Modeling the Throughput of TCP Vegas

Charalampos (Babis) Samios
Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706
babis@cs.wisc.edu

Mary K. Vernon
Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706
vernon@cs.wisc.edu

ABSTRACT

Previous analytic models of TCP Vegas throughput have been developed for loss-free (all-Vegas) networks. This work develops a simple and accurate analytic model for the throughput of a TCP Vegas bulk transfer in the presence of packet loss, as a function of average round trip time, minimum round trip time, and loss rate for the transfer. Similar models have previously been developed for TCP Reno. However, several aspects of TCP Vegas need to be treated differently than their counterparts in Reno. The proposed model captures the key innovative mechanisms that Vegas employs during slow start, congestion avoidance, and congestion recovery. The results include (1) a simple, validated model of TCP Vegas throughput that can be used for equation-based rate control of other flows such as UDP streams, (2) a simple formula to determine, from the measured packet loss rate, whether the network buffers are overcommitted and thus the TCP Vegas flow cannot reach the specified target lower threshold on throughput, (3) new insights into the design and performance of TCP Vegas, and (4) comparisons between TCP Vegas and TCP Reno including new insights regarding incremental deployment of TCP Vegas.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Performance, Experimentation, Design

Keywords

TCP, TCP Vegas, Performance Model, Throughput

1. INTRODUCTION

Recently researchers have proposed a number of analytic models of the throughput of a single TCP flow as a function of round-trip-time (RTT) and packet loss rate. These models have provided improved understanding of the sensitivity of TCP performance to these network parameters, and have also been used in proposed approaches for controlling the rate of other types of Internet flows such as UDP streams e.g. [9, 23]. All of these models address the most widely deployed variant of TCP, namely TCP Reno (e.g.,[8, 20, 22, 19, 12]).

Another variant of TCP that has been proposed is TCP Vegas [6, 7]. Vegas employs several new techniques that, together, can result in significant improvement in throughput as well as decreased packet loss [6, 2]. Some of these improvements have recently been implemented, in some cases using alternate mechanisms, in other forms of TCP. For example, like Vegas, TCP New-Reno [10] only reduces the window size once when multiple packets are dropped from the same window, whereas TCP Reno reduces the window size for each triple-duplicate ACK that is received. Some of the other innovations are still not well understood and have so far not been widely deployed. For example, Vegas' congestion avoidance algorithm has some key advantages in terms of avoiding packet loss as well as reducing bias against connections with longer propagation delays.

The performance of TCP Vegas in complex network environments that include interaction with other types of flows, is not thoroughly understood. Recent studies have used simulation or analytic models of Vegas behavior in the absence of losses to study some of these issues. An analytic model of the throughput of a TCP Vegas flow in the presence of packet losses that might be caused by sharing the network with other types of flows – which has not, to our knowledge, been previously proposed – can also be an important tool in understanding the protocol performance and mechanisms.

Several aspects of TCP Vegas need to be treated quite differently from their counterparts in Reno. These include the congestion detection and avoidance algorithm that preemptively adjusts the sending rate to avoid packet loss, and the new congestion recovery mechanisms. To capture these features of the protocol, we partition the flow into statistically equivalent time intervals, and derive a closed-form solution for the throughput of a random such interval. Loss indications in the form of both duplicate ACKs and timeouts are modeled, along with the impact of the maximum window size.

The model is developed gradually by incorporating a new set of Vegas mechanisms at a time. This provides the opportunity to examine the intuition behind the different mechanisms employed by Vegas by characterizing them analytically. Also, we derive a closed form expression to determine, from the measured packet loss probability, whether the TCP Vegas flow can achieve the specified target lower threshold on throughput.

We conducted a large number of simulation experiments using ns-2 [1] to validate our model against a wide range of network conditions, and to examine TCP Vegas behavior under network conditions that haven't been explored previously. New simulation results regarding the relative performance of TCP Vegas and TCP Reno

SIGMETRICS'03, June 10–14, 2003, San Diego,California, USA.

are also presented, yielding new insights regarding the incremental deployment of TCP Vegas.

The rest of the paper is organized as follows. In Section 2 we outline the innovative mechanisms employed in TCP Vegas and summarize related work. The model is developed in four stages in Section 3. Section 4 presents the model validation and the other experimental results, and Section 5 concludes the paper, including topics for future work.

2. BACKGROUND

2.1 TCP Vegas

This section briefly reviews the innovations of TCP Vegas with respect to TCP Reno that are most relevant to developing the throughput model. The first important aspect is the Vegas congestion avoidance mechanism, which differs significantly from TCP Reno. TCP Reno uses the loss of packets as a signal that there is congestion in the network. In fact, Reno needs to *create* losses to find the available bandwidth of the connection. In contrast, the goal of Vegas is to pro-actively detect congestion in its incipient stages, and then reduce throughput in an attempt to *prevent* the occurrence of packet losses.

To detect network congestion, once every round trip time (RTT), TCP Vegas uses the current window size (W), the most recent RTT (RTT) and the minimum RTT observed so far ($baseRTT$) to compute:

$$diff = \left(\frac{W}{baseRTT} - \frac{W}{RTT}\right) baseRTT = W\frac{RTT - baseRTT}{RTT}. \quad (1)$$

Since $(RTT - baseRTT)$ is the total path queueing delay and W/RTT is an estimate of the current throughput, the product of these two values is an estimate of the number of packets from this flow that are backlogged in the network. The goal of the Vegas congestion avoidance algorithm is to keep this number within a fixed range defined by two thresholds, α and β. Thus, once every RTT when not in slow-start mode, TCP Vegas adjusts the window size as follows:

$$W = \begin{cases} W+1 & , \quad diff < \alpha \\ W & , \quad \alpha \le diff \le \beta \\ W-1 & , \quad diff > \beta \end{cases} \quad (2)$$

Alternatively, *diff* can be divided by $baseRTT$ [6], in which case the thresholds α and β are defined in a standard unit of throughput (e.g., packets/second); however, this results in unequal treatment of connections with different $baseRTT$ [16]. All Vegas implementations and simulations that we are aware of have the thresholds and *diff* in the unit of packets, which is assumed in the remainder of this paper. Both versions of the thresholds adjust the sending rate so as to utilize the available bandwidth without incurring congestion.

Another feature of Vegas is its modified slow-start behavior, which is more conservative than Reno's. Specifically, Vegas checks *diff* every *other* RTT and exits slow start if *diff* exceeds a threshold γ (or if a loss is experienced); otherwise, the window size is doubled. For simplicity, in the rest of the paper we will assume that $\gamma = (\alpha+\beta)/2$. This algorithm is another instance of Vegas' proactive congestion detection and loss avoidance mechanisms. Doubling the window every other RTT also facilitates obtaining a good measure of $baseRTT$.

The final four innovative mechanisms in TCP Vegas are congestion recovery mechanisms[1]. First, a window size of two packets (instead of one) is used at initialization and after a time-out. Second, Vegas records the time each packet is sent, and when a duplicate ACK is received, the sender retransmits the oldest unacknowledged packet if it was sent longer ago than a specified "fine grain timer" value. As in Reno, a triple-duplicate ACK always results in packet retransmission, but the fine-grain timers detect losses earlier, leading to packet retransmissions after just one or two duplicate ACKs. If the retransmission occurs, each of the next two normal ACKs will also trigger a retransmission of the oldest unacknowledged packet if its fine-grain timer has expired. Note that packet retransmission due to expired fine-grain timers is conditioned on receiving certain ACKs. Third, after packet retransmission triggered by a duplicate ACK, the congestion window size is reduced only if the time since the last window size reduction is more than the current RTT. After a retransmission triggered by a non-duplicate ACK, the window size is not reduced. Note that when multiple losses occur in a single window, Vegas decreases the congestion window size only for the first of those losses. Fourth, when the window is reduced due to a loss identified by a duplicate ACK, Vegas reduces the window size by 25%, instead of 50% as in Reno.

