_{\text{on}} = 50$ and $T_{\text{off}} = 5$). In both figures, column (a) shows the results under a small number of sources, column (b) shows the results under an intermediate number of sources, and column (c) shows the results under a large number of sources. 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 TCP-Vegas is even better than the accuracy achieved earlier by the TCP-Reno and TCP-SACK models.[5]

Now, we compare the performance of our TCP-Vegas model to the performance of the model proposed by Samios and Vernon [33]. Although the Samios and Vernon model is the most general among all the prior analytical TCP-Vegas models, their work is still restricted to the situation of a single source performing a bulk

[4]Fairness here refers to maintaining similar throughput across connections with varying propagation delays.

[5]The accuracy of the model might be due the TCP-Vegas protocol causing a different queueing distribution at the bottleneck link that is more compatible with the $M/M/1/B$ queueing model.

Validation against *ns*-2 simulation: TCP-Vegas — Semi-persistent sources: On/Off = $50/5$



(a) 100 sources (b) 300 sources (c) 500 sources

Figure 9: *Comparison of ns-2 and Markov model for TCP-Vegas with respect to offered traffic (row 1) and loss rate (row 2) when TCP sources with semi-persistent traffic (On/Off =* $50/5$*) share* 50 *Mbps core network. (a)* 100 *TCP sources; (b)* 300 *TCP sources; (c)* 500 *TCP sources.*

transfer in a network having a known loss rate. The model we present applies in much more general situations. Our model allows transient connections, semi-persistent connections, and bulk transfers as well as modeling the performance of an arbitrary number of TCP-Vegas sources. Further, our model is able to predict the operating point of the network, instead of depending on an input of the achieved network loss rate. Thus, we can compare our model to existing models, although not across the full generality of our model.

This comparison is achieved in the following manner. We first select a network topology: in Figure 10 we fix the bottleneck capacity to 10 Mbps, the buffer size to 100, and the propagation delay to 0.15 seconds. We then use *ns*-2 and let N sources perform persistent transfers and record the observed loss rate together with offered traffic divided by the number of sources as an estimate of the offered throughput per source. The recorded loss rate and average queueing delay are used to fix the parameters of our network model, which is then input to our source model to obtain the throughput. Finally the throughput observed in the *ns*-2 simulation and the throughput predicted by our model are compared.

Figure 10 shows the result of the comparison to the Samios and Vernon model. With respect to accuracy, our model shows significant improvement over the Samios and Vernon model under high loss environments. With respect to computational complexity, the closed form solution of the Samios and Vernon model is superior to ours; however, notice that when the loss rate is given as an input, obtaining the throughput by our model does not require any iteration. All that is needed is to solve a small (< 100 states) source model Markov chain only once. Further, the extra computational complexity in our model allows it to be applicable in much more general

Validation: TCP-Vegas — Persistent sources

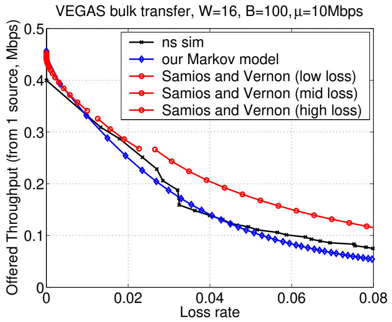


Figure 10: *Comparison of ns-2, Markov model, and Samios/Vernon model with a* 10 *Mbps core network, persistent connections and a propagation delay of* 0.15 *seconds.*

network situations.

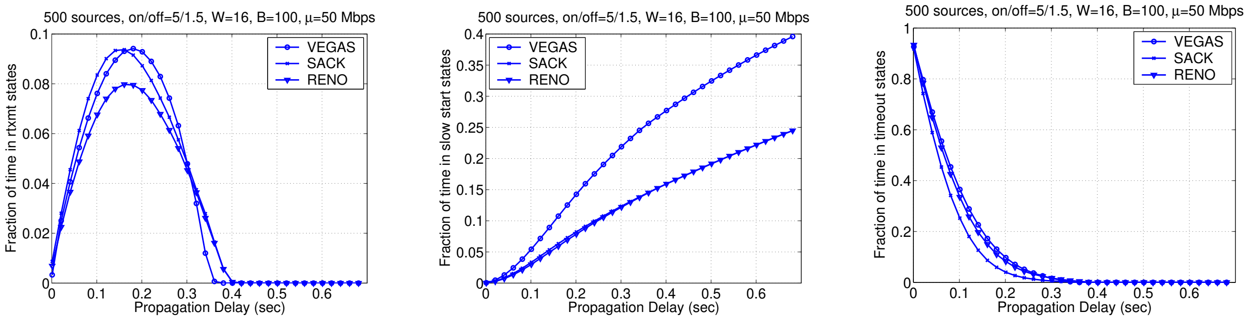
6 Comparison of TCP versions

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 11 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 10 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 can observe that selective acknowledgements help avoid timeouts by increasing the time spent in fast retransmit states over all propagation delays (Figure 11 (a)). Second, the modified slow start mechanism in TCP-Vegas does not help avoid future losses or time spent in timeout states, but just results in wasting 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 significantly greater than the time TCP-Reno spends in slow start states (Figure 11 (b)), and that the time TCP-Vegas spends in timeout states is almost the same as the time TCP-Reno spends in timeout states (Figure 11 (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 [10]. 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



