TCP NewReno: Slow-but-Steady or Impatient?

Nadim Parvez Anirban Mahanti Carey Williamson
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, AB, Canada T2N 1N4
Email: {parvez, mahanti, carey}@cpsc.ucalgary.ca

Abstract— **In this paper, we compare the throughputs of two different TCP NewReno variants, namely Slow-but-Steady and Impatient. We develop analytic throughput models of these variants as a function of round-trip time, loss event rate, and the burstiness of packet drops within a loss event. Our models build upon prior work on TCP Reno throughput modeling, but extend this work to provide an analytical characterization of the NewReno fast recovery algorithms. We validated our models using the ns-2 simulator. Our models accurately predict the steady-state NewReno throughput for a wide range of loss rates. Based on these models, we analytically determine the preferred operating regions for each TCP variant. Our results show that the Slow-but-Steady variant is comparable to or superior to the Impatient variant in all but the most extreme scenarios for network packet loss.**

Keywords: TCP, TCP NewReno, Congestion Control, Fast Recovery, Throughput Model

I. Introduction

Several variants of the Transmission Control Protocol (TCP) are used for reliable data transfer on the Internet. This paper develops bulk data transfer throughput models for the Slow-but-Steady and Impatient variants of TCP NewReno [9]. We use these models to provide new insight into the performance of the two NewReno variants.

Our work is motivated by three observations. First, although many analytic models of TCP's congestion-controlled throughput have been proposed, most have modeled TCP Reno [3], [6], [8], [11], [16], [18], [20], [21], [22], [24], rather than TCP NewReno. Second, recent measurement studies indicate that TCP NewReno is widely deployed on the Internet [19], [23]. Third, it seems reasonable to expect that TCP NewReno implementations will continue to be dominant, at least in the near future, in order to provide better support for TCP peers without SACK [9].

The Slow-but-Steady and Impatient variants of NewReno differ in their fast recovery behavior, specifically with respect to how they reset the retransmit timer. The Slow-but-Steady variant resets the timer after each partial[1] acknowledgement (ACK), and continues to plod along with small adjustments to the congestion window. The Impatient variant resets the retransmit timer only upon the receipt of the *first* partial ACK. Subsequent losses or partial ACKs typically trigger a coarse timeout, which in turn triggers a drastic congestion window reduction, and a renewed slow start phase to probe the (new) available capacity of the network.

RFC 3782 recommends the Impatient variant, but provides no justification for this recommendation [9]. Our results, however, show that Impatient outperforms Slow-but-Steady only in very unusual network packet loss conditions. We thus recommend Slow-but-Steady as the preferred NewReno variant.

The remainder of this paper is organized as follows. Section II presents an overview of NewReno congestion control. The throughput models for the NewReno variants are developed in Section III. Model validation and performance results are presented in Section IV. Section V provides further insight into the performance of the Impatient and Slow-but-Steady variants. Section VI presents our conclusions.

II. TCP NewReno Congestion Control

Modern TCP implementations incorporate congestion control algorithms that adapt the sending rate of the source according to perceived changes in available network bandwidth. This dynamic adaptation is achieved by computing the *congestion window size (cwnd)*, a TCP state variable that places an upper bound on the number of (maximally-sized) unacknowledged segments in the network. Depending on the congestion window adaptation policy in use, TCP may be classified as Tahoe [13], Reno [14], NewReno [9], SACK [7], or Vegas [5]. The following sections briefly discuss the components of NewReno congestion control. The reader is referred to references [9], [14], [27], [28] for a detailed treatment of TCP NewReno congestion control.

A. Slow Start and Congestion Avoidance

Let $cwnd$ and $ssthresh$ refer to the current congestion window size and the current slow start threshold, respectively. Then, if $cwnd < ssthresh$, receipt of a non-duplicate ACK results in $cwnd$ increasing by one segment. Thus, in the absence of segment loss, $cwnd$ doubles every round-trip time (RTT) until it reaches the slow start threshold value $ssthresh$. This algorithm is called the *slow start* algorithm [13].