If a loss episode is severe enough that no ACKs are received to trigger the fine-grain timer checks, losses are identified by Reno-style coarse-grain time-outs. In the remainder of the paper, the term time-out (TO) refers to the coarse grain TOs unless otherwise stated.

2.2 Related Work

Several analytic models for the throughput of a single TCP Reno batch transfer as a function of measured loss rate and average RTT have been proposed in the literature. Mathis et al. [19] analyzed the congestion avoidance of TCP Reno ignoring time-outs. Padhye et al. [22] provided a more complete approach by including time-outs. Using their results and including initial slow-start in the analysis Cardwell et al. [8] derived a model for estimating the latency of an arbitrary size TCP Reno transfer. The model in [22] is revisited by Goyal et al. in [12] and a revised version is proposed. A different approach is taken by Misra et al. [20] where the steady state behavior of TCP Reno is modeled using fluid analysis. Our modeling approach is similar to that in [22], in that we analyze the flow in a per-round basis. On the other hand, our approach differs by analyzing the very different behavior of TCP Vegas, and also by using a simpler approach to capture the TCP window size evolution.

The main goal of prior measurement studies of TCP Vegas has been to compare Vegas with TCP Reno. Brakmo et al. [6], [7] performed Internet experiments and simulation, reporting 40-70% improvements in throughput, with 20-50% fewer retransmissions. They also conclude that Vegas is at least as fair as Reno. Ahn et al. [2] performed Internet measurements, and found 4-20% improvement in throughput, fewer retransmissions, and lower average and variance in the RTT. Hengartner et al. [14] isolate the different innovative mechanisms of Vegas. Using simulation, they find that in the presence of more than 2% loss, Vegas outperforms Reno; they conclude that the most effective mechanisms in TCP Vegas are the 25% decrease of the window size and the use of non-duplicate ACKs to identify losses. Mo et al. [21] use simulation with different buffer sizes, and find that Vegas obtains more bandwidth than Reno when the buffer size is very small.

To our knowledge, all of the analytic models developed for the throughput of TCP Vegas to date, assume a loss-free operation of the protocol. Using these models, several properties of the conges-

[1]The first and fourth of the congestion recovery mechanisms are not mentioned in [6, 7] but are identified as parts of TCP Vegas in [2] and [14].

Variable	Definition
α, β	Vegas throughput thresholds, measured in packets
γ	Threshold for exiting slow start, $\gamma = (\alpha + \beta)/2$
p	Inverse of the average number of packets transmitted between loss episodes
$baseRTT$	The minimum round-trip-time observed throughout the flow
RTT	An arbitrary round trip time
R	The average-round-trip time for the transfer
W	The window size at an arbitrary point in time
W_{max}	The maximum window size advertised by the receiver
W_0	The average window size during stable-backlog state
T_0	The average duration of the first TO in a TO series

Table 1: Model Notation

tion avoidance mechanism of Vegas have been investigated. Hasegawa et al. [13] find that Vegas can be unfair if $\alpha \neq \beta$, and that $\alpha = \beta$ improves fairness. Boutremans et al. [5] use a simple analytic model of one queue shared by a number of Vegas flows that arrive at different times, showing that Vegas is unfair due to the inaccurate measures of propagation delays and the difference between α and β. Bonald [4] develops a fluid approximation, and proves that (a) equilibrium is guaranteed to be reached if the available buffers are large enough for the desired backlog of all Vegas flows, (b) otherwise Vegas falls back to Reno, and (c) Vegas utilizes the network more efficiently than Reno and avoids the bias of Reno against flows with long propagation delays. Mo et al. [21] also used a fluid approximation and also conclude that Vegas throughput isn't dependent on propagation delay. Low et al. [18] model Vegas as a distributed optimization algorithm. They show that Vegas uses queueing delay as a congestion measure and verify all the above findings. Using a duality model in [16], Low finds that Vegas achieves proportional fairness and that when Vegas and Reno flows share a common network, their relative throughput mainly depends on the network configuration.

Although several models have been proposed for the all-Vegas no-loss environment, the impact of losses due to either buffer limitations or interaction with other Internet traffic has not been studied. As a result, there is no analytic characterization of the congestion recovery mechanisms of Vegas, which were shown in [14] to significantly contribute to increased performance. Our work bridges the gap between the experimental studies of TCP Vegas in environments where a wide range of loss rates is experienced, and the analytical models proposed that address Vegas congestion avoidance mechanism in an idealized loss-free environment.

3. THE MODEL

The model notation is summarized in Table 1. Model input parameters are R and p (as in previous TCP Reno models [8, 22, 12]), $baseRTT$, W_{max}, T_0, α, and β.[2]

[2]Since TCP Vegas uses the measured RTT to compute the throughput in each RTT, which in turn affects the number of packets sent in the next RTT, one might imagine that an accurate throughput model would require a description of the distribution of round-trip-times. On the other hand, the maximum RTT is bounded by the sum of the maximum delay at each node in the path traversed by the flow. Simulations of bulk transfers with bursty HTTP and other TCP cross traffic described later in the paper, show that the throughput calculated from the average RTT and packet loss rate is reasonably accurate. This indicates that, to a first approximation for current networks, the fluctuations in the RTT and packet loss rate do not need to be captured in the throughput model. Modeling of the fluctuations, which would more precisely characterize the conditions under which the mean values alone determine throughput, is an interesting topic for future work.

As in previous successful TCP Reno throughput models, we model the TCP Vegas behavior in terms of rounds, where a window of data is transmitted per round and the round duration is assumed to be equal to the RTT and independent of the window size. We assume that packet losses occurring in different rounds are independent, but when a packet is lost, all the remaining packets in the same round are also lost, constituting a loss episode.

One further assumption, namely that $baseRTT$ is relatively stable throughout the flow, is needed so that the throughput of a randomly selected interval is equal to the flow throughput. Experiments in Section 4 show that this assumption holds under most practical network conditions. If it does not hold, the throughput model could be applied to each portion of the flow that has a different value of $baseRTT$.

Below we first consider TCP Vegas throughput for flows that experience no packet loss, followed by flows that experience no timeouts, flows that experience only single timeout events, and finally flows that experience timeouts for consecutive packet transmissions. In each case, we model an expanded set of Vegas' mechanisms and compute a closed form expression for throughput.

3.1 Model 1: No Packet Loss

The evolution of the expected TCP Vegas window size when no packet loss occurs is illustrated in Figure 1. The flow begins in slow start with window size equal to two, and the window size is doubled every other RTT until *diff* exceeds γ (the common case), or until the window size reaches W_{max}. After that, the flow remains in congestion avoidance.

Consider an arbitrary point after the slow start period. Let $W_0^{no-loss}$ represent the expected size of the window at that point, W represent the actual window size and RTT be the most recently measured round trip time. Then the value of *diff* at this point in time is given by equation 1.

We assume that the average value of *diff* is approximately β, for two reasons. First, since $\alpha = \beta$ improves Vegas fairness and implies that $\gamma = \beta$, the doubling of the window size during the initial slow start period will tend to terminate when *diff* exceeds β. Furthermore, due to absence of significant congestion in the network, RTT does not fluctuate very much, and thus once *diff* decreases to β, it tends to stay relatively constant, as observed during extensive simulations of TCP Vegas with a wide variety of network cross traffic. Second, in the no loss case, RTT will tend to fluctuate near $baseRTT$, and thus the congestion avoidance algorithm will keep the number of packets queued as high as possible.