(a) fast retransmit states (b) slow-start states (c) timeout states

Figure 11: *Comparison of time spent (a) in fast retransmit states, (b) in slow-start states, and (c) in timeout states for TCP-Reno, TCP-SACK and TCP-Vegas.*

model previously proposed for TCP-Reno. Analytical models of TCP-Vegas have previously been limited to bulk transfer and our framework allows the first analytical modeling of TCP-Vegas under on-off traffic. Our work also improves of the accuracy of modeling TCP-Reno in this framework by adding an exponential backoff mechanism and a 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 counter intuitive 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 wasting 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, this can be generalized to an arbitrary network topology connected by heterogeneous TCP sources (different TCP flavors, propagation delays, etc.). Also, the distribution of sizes 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 as approximations. Furthermore, the arrival process to the network is assumed to be Poisson, but this can be extended to a Markovian Arrival Processes (MAPs).

References

[1] S. Alouf. *Parameter estimation and performance analysis of several network applications*. PhD thesis, Department of Sciences, University of Nice Sophia-Antipolis, France, November 2002.

[2] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of TCP/IP with stationary random losses. *ACM SIGCOMM Computer Communication Review*, 30:231 – 242, 2000.

[3] E. Altman, F. Boccara, J. Bolot, F. Nain, P. Brown, D. Collange, and C. Fenzy. Analysis of the TCP/IP flow control mechanism in high-speen wide-area networks. In *Proceedings of* 34th IEEE Conference on Decision and Control, pages 368 – 373, December 1995.

[4] Å. Arvidsson and A. Krzesinski. The design of optimal multi-service MPLS networks. In *Proceedings of* 10th International Telecommunication Network Strategy and Planning Symposium, pages 31 – 40, June 2001.

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

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

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

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

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

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

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

[12] A. Feldmann and W. Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Performance Evaluation*, 32:245–279, 1998.

[13] B. Firoiu. A study of active queue management for congestion control. In *Proceedings of* IEEE INFOCOM, pages 1435 – 1444, March 2000.

[14] M. Garetto, R. L. Cigno, M. Meo, E. Alessio, and M. A. Marsan. Modeling short-lived TCP connections with open multiclass queuing networks. In *Proceedings of* 7th International Workshop on Protocols For High-Speed Networks (PfHSN), pages 100 – 116, April 2002.

[15] M. Garetto, R. L. Cigno, M. Meo, and M. A. Marsan. A detailed and accurate closed queueing network model of many interacting TCP flows. In *Proceedings of* IEEE INFOCOM, pages 1706 – 1715, April 2001.

[16] M. Garetto, R. L. Cigno, M. Meo, and M. A. Marsan. On the use of queueing network models to predict the performance of TCP connections. In *Proceedings of* Tyrrenian International Workshop on Digital Communications (IWDC), pages 536 – 555, September 2001.

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

[18] I.Kaj and J. Olsén. Throughput modeling and simulation for single connection tcp-tahoe. In *Proceedings of* the 17th International Teletraffic Congress ITC-17, pages 705–718, December 2001.

[19] I. Kaj and J. Olsén. Stochastic equilibrium modeling of the TCP dynamics in various AQM environments. In *Proceedings of* SPECTS, pages 481–493, July 2002.

[20] 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.

[21] S. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans./ Networking*, Oct. 2003 (to appear).

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

[23] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review*, 27:67 – 82, 1997.

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

[25] J. Mo, R. J. La, V. Anantharam, and J. C. Walrand. Analysis and comparison of TCP Reno and Vegas. In *Proceedings of* IEEE INFOCOM, pages 1556 – 1563, March 1999.

[26] J. Norris. *Markov Chains*. Cambridge University Press, 1997.

[27] J. Olsén. On packet loss inferences used for TCP network modeling. Technical Report U.U.D.M. Report 2003, Uppsala University, 2003 (to appear).

[28] 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.

[29] 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 *Proceedings of* IEEE INFOCOM, pages 535 – 544, June 2002.