If $cwnd \geq ssthresh$, $cwnd$ increases by $1/cwnd$ for each non-duplicate ACK received. This window evolution in which the congestion window size increases by about one segment every RTT is referred to as the *congestion avoidance* algorithm [13].

[1] An ACK that acknowledges some but not all of the outstanding data.

B. Fast Retransmit and Fast Recovery

During slow start or congestion avoidance, receipt of four back-to-back identical ACKs (referred to as "triple duplicate ACKs") causes the sender to perform *fast retransmit* [14]. In fast retransmit, the sender does the following. First, the segment implicitly requested by the triple duplicate ACK is retransmitted. Second, $ssthresh$ is set to $cwnd/2$. Third, $cwnd$ is set to $ssthresh$ (new) plus 3 segments. Following these steps, the sender enters *fast recovery* [9], [14].

Upon entering fast recovery, the sender continues to increase the congestion window by one segment for each subsequent duplicate ACK received. The intuition behind the fast recovery algorithm is that duplicate ACKs indicate the reception of some segments by the receiver, and thus can be used to trigger new segment transmissions. The sender transmits new segments if permitted by its congestion window.

TCP NewReno (unlike Reno) distinguishes between a "partial" ACK and a "full" ACK. A full ACK acknowledges all segments that were outstanding at the start of fast recovery, while a partial ACK acknowledges some but not all of this outstanding data. Unlike Reno, where a partial ACK terminates fast recovery, NewReno retransmits the segment next in sequence based on the partial ACK, and reduces the congestion window by one less than the number of segments acknowledged by the partial ACK. This window reduction, referred to as *partial window deflation*, allows the sender to transmit new segments in subsequent RTTs of fast recovery. On receiving a full ACK, the sender sets $cwnd$ to $ssthresh$, terminates fast recovery, and resumes congestion avoidance.

C. Slow-but-Steady versus Impatient

The Slow-but-Steady (SBS) variant of NewReno resets the retransmit timer on receipt of each partial ACK. Thus, the Slow-but-Steady variant can recover from multiple segment losses in the same window by potentially retransmitting one lost segment per RTT. The TCP sender remains in fast recovery mode until a full ACK is received.

The Impatient (IMP) variant of NewReno resets the retransmit timer only for the first partial ACK. Thus, the Impatient variant attempts to avoid lengthy fast recovery periods by invoking slow start following a timeout, and recovering lost segments using a Go-back-N approach.

Exactly when the Impatient variant experiences a timeout depends upon the retransmit timeout (RTO) estimation technique and the timer granularity. For example, if $2RTT < RTO < 3RTT$, Impatient will continue in fast recovery for at least 3 $RTTs$. Thus, if 4 or more segments are dropped from a large window, Impatient will incur a timeout. In RFC 3782, the Impatient variant is recommended over the Slow-but-Steady variant [9].

III. Throughput Models for TCP NewReno

This section develops simple stochastic models for the steady-state throughput of the Slow-but-Steady and the Impatient variants of TCP NewReno. We begin this section by outlining our assumptions. Model notation is summarized in Table I.

TABLE I
Model Notation

Parameter	Definition
p	Loss event rate
m	Avg. number of segment losses per loss event
R	Avg. round-trip time
RTO	Avg. duration of first timeout in a series of timeouts
W_{SBS}	Avg. of the peak $cwnd$ during CA (Slow-but-Steady)
W_{IMP}	Avg. of the peak $cwnd$ during CA (Impatient)

A. Model Assumptions

Our assumptions are similar to those in prior works (e.g., [6], [11], [18], [22], [25], [26]), except for those pertaining to segment losses.

Our models characterize the steady-state throughput of bulk transfers. TCP's 3-way connection establishment and initial slow start phases are ignored. We also assume that the sender always transmits full-sized (i.e., MSS) segments, that the receiver's advertised buffer space does not limit the congestion window, and that an ACK is sent for each received segment.