Taking the expectation on both sides of equation (1), assuming RTT has low variance and is independent of window size, solving for $W_0^{no-loss} = E[W]$, and accounting for the maximum window size, W_{max}, we get

$$W_0^{no-loss} = min\left(\beta \times \frac{R}{R - baseRTT}\ ,\ W_{max}\right). \qquad (3)$$

When computing the throughput of a bulk transfer, the through-

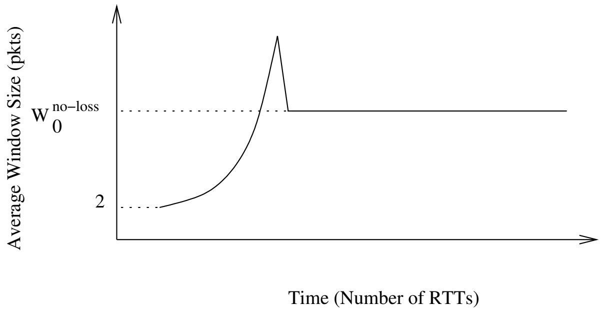

Figure 1: Evolution of Expected Window Size: No Loss

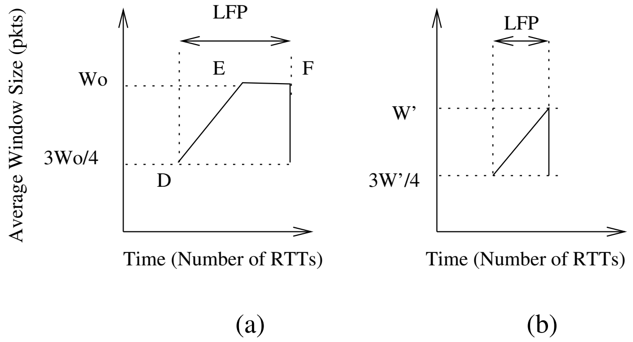

Figure 2: Evolution of Expected Window Size During a Loss Free Period

put during the initial slow start phase is negligible. Thus, on average, TCP Vegas will transmit $W_0^{no-loss}$ packets per round. When $W_0^{no-loss} < W_{max}$, this yields:

$$\Lambda_{no-loss} = \frac{\beta}{R - baseRTT} \qquad (4)$$

Whenever loss is negligible (e.g., in an all-Vegas environment [4]) the TCP Vegas throughput is estimated by the above formula. This formula shows that the measure that Vegas uses to reduce throughput, detect network congestion, or determine available bandwidth is $R - baseRTT$ (i.e., queueing delay) [16]. Estimated queueing delay is expected to be approximately the same for flows sharing the same bottleneck, assuming an accurate $baseRTT$ for each flow. Thus, as shown in equation (4), when loss is negligible TCP Vegas does not have a bias against flows with large propagation delays, as occurs in Reno. This agrees with the results in [21] and [4], and is further verified in section 4.4.1.

3.2 Model 2: No Time-Outs

When a TCP Vegas flow shares a bottleneck link with Reno-like TCP sources, or with uncontrolled cross traffic, losses will be experienced. In this section we assume that such losses occur, but all loss episodes are identified by duplicate ACKs (any number between one and three), where a loss episode is a series of packet losses during a single round. Given this assumption, when a loss episode occurs, Vegas will react to the first detected loss in the round by reducing the window size by 1/4. Further packets that are lost in the same round cause no further reduction in the window size (see section 2.1). Once the window size is reduced, Vegas continues congestion avoidance, regulating window size according to equation (2).

We call the intervals between loss episodes Loss Free Periods (LFPs). Ignoring the initial slow start period that has negligible impact on the throughput of the bulk transfer, the flow consists of a series of statistically identical LFPs. We consider two cases. First, for small enough values of p, the flow reaches the "stable backlog state" that characterizes the no-loss flow, as illustrated in Figure 2(a), in essentially every LFP. Second, for large p, the flow essentially never reaches this state, as illustrated in Figure 2(b). Note that as p decreases from case (b) to case (a), the expected maximum window size for the LFPs that do *not* reach stable backlog tends toward W_0. Thus, for simplicity in the analysis, we analyze a random LFP assuming that Figure 2(a) represents the expected window size evolution if the *average* number of packets that arrive between loss episodes is sufficient for the window size to reach W_0 from $3W_0/4$, i.e., "stable backlog is attainable, on average". Otherwise, we assume Figure 2(b) represents the expected window size evolution of the random LFP. Sections 3.2.1 and 3.2.3 compute the throughput of the random LFP for each of these cases, respectively. More precise analysis of flows that are mixtures of both types of LFPs could be pursued in future work, although the model validations later in this paper show that this approximate model is quite accurate.

Section 3.2.2 derives a simple formula to determine from the model inputs whether the LFP reaches the stable backlog state, on average. This formula is used in the model, and could also be used in equation-based rate control, to determine whether the throughput formula from section 3.2.1 or the formula from section 3.2.3 should be used to estimate the throughput from the measured model inputs.

3.2.1 Stable-Backlog is Attainable

The expected window size during the stable-backlog state (W_0) can be derived in a manner similar to the no loss case. However, since the level of congestion in the network is fluctuating, Vegas will adjust the backlog in the network between α and β, and thus the expected value of *diff* is estimated as $(\alpha + \beta)/2$ rather than β, which yields

$$W_0 = min\left(\frac{\alpha + \beta}{2} \times \frac{R}{R - baseRTT}, W_{max}\right). \qquad (5)$$

To derive the throughput of the LFP we need to calculate the average number of packets transmitted during an LFP, P_{LFP}, and the expected duration of an LFP, D_{LFP}. Using arguments analogous to those in [22], the throughput of the LFP is the ratio of these two expectations. P_{LFP} can be expressed as the expected number of packets transmitted between two loss episodes (i.e., $1/p$), plus the number of packets transmitted between the time the first lost packet is sent and the time the sender identifies the loss [22]:

$$P_{LFP} = \frac{1}{p} + W_0 - 1, \qquad (6)$$

To calculate D_{LFP}, we use the notation in figure 2(a) and let $D_{LFP} = D_{DE} + D_{EF}$. During stage D to E, Vegas (ideally) increases the window size by one in each round, for $W_0/4$ rounds, on average. Thus,

$$D_{DE} = \frac{W_0}{4} \times R \ . \qquad (7)$$

During stage (E to F) Vegas transmits an average of W_0 packets per round. Thus, assuming low variance in the window size during this stage, the expected number of rounds in this stage is equal to the expected number of packets transmitted during the interval (i.e., P_{EF}) over W_0. Since $P_{EF} = P_{LFP} - P_{DE}$, we have

$$D_{EF} = \frac{P_{LFP} - P_{DE}}{W_0} \times R \ , \tag{8}$$

where

$$P_{DE} = \sum_{i=\frac{3W_0}{4}}^{W_0-1} i = \frac{7W_0^2}{32} - \frac{W_0}{8} \ . \tag{9}$$

Using equations (6) - (9) and simplifying, yields

$$D_{LFP} = \left(\frac{1-p}{pW_0} + \frac{W_0}{32} + \frac{9}{8}\right) R. \tag{10}$$

Finally, dividing equations (6) and (10) we find

$$\Lambda_{no\_TO}^{stable} = \frac{\frac{1-p}{p} + W_0}{\left(\frac{1-p}{pW_0} + \frac{W_0}{32} + \frac{9}{8}\right) R} \ , \tag{11}$$

where W_0 is given in equation (5). We note that this analysis has yielded a fairly simple formula for TCP Vegas throughput. In this case, substituting equation (5) in the throughput equation shows that when packet loss occurs (e.g., due to other types of flows in the network) there is some bias against connections with longer average RTT. The bias doesn't have a simple characterization, but we explore this new insight further in the experiments in section 4.4.1.