[30] M. Roughan, A. Erramilli, and D. Veitch. Network performance for TCP networks part I: Persistent sources. In *Proceedings of* the International Teletraffic Congress-ITC-17, pages 24 – 28, September 2001.

[31] R. Roy, R. Mudumbai, and S. Panwar. Analysis of TCP congestion control using a fluid model. In *Proceedings of* IEEE International Conference on Communications, pages 2396 – 2403, June 2001.

[32] S. M. S. Floyd. *ns-LBL Network Simulator*, 1997.

[33] C. Samios and M. Vernon. Modeling the throughput of TCP Vegas. In *Proceedings of* ACM Sigmetrics, June 2003 (to appear).

A Analysis of network models

In this section, we show well known formula's for the performance of the three queueing models we apply. We define λ to be the aggregated arrival rate from the TCP sources, B to be the buffer size of the bottleneck link, and μ to be the bottleneck link speed. We also define the load $\rho \stackrel{\text{def}}{=} \lambda/\mu$.

$\mathbf{M/D/1/B}$ From the analysis in [1], the loss rate $p^{M/D/1/B}$ of an $M/D/1/B$ queue is given by:

$$p^{M/D/1/B} = \frac{1 + (\rho - 1)\alpha_B(\rho)}{1 + \rho\alpha_B(\rho)},$$

where

$$\alpha_j(\rho) = \sum_{k=0}^{j-2} \frac{e^{\rho(j-k-1)}(-1)^k \rho^k (j-k-1)^k}{k!}, \quad j \geq 2,$$

and the delay distribution $Pr(D_q \leq t)$ is given by:

$$Pr(D_q \leq t) = \sum_{k=0}^{\min(\lfloor t\mu/S-1 \rfloor, B-1)} \frac{\pi_k}{(1-\pi_B)},$$

where

$$\begin{aligned}\pi_0 &= \frac{1}{1+\rho\alpha_B(\rho)},\\ \pi_1 &= \frac{\alpha_2(\rho)-1}{1+\rho\alpha_B(\rho)},\\ \pi_j &= \frac{\alpha_{j+1}(\rho)-\alpha_j(\rho)}{1+\rho\alpha_B(\rho)}, \quad (j=2,\ldots,B-1),\\ \pi_B &= \frac{1+(\rho-1)\alpha_B(\rho)}{1+\rho\alpha_B(\rho)}.\end{aligned}$$

Notice that α_j's are quite complicated and, in fact, it is quite difficult to evaluate the loss rate by maintaining the necessary numerical precision when the buffer size is large. The formula for the delay distribution has a similar problem. We use Maple, which can specify an arbitrary precision, to evaluate the loss rate and delay distribution.

$\mathbf{M/M/1/B}$ The performance of an $M/M/1/B$ is well known and the loss rate $p^{M/M/1/B}$, the expected delay $E[D_q]^{M/M/1/B}$, and the delay distribution $Pr(D_q \le t)^{M/M/1/B}$ are given by:

$$\begin{aligned}p^{M/M/1/B} &= \frac{\rho^B(1-\rho)}{1-\rho^{B+1}},\\ E[D_q]^{M/M/1/B} &= \left(\frac{1}{\mu}\right)\frac{1-(B+1)\rho^B+B\rho^{B+1}}{(1-\rho)(1-\rho^B)},\\ Pr(D_q \le t)^{M/M/1/B} &= 1-\frac{e^{-\mu t}}{1-\rho^B}\sum_{i=0}^{B-1}\frac{(\lambda t)^i}{i!}+\frac{\rho^B e^{-\mu t}}{1-\rho^B}\sum_{i=0}^{B-1}\frac{(\mu t)^i}{i!}.\end{aligned}$$

$\mathbf{M^r/M/1/B}$ The loss rate $p^{M^r/M/1}$ of an $M^r/M/1/B$ is:

$$p^{M^r/M/1} = \sum_{j=1}^{r}\frac{j}{r}\pi_{B-r+j},$$

where the π_i's are solved iteratively using the balance equations:

$$\begin{aligned}\lambda\pi_0 &= \mu\pi_1\\ (\lambda+\mu)\pi_k &= \mu\pi_{k+1} \quad 1\le k<r\\ (\lambda+\mu)\pi_k &= \mu\pi_{k+1}+\pi_{k-r}\lambda, \quad r\le k<B-1\\ \mu\pi_B &= \sum_{j=B-r}^{B-1}\lambda\pi_j\\ 1 &= \pi_0+\pi_1+\cdots+\pi_B.\end{aligned}$$

Using the Little's formula, the expected delay can be computed from the π_i's. In Section 5, we conclude that the $M^r/M/1/B$ queue is not as appropriate for our purposes, and we do not apply this model to TCP-Vegas. As a result, we do not include a formula for the delay distribution under this network model.