3.2.2 Condition for Attainable Stable-Backlog

Equation (11) holds only if the loss episode happens (on average) after W reaches W_0; that is, $P_{DE} \leq (1/p + W_0 - 1)$, or using equation (9),

$$p \leq \frac{32}{7W_0^2 - 36W_0 + 32} \tag{12}$$

If equation (12) together with (5) does not hold, we use the analysis presented next.

3.2.3 Stable-Backlog is Not Attainable

When a loss episode occurs on average before stable backlog is reached, the behavior of Vegas (depicted in Figure 2(b)) is similar to that of Reno since the congestion avoidance mechanism reverts to that of Reno; however, there are significant differences in the congestion recovery mechanisms.

To compute W', we note that during the LFP, Vegas (ideally) increases the window size by one in each round, and that the expected number of packets transmitted (P'_{LFP}) is $1/p + W' - 1$. Thus,

$$\sum_{i=\frac{3W'}{4}}^{W'} i = \frac{1}{p} + W' - 1 = P'_{LFP} \Rightarrow W' = \frac{2 + 2\sqrt{\frac{56}{p} - 55}}{7}. \tag{13}$$

Since the expected number of rounds in the LFP is $W'/4$,

$$\Lambda_{no\_TO}^{not-stable} = \frac{P'_{LFP}}{D'_{LFP}} = \frac{\frac{1-p}{p} + W'}{\frac{W'}{4} R} = \frac{4\sqrt{\frac{56}{p} - 55} + \frac{14}{p} - 10}{(1 + \sqrt{\frac{56}{p} - 55})R} \tag{14}$$

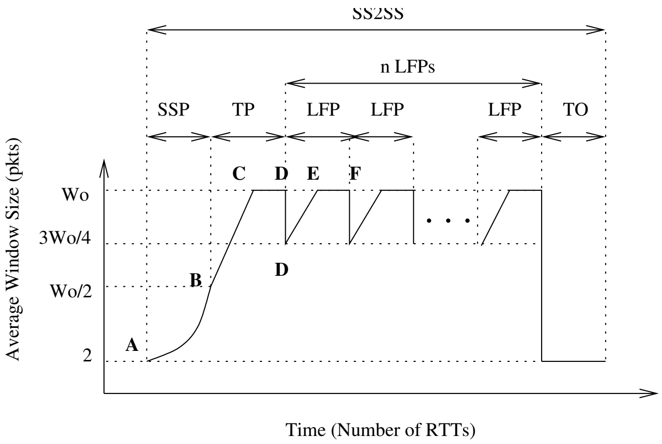

Figure 3: Example SS2SS Period

The above equation shows that if stable backlog is never reached, Vegas throughput is inversely proportional to the average propagation delay (R), as is the case for TCP Reno. In general, the dependence of TCP Vegas throughput on R is not as straightforward as in Reno. There are two extremes, namely (1) the case where $p = 0$ and Vegas throughput does not depend on R, and (2) the case where p is large enough that stable backlog is never reached and Vegas throughput is inversely proportional to R. As p increases between these two extremes, the dependence of throughput on R becomes stronger until it reaches the inverse proportional dependence.

3.3 Model 3: Single Time-Outs Only

The next aspect of TCP Vegas flows to be represented in the model is that of Time-Outs (TOs). In this case, loss episodes are identified by duplicate ACKs or by TOs. A TO occurs if after a loss episode, not enough duplicate ACKs return to the sender to trigger lost packet retransmissions.

When the coarse-grain timer expires for a packet, Vegas remains idle for a period of T_0, and then sets the window to two and goes into Slow-Start. T_0, is calculated every RTT as two times the smoothed RTT average plus four times the RTT variance.

Here, we will assume that all TO series consist of a single TO. We first analyze the case where all loss episodes occur when Vegas is in the stable-backlog state. Under this scenario, the behavior of the flow can be partitioned into a sequence of adjacent statistically identical intervals that have expected window size evolution as illustrated in Figure 3. We call each such interval a Slow-Start-to-Slow-Start (SS2SS) period.

To derive the throughput in a random SS2SS period, we compute the expected number of packets transmitted and the expected duration of such a period. To do this, we partition the SS2SS into the following periods (see figure 3): (1) the Slow Start Period (SSP) in which the window size starts at two and doubles every other round until it reaches the slow start threshold ($ssthres\_$)[3], (2) the Transition Period (TP) during which the window size increases by one each round until stable-backlog state is reached, and then the flow stays in stable-backlog state until a loss episode occurs, (3) if the TP does not end with a TO, a series of n consecutive Loss Free Periods (LFPs) follows, with the first $n-1$ LFPs ending with a loss episode identified by a duplicate ACK, and the n-th LFP ending with a loss episode identified by a TO, (4) a single time-out. Thus,

[3] $ssthresh\_$ is on average equal to $W_0/2$ since the expected window size when the TO occurs is W_0.

$$\Lambda_{SS2SS} = \frac{P_{SSP} + P_{TP} + nP_{LFP} + P_{TO}}{D_{SSP} + D_{TP} + nD_{LFP} + D_{TO}}. \quad (15)$$

where P_X denotes the average number of packets transmitted in period X and D_X denotes the average duration of the period. These terms for each component of the SS2SS period are derived in each of the next three sections, respectively.

3.3.1 Slow Start (SSP) and Transition Phase (TP)

The SSP and TP are illustrated in Figure 3 between points A and B and points B and D, respectively. The period from A to D starts and ends with consecutive loss episodes; thus,

$$P_{AD} = P_{SSP} + P_{TP} = \frac{1}{p} + W_0 - 1 \ . \quad (16)$$

Since slow start begins with a window size of two, and the window size is doubled every other RTT until the slow start threshold ($ssthresh\_ = W_0/2$) is reached,

$$P_{SSP} = 2(2 + 4 + \ldots + \frac{W_0}{4}) = 2\sum_{i=0}^{log\frac{W_0}{4}} 2^i = 2^{logW_0} - 4$$

$$\text{and} \qquad D_{SSP} = 2(log\frac{W_0}{4})R = 2(logW_0 - 2)R. \quad (17)$$

The analysis of a random TP is similar to the analysis of an LFP (section 3.2), except that the expected initial window size is $W_0/2$. Thus, for stage B to C in figure 3,

$$P_{BC} = \sum_{i=\frac{W_0}{2}}^{W_0-1} i = \frac{3W_0^2}{8} - \frac{W_0}{4} \quad \text{and} \quad D_{BC} = \frac{W_0}{2}R \ . \quad (18)$$

The expected duration of stage (C to D) is $(P_{CD}/W_0)R$. Since $P_{CD} = P_{AD} - P_{SSP} - P_{BC}$, using (16)-(18), we get

$$\begin{aligned} D_{TP} &= D_{BC} + D_{CD} = \frac{W_0}{2}R + \frac{P_{AD} - P_{SSP} - P_{BC}}{W_0}R = \\ &= \left(\frac{1-p}{pW_0} + \frac{W_0}{8} + \frac{5}{4} + \frac{4 - 2^{logW_0}}{W_0}\right)R \ . \end{aligned} \quad (19)$$

3.3.2 Series of Loss Free Periods (LFPs)

Equations (6) and (10) give the expected number of packets transmitted in an LFP and its expected duration, respectively.

To derive an expression for n, the expected number of consecutive LFPs during an SS2SS period, we note from Figure 3 that the fraction of loss episodes that are identified by a TO, p_{TO}, is given by $p_{TO} = 1/(n+1)$. Solving this for n,

$$n = \frac{1 - p_{TO}}{p_{TO}}. \quad (20)$$

The probability p_{TO} has been derived for TCP Reno [22] by analyzing the probability that less than three duplicate-ACKs return to the sender after a loss episode. Vegas has the key difference that packets can be retransmitted with fewer than three duplicate ACKs, which may significantly reduce the probability of getting a time-out. In [14], it is claimed that this does not contribute greatly to the gains in performance of TCP Vegas compared to Reno. However, our simulation results showed that out of the losses identified by duplicate ACKs, the great majority are identified after one or two duplicate ACKs, and for loss rates higher than 5% only one duplicate ACK was needed to identify the loss in most cases. This motivated the following fairly simple analysis of p_{TO}, which assumes that every loss identified by duplicate ACKs is identified by the first duplicate ACK received.

Let $A(w,k)$ denote the probability that k out of w packets are acknowledged, given that there is a loss episode in a round with window size w. Let $C(w,k)$ denote the probability that exactly k packets are received from a round of w. That is,

$$A(w,k) = \frac{(1-p)^k p}{1-(1-p)^w} \qquad C(w,k) = \begin{cases} (1-p)^k p & , \quad k < w \\ (1-p)^w & , \quad k = w \end{cases}$$

Given a round of w packets that has a loss episode, the scenarios that lead to no duplicate ACKs, and thus a TO in the round, are: 1) the entire window is lost, or 2) i out of w packets reach the receiver, the receiver sends i ACKs, the sender sends i new packets and all i packets are lost. Recalling that W_0 is the expected window size when a loss episode occurs,

$$\begin{aligned} p_{TO}(W_0) &= min\left(1 \ , \ A(W_0,0) + \sum_{i=1}^{W_0-1} A(W_0,i)C(i,0)\right) = \\ &= min\left(1 \ , \ \frac{p + p(1-p)(1-(1-p)^{W_0-1})}{1-(1-p)^{W_0}}\right) \end{aligned} \quad (21)$$

3.3.3 Time-Out (TO)

During the TO no packets are transmitted; thus,

$$P_{TO} = 0 \quad , \quad D_{TO} = T_0. \quad (22)$$

Substituting equations (6), (10), (16), (17), (19) and (22) into (15) we get the following estimated throughput of a TCP Vegas flow for the case that only single TOs occur:

$$\Lambda_{SS2SS} = \frac{(n+1)\left(\frac{1-p}{p} + W_0\right)}{N_{SS2TO}R + T_0} \quad (23)$$

where,

$$N_{SS2TO} = \ 2logW_0 + (n+1)\tfrac{1-p}{pW_0} + \left(1 + \tfrac{n}{4}\right)\tfrac{W_0}{8} + \quad (24)$$

$$+ \tfrac{9n}{8} - \tfrac{11}{4} + \tfrac{4-2^{logW_0}}{W_0} \quad (25)$$

This expression is significantly more complex than the previous formula for Vegas throughput that assumes no time-outs, but the dependence on R is still as explained in section 3.2.3. The value of n can be computed with equations (20), (21) and (5).

3.3.4 Stable-Backlog is Not Attainable

The constraint required in the above analysis, together with (12), is that the flow will be able, on average, to fully recover from the TO, reaching stable-backlog state *before* the next loss episode occurs. That is, $P_{AC} \leq (1/p + W_0 - 1)$, or

$$p \leq \frac{8}{3W_0^2 - 10W_0 + 16logW_0 + 8}. \quad (26)$$

In the case where either constraint (12) or (26) does not hold, Vegas, on average, will not reach the stable-backlog state between certain loss episodes. The window evolution in those cases is similar to figure 3, with the TP and LFPs having sharp peaks. With a few further simplifying assumptions, the following expression can be used for the expected TCP Vegas throughput under those conditions

$$\Lambda_{SS2SS}^{not-stable} = \frac{P'_{SSP} + P'_{TP} + nP'_{LFP} + P_{TO}}{D'_{SSP} + D'_{TP} + nD'_{LFP} + D_{TO}} \tag{27}$$

where P'_{LFP} and D'_{LFP} where derived in section 3.2.3. The analysis for SSP and TP is identical to that in section 3.3.1, with the transition between the two phases taking place when the window size is $W'/2$ instead of $W_0/2$. The main assumption for this analysis to hold is that the expected window size when a loss occurs at the end of the TP is the same as in section 3.2.3 for the LFPs, i.e., W'.

3.4 Model 4: Full Model

When a TO occurs, further TOs can occur back to back with the first. Here we derive the expected number of packets transmitted during a series of TOs and the expected duration of such a series in order to provide more accurate estimates for the terms P_{TO} and D_{TO}, in equation (15).

We identify three cases with respect to the round of two packets following a TO: (A): Neither packet is lost, with probability $\mathcal{P}_0 = (1-p)^2$; (B): Only the second packet is lost, with probability $\mathcal{P}_1 = (1-p)p$; (C): Both packets are lost , with probability $\mathcal{P}_2 = p$ (given the first loss the second loss happens with probability 1). Note that the probabilities sum to one. Case A signals the end of the TO series, whereas C result in a new TO. Case B results in a new TO only if the one extra packet transmitted as a response to the single ACK sent back by the receiver is also lost. Here, in order to simplify our analysis, we assume that this always happens, i.e. case B always results in a further TO. Thus, immediately following a TO round, the probability that a further TO will occur is $\mathcal{P}_1 + \mathcal{P}_2$. Let M be the number of consecutive TOs in a random TO series. Then, since the first TO is given,

$$\mathcal{P}[M = k] = (\mathcal{P}_1 + \mathcal{P}_2)^{k-1} \times \mathcal{P}_0 = (2p - p^2)^{k-1}(1-p)^2, \tag{28}$$

and the expected number of consecutive TOs, $E[M]$, is

$$E[M] = \sum_{k=1}^{\infty} k\mathcal{P}[M = k] = \frac{1}{(1-p)^2}. \tag{29}$$

Since the window size is 2 after each TO, $P_{TO-series} = 2(E[M]-1)$. We exclude the 2 packets transmitted in the round right after the last TO, since that round is included in the next SSP (see equation (17)). Finally, the expected number of packets transmitted during a random TO series, is

$$P_{TO-series} = \frac{2p(2-p)}{(1-p)^2}. \tag{30}$$

The duration of the first TO is T_0, and for each new TO the duration is doubled until the duration reaches $64T_0$. For further TOs the duration remains constant. The duration of a series of k TOs is thus:

$$D_k = \begin{cases} (2^k - 1)T_0 & , \quad k < 6 \\ (63 + 64(k-6))T_0 & , \quad k \geq 6 \end{cases}$$

The expected duration of a random TO series, $D_{TO-series}$ is given by

$$D_{TO-series} = \sum_{k=1}^{\infty} D_k \mathcal{P}[M = k] =$$
$$= \left(\frac{64}{(1-p)^2} - 321 + (1-p)^2 \sum_{k=1}^{6} (2^k - 64k + 320)(2p - p^2))^{k-1} \right) T_0 \tag{31}$$

We will refer to the quantity in parentheses in the above equation as $d(p)$; i.e., $D_{TO-series} = d(p)T_0$.

In equation (15) we replace P_{TO} and D_{TO} which refer to a single TO with the more accurate estimates $P_{TO-series}$ and $D_{TO-series}$. The analysis for the rest of the SS2SS period is exactly as it was described in the previous section. Thus, in equation (15), we get

$$\Lambda_{loss} = \frac{(n+1)\left(\frac{1-p}{p} + W_0\right) + \frac{2p(2-p)}{(1-p)^2}}{N_{SS2TO}R + d(p)T_0} \tag{32}$$

Variable n can be computed from equations (20)-(21), N_{SS2TO} is given in equation (25) and W_0 is given in (5).

The set of equations (4) and (32) constitute a complete throughput model for TCP Vegas, including both the loss and the no-loss scenario, given that constraints (12) and (26) are met.

4. RESULTS

We were unable to find an implementation of TCP Vegas that would compile on the systems available to us at various sites. Thus, instead we conducted many experiments using the ns-2 simulator [1] to validate the throughput model developed in section 3, to examine some of the new qualitative insights obtained from the model, and to reevaluate the results in the literature concerning the relative performance of TCP Vegas and TCP Reno and New-Reno.

Most previous simulation comparisons of TCP Vegas and TCP Reno performance have assumed a small buffer at the shared bottleneck link; for example, the buffer size is 4-16KB in [2], and in [6, 7, 14] it is 10-20KB. This assumption affects the relative performance of the protocols, and in particular favors Vegas, as has been shown in a simple environment with two flows [21] and [16]. In the experiments below, we assume larger finite buffers at the shared bottleneck (e.g., 50 - 200 packets for a T3 bottleneck link), unequal propagation delays in the flows that share the link, and a rich mixture of background traffic.

We perform experiments for the simple dumb-bell network topology, in which each source transmits through its own non-shared incoming path to a shared bottleneck link and then to its own non-shared outgoing path. The flows sharing the bottleneck are a specified mix of Vegas, New-Reno, and Reno bulk TCP transfers, and bursty ON/OFF HTTP flows. Multihop link configurations are also of interest; however, there are many parameters to vary in this dumb-bell topology with rich traffic mixtures and finite buffer on the shared link. We obtain insights and understanding from this representation of the principal bottleneck for the flow of interest; deferring study of more complex systems to future work.

For the cross traffic in our experiments we use round-trip propagation delays ranging from 20 to 460 milliseconds [3, 15, 11]. To compare the different TCP variants, we distinguish three monitored flows, one Vegas, one New-Reno and one Reno, which are configured with the same path delay. We analyze the performance of these "foreground" flows in a variety of network settings. All other flows are called "background" flows.

Packet size	1KB
Buffer size	200 pkts
Bottleneck link capacity	T3 (44.736 Mbps)
Incoming/Outgoing link capacity	100 Mbps
Bottleneck link propagation delay	10ms
Path delay for background flows	uniform(20,460) ms
Queue management scheme	Drop Tail
α, β	6 pkts
TCP maximum window size	64 pkts

Table 2: Default System Configuration

Each ON/OFF HTTP background flow is configured similarly to the one in [17]. That is, each HTTP client sends a single packet request across the reverse bottleneck link (shared with the acknowledgment traffic for the TCP flows) to a non-shared server. The HTTP server, upon receiving the request, uses TCP-New Reno to send a file to the client of size exponentially distributed with mean 50KB. After the client receives the entire file, the client waits for a time that is exponentially distributed with mean 500 milliseconds and then sends another request to the server.

While verifying that all the Vegas mechanisms described in section 2.1 are present in the ns implementation, *and* through comparisons with the analytic model predictions, we discovered and fixed several bugs in the Vegas implementation of ns. The most important was a mistaken calculation that computed artificially large RTT values for retransmitted packets.

We derive the model inputs from the measured foreground Vegas flow. Specifically, we use the measured packet loss rate as an approximation of the parameter p in the model, the average RTT of the flow over the duration of the connection as R, and the minimum RTT measured during the flow as $baseRTT$. We also obtain the average duration of the first TO in a TO series, T_0, from the simulation, although we experimented with $T_0 = 2R$ and found that this value gave similar model accuracy. The inputs α, β and W_{max} are configuration settings in the simulations.

We use equal values for the two thresholds α and β in most experiments since it has been shown that such a configuration leads to fair allocation of bandwidth between competing Vegas flows [13]. To extensively validate the throughput model, we varied the bottleneck link speed, buffer size, incoming/outgoing link capacities, path delay distributions, the TCP Vegas configuration parameters α and β, and the number and mix of background sources. These experiments also yield new insights about TCP Vegas performance. Below are representative results that unless otherwise noted were based on the parameters shown in Table 2. Note that the given bottleneck buffer size, 200 packets (four times the delay-bandwidth product for the link) does not favor TCP Vegas performance.

4.1 Varying cross traffic

In our first set of experiments, we varied the total number of background flows (e.g., from 30 to 300 over a T3 bottleneck link), while keeping the relative mix of background flows fixed (e.g., half New Reno and half HTTP). The goals are to compare the protocols under a range of cross traffic loads and network configurations, and to validate the model against a wide range of loss rates and cross traffic behavior.

Figure 4 shows the throughput of each of the three foreground flows as a function of the total number of background flows, for the case that the background sources are an equal mix of TCP New Reno, TCP Vegas, and HTTP, and the path delay of the foreground flows is set to 80 milliseconds. For given path delay, α and β, the relative throughput of the protocols depends on the buffer size as well as the fractions of flows that use Vegas and Reno, a result that agrees with the theoretical analysis in [16]. For a small number of background flows (relative to the link speed and buffer size) there is very little congestion in the network. The flows spend most of their time in congestion avoidance where, in the given example, the New-Reno flows are more aggressive, using a larger fraction of the available buffer space than the Vegas flows. The result is that the New-Reno flow has higher throughput and also suffers more losses. As the number of background sources increases, the Vegas and New Reno flows have similar performance, both in throughput and loss rate. (That is, the Vegas sources cannot attain the specified number of buffers and their behavior becomes Reno-like.) Finally, when congestion becomes heavier Vegas outperforms New-Reno since it employs more aggressive congestion recovery mechanisms. The Reno flow has consistently worse performance than the other two, mainly due to the increased number of TOs that it experiences.

Note that the relative aggressiveness of each New Reno flow during congestion avoidance depends on the path propagation delay, whereas the relative aggressiveness of Vegas depends on the fraction of flows that use TCP Vegas as well as on the values of the parameters α and β. Higher fraction of Vegas flows or higher values of α and β result in Vegas being more aggressive. However, for larger α an β there may not be enough space for a large number of Vegas sources to attain their stable backlog. A similar argument applies for the relative performance of Vegas and Reno with respect to buffer size. That is, for a smaller buffer size, Vegas will be more aggressive relative to Reno, but for fixed α and β, fewer Vegas sources can share the buffer and attain their target backlog.

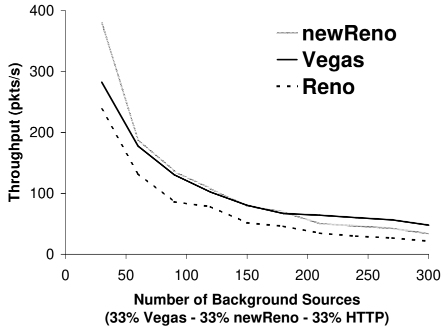

Figure 4: Throughput for Varying Cross Traffic (monitored flow path delay = 80 ms)

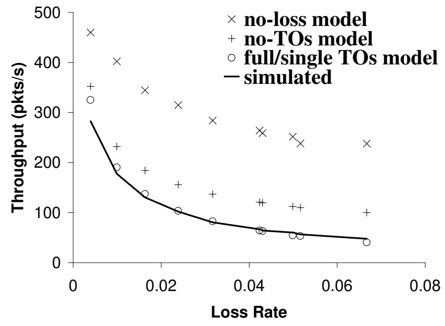

Figure 5: Model Accuracy for Varying Cross Traffic

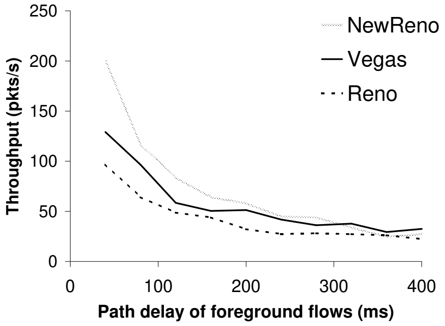

Figure 6: Throughput for Varying Propagation Delay

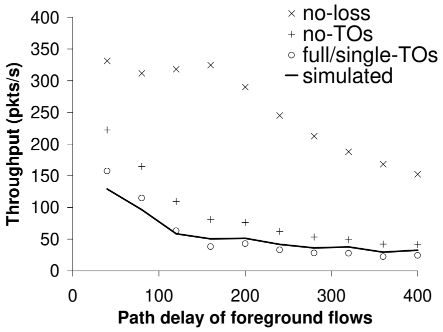

Figure 7: Model Accuracy for Varying Propagation Delay

A possible topic for future research is to investigate variants of Vegas that set these thresholds dynamically, possibly making use of equation (12) to determine when to adjust the thresholds.

Figure 5 provides a representative comparison of the analytic model throughput estimates compared with the simulation results. For each experiment shown in Figure 4, the simulation and analytic throughputs are plotted against the packet loss rate of the simulated the Vegas flow. Both the full model and the 'Single-TOs' model agree well with the throughputs measured in the simulations, having an average error of 6.8% and a worst case error near 15%. This model accuracy is typical among all experiments we have conducted. Notably, the 'single-TOs' model estimate is indistinguishable from the full model, for all the experiments. This suggests the rare occurrence of back to back TO events in Vegas, even for fairly large loss rates. Consequently, it appears that equation (23) can be used instead of the more complex equation (32) without any appreciable loss of accuracy. On the other hand the 'no-TOs' model that ignores TOs is significantly less accurate in the majority of cases.

4.2 Varying propagation delay

The previous section validated the model for varying cross traffic, which in turn varied the loss rate p. In this section, we explicitly vary a different key model input, namely $baseRTT$.

In a representative experiment, we set the number of background sources to 120 (40 Vegas, 40 New-Reno and 40 HTTP), and vary the path delay of the foreground flows from 40 ms to 400 ms, keeping the rest of the network configuration parameters the same as in the previous experiment.

The throughputs of the three foreground flows can be seen in figure 6. As before, the relative throughputs of the different protocols is due to the factors that affect their respective relative levels of aggressiveness. The model accuracy is shown in figure 7 and is similar to the previous experiment.

4.3 Varying Bottleneck Link Speed and Buffer Size

One further step in validating the model is to vary two of the most important network configuration parameters, namely the buffer size and the bottleneck link bandwidth. Again, we present representative results from among the many different experiments we conducted by varying these parameters.

The initial configuration has buffer size 50, bottleneck link speed of 5 Mb/s and 30 background sources (10 Vegas, 10 New-Reno and 10 HTTP ON/OFF clients). We vary these parameters simultaneously, in each new experiment increasing buffer size by 50, link speed by 5 Mb/s and number of background sources by 15 (five of each type). We stop at a link speed of 100 Mb/s. The path delay of

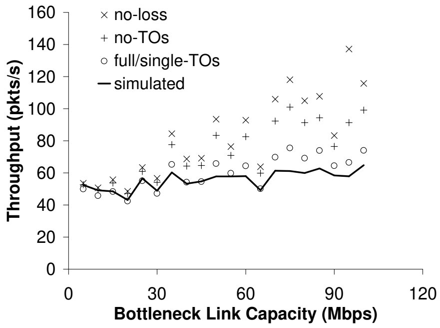

Figure 8: Model Accuracy for Varying Bottleneck Capacity

the foreground flows is set to 40 ms. The achieved Vegas throughput together with the model estimates are shown in figure 8. The full and the 'single-TOs' models perform equally well for the entire range of network configurations, with maximum error around 20%.

4.4 Fairness of TCP Vegas

The two main findings from previous investigations of TCP Vegas fairness, reviewed in 2.2, are: (1) Vegas removes Reno's bias against flows with large propagation delays [4, 18, 21] , and (2) Vegas creates persistent congestion which causes unfair distribution of the throughput due to inaccurate measures of $baseRTT$ [5, 18]. However, these issues have primarily been analyzed in environments where the Vegas sources experience no loss. Our analytic models suggest that Vegas has some bias against flows with larger propagation delay when packet loss is not negligible. In section 4.4.1 we obtain insight into the magnitude of this bias. In section 4.4.2 we provide results from our simulation experiments regarding the accuracy of the $baseRTT$ measures for network configurations in which packet losses occur.

4.4.1 Propagation Delay Bias

To compare bias against the connections with larger propagation delays, we compare two separate network environments: one has an equal number of TCP-Vegas sources and HTTP ON/OFF sources; the other has an equal number of TCP-New Reno sources and HTTP ON/OFF sources. In each environment, the bottleneck is a T1 (1.544 Mbps) link, the propagation delays of the TCP sources are uniformly distributed between 20 and 460 ms, and the number of TCP sources is varied from 6 to 60.

For each number of TCP sources, Figure 9 plots the average of the packet loss rates observed by the TCP flows in each envi-

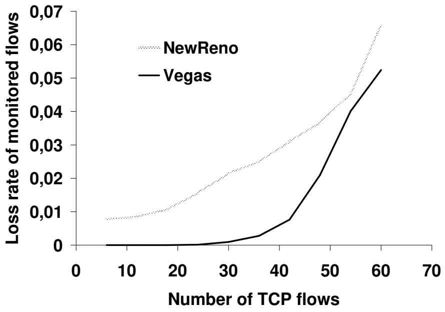

Figure 9: Average Packet Loss Rate (50% Vegas & 50% HTTP; or 50% New-Reno & 50% HTTP)

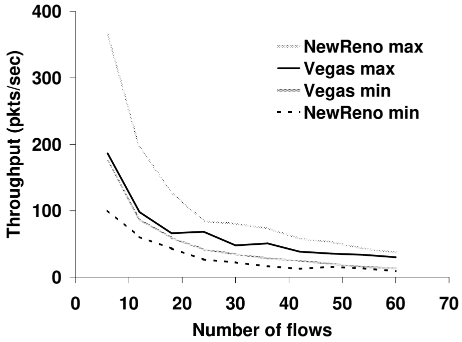

Figure 10: Minimum and Maximum Throughput of Vegas and New-Reno Flows

ronment,[4] whereas Figure 10 shows the maximum and minimum throughput obtained by the TCP flows in each environment. For up to 18 sources sharing the T1 link there is no loss, and all Vegas flows achieve approximately the same throughput, whereas there is considerable variation in the TCP New Reno throughputs. As packet loss increases, the difference between minimum and maximum Vegas throughput increases. For 60 background sources (5-6% packet loss), the ratio of maximum to minimum throughput is 2.3 for the Vegas flows and over 3 for the New Reno flows. We conclude that in the presence of low packet loss (i.e., a few percent), Vegas greatly reduces the bias against flows with larger propagation delays, but does not eliminate the bias.

4.4.2 Bias due to inaccurate baseRTT

If a new TCP Vegas connection experiences persistent queueing delay, it may constantly overestimate $baseRTT$. This situation results in a higher bandwidth share for a flow that begins when its bottleneck is busy [5, 14].

Results from our model validation experiments provide some new insights into this issue. Here we present results when a number of long-lived Vegas sources, ranging from 1 to 10, share a common T2 (6.132 Mbps) bottleneck link. The propagation delay of each connection is 40 ms, to avoid propagation delay bias, thus

[4]Note that while both TCP protocols experience similar loss rate when there is a mix of Vegas and New Reno bulk transfer flows, they experience quite different loss rates in an environment that contains only TCP flows of the one type, together with ON/OFF flows. The 50% Vegas/50% HTTP environment shows the advantages of the Vegas congestion avoidance mechanism.

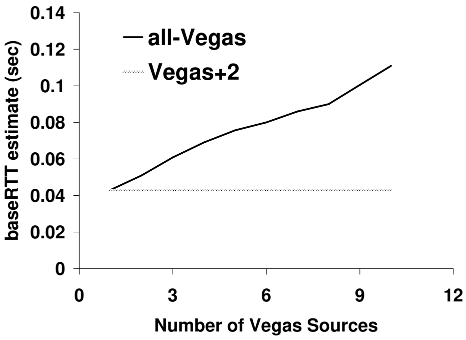

Figure 11: $baseRTT$ Estimate of the Last Flow to Start (All-Vegas; Vegas + 1 NewReno & 1 HTTP ON/OFF)

isolating unfairness due to inaccurate estimates of $baseRTT$. A low value of propagation delay is used, so that overestimation of $baseRTT$, if any, is significant compared to the actual propagation delay, creating a greater fairness challenge for the TCP Vegas protocol. Furthermore, with a buffer size of 200 packets, fairly large queue lengths can potentially be observed. The connections are started one after the other with one second intervals between their start times. These settings, particularly with ten Vegas sources, highly favor the creation of persistent congestion.

The curve labelled 'all-Vegas' in Figure 11 shows the baseRTT estimate obtained by the last flow to join the network, for each different total number of Vegas flows. This curve shows that $baseRTT$ is increasingly overestimated as a flow finds more flows already operating in the network. Note that there are no packet losses in this all-Vegas environment.

In practice, the problem only arises in cases where the buffer never empties. Under realistic scenarios with a variety of propagation delays and at least a small amount of bursty or TCP Reno cross traffic, the buffer occasionally empties and Vegas flows are able to get an accurate estimate of their propagation delays. To illustrate this point we repeat the experiment above with two changes. First, instead of fixing all propagation delays at 40 ms, the propagation delay of the last flow to join is 40 ms and the other flows have propagation delays uniformly distributed between 20 and 460 ms. Second and more importantly, for each number of Vegas sources from 1 to 10, we add two additional sources: one New-Reno bulk transfer and one HTTP source. Curve 'Vegas+2' in Figure 11 shows the $baseRTT$ estimates of the last flow to join the network for each different total number of Vegas flows. The estimates, unlike the original experiment (curve 'all-Vegas'), are perfectly accurate in this case.

5. CONCLUSIONS

This paper has developed a simple and accurate model to estimate the throughput of a Vegas flow as a function of packet loss rate, average round trip time, minimum observed round trip time, and protocol parameters α, β. The model provides two closed-form analytic throughput estimates, respectively for the cases that the network conditions do and do not permit to the TCP Vegas flow to acquire its target backlog in the connection path. A simple constraint on packet loss rate was developed to determine which of the two expressions should be used.

The model elucidates the dependence of TCP Vegas throughput on the model inputs, and provides insights for the protocol performance and characteristics. For example, when the flow is able to establish the target backlog, the throughput expression shows the absence of bias against flows with larger propagation delays in a loss free environment (as shown in previous work), and a weaker bias than that of Reno if losses occur. The expression for the non-stable backlog state is similar to expressions developed for the throughput of TCP Reno, which shows that throughput is inversely proportional to average round trip time, since under such conditions Vegas congestion avoidance falls back to that of Reno.

The model was found to be quite accurate over many different network settings that exercised both cases of the model and provided extensive variation in the three key model inputs, namely loss rate, average round-trip time and path propagation delay. The worst case error in the model throughput estimates was near 25%, but typical error is within 10-15%. The validations revealed that the 'single-TO' model is as accurate as the full model, indicating that timeouts for consecutive packets are rare in TCP Vegas flows.

The experiments that validate the model, also provided further insights into the performance of key innovative mechanisms in TCP Vegas and the relative performance of TCP Vegas, New-Reno and Reno. These results agree with previous observations that Vegas flows sharing the network with flows employing Reno-like congestion control can get a smaller or larger share of the available bandwidth, depending on the buffer size at the bottleneck router, the mix of flows sharing the buffer, the flow path delays, and the values of the thresholds α and β. Furthermore, the experiments show that setting the α and β thresholds statically is sub-optimal for the TCP Vegas flow. The simulation experiments also showed that one of the main concerns about the deployment of TCP Vegas, namely the potentially inaccurate estimates of propagation delay that can lead to unfair bandwidth allocation, does not occur in practical configurations that include even a very small number of bursty or Reno-like flows as well as variable propagation delays for the TCP flows that share the bottleneck link.

Fruitful avenues for future research include investigating: (1) the use of the expression that determines whether the Vegas flow can attain its target backlog to dynamically set α and β, and (2) use of the TCP Vegas throughput model for rate control of UDP streams in the Internet.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under grant ANI 0117810. The authors thank Derek Eager for his insights into modeling TCP Vegas throughput, Paul Barford for his valuable comments concerning TCP Vegas and the ns2 simulator, and Armageddon Brown for his help in conducting an initial set of ns2 simulation experiments.

6. REFERENCES

[1] Ns-2 simulator, http://www.isi.edu/nsnam/ns.

[2] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: Emulation and experiment. In *SIGCOMM 95*, Cambridge, MA, August 1995.

[3] M. Allman. A web server's view of the transport layer. *ACM Computer Communication Review*, 30(5), Oct. 2000.

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

[5] C. Boutremans and J. Y. L. Boudec. A note on the fairness of TCP Vegas. In *International Zurich Seminar on Broadband Communications*, Zurich, Switzerland, February 2000.

[6] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *SIGCOMM 94*, London, UK, Sept. 1994.

[7] L. S. Brakmo and L. L. Peterson. TCP Vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8), 1995.

[8] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *INFOCOM 00*, Tel Aviv, March 2000.

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 00*, Stockholm, Sweden, August 2000.

[10] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. *RFC 2582*, 1999.

[11] S. Floyd and E. Kohler. Interent reasearch needs better models. In *Hotnets I*, Princeton, NJ, October 2002.

[12] M. Goyal, R. Guerin, and R. Rajan. Predicting TCP throughput from non-invasive network sampling. In *INFOCOM 02*, New York, NY, June 2002.

[13] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and stability of congestion control mechanisms. In *Globecom*, Rio de Janeiro, Brazil, December 1999.

[14] U. Hengartner, J. Bolliger, and T. Gross. TCP Vegas revisited. In *INFOCOM 00*, Tel Aviv, March 2000.

[15] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM Computer Communications Review*, July 2002.

[16] S. Low. A duality model of TCP and queue management algorithms. In *ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management 00*, Monterey, CA, September 2000.

[17] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle. Dynamics of TCP/RED and a scalable control. In *INFOCOM 02*, New York, NY, June 2002.

[18] S. H. Low, L. L. Peterson, and L. Wang. Understanding TCP Vegas: A duality model. In *SIGMETRICS/Performance 01*, pages 226–235, Cambridge, MA, June 2001.

[19] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), 1997.

[20] V. Misra, W. Gong, and D. Towsley. Stochastic differential equation modeling and analysis of TCP-windowsize behavior. In *Performance*, Istanbul, Turkey, October 1999.

[21] J. Mo, R. J. La, V. Anantharam, and J. C. Walrand. Analysis and comparison of TCP Reno and Vegas. In *INFOCOM 99*, New York, NY, March 1999.

[22] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM 98*, Vancouver, CA, September 1998.

[23] M. Vojnovic and J.-Y. L. Boudec. On the long-run behavior of equation-based rate control. In *SIGCOMM 02*, Pittsburgh, PA, August 2002.