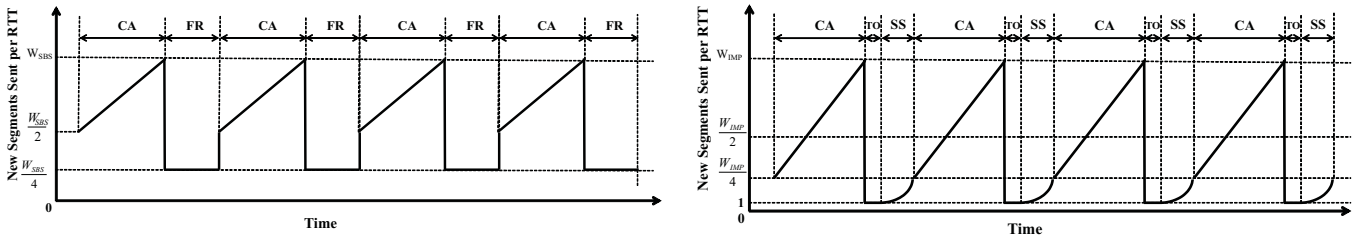
The evolution of the TCP congestion window is modeled in units of "rounds" [22]. The first round begins with the start of congestion avoidance; its duration is one RTT. All other rounds begin immediately after the previous round, and their durations are also one RTT. As in prior work, we assume that the round duration is larger than the time required to transmit segments in a round, and that the round duration is independent of the congestion window size.

We characterize segment losses in terms of loss events. A loss event begins with the first loss in a round that eventually causes TCP to transition from the congestion avoidance phase to either the fast recovery or the timeout phase. The occurrence of loss events is characterized by a Bernoulli process with parameter p. That is, p is the loss event rate. Within a loss event, it is assumed that there are a total of m segment losses. Furthermore, we assume that all loss events are identified by triple duplicate ACKs (i.e., $m < W - 2$), and all loss events therefore trigger fast recovery. Within a fast recovery, we assume that retransmitted segments are never lost.

B. Model for the Slow-but-Steady Variant

From our assumptions and using arguments analogous to those in [22], [25], it follows that TCP's segment transmissions can be viewed as a concatenation of statistically identical *cycles*, where each cycle consists of a congestion avoidance (CA) period followed by a fast recovery (FR) period, as illustrated in Figure 1(a). Therefore, the throughput of the flow can be determined by analyzing one of these cycles.

In the congestion avoidance period, TCP's window grows from $\frac{W_{SBS}}{2}$ to W_{SBS}, with the window size increasing by one segment per RTT. For m segment losses per loss event, the fast recovery period continues for m RTTs, with TCP transmitting



(a) Slow-but-Steady (b) Impatient

Fig. 1. Segment Transmission Cycles of the TCP NewReno Variants

approximately $\frac{W_{SBS}}{4}$ segments per RTT.[2] Hence, the number of segments S_{SBS} transmitted in a SBS cycle is:

$$S_{SBS}=\sum_{i=\frac{W_{SBS}}{2}}^{W_{SBS}} i+m\frac{W_{SBS}}{4}=\frac{W_{SBS}}{4}\left(3\frac{W_{SBS}}{2}+m+3\right). \quad (1)$$

From the Bernoulli loss event assumption, it follows that $S_{SBS}=1/p$. Therefore, we obtain:

$$W_{SBS}=\frac{-3p-mp+\sqrt{9p^2+6mp^2+m^2p^2+24p}}{3p}. \quad (2)$$

The elapsed time duration D_{SBS} for each SBS cycle is:

$$D_{SBS}=\left(\frac{W_{SBS}}{2}+1\right)R+mR. \quad (3)$$

Therefore, the steady-state throughput T_{SBS} of a Slow-but-Steady NewReno bulk transfer is:

$$T_{SBS}=\frac{S_{SBS}}{D_{SBS}}=\frac{1/p}{R\left(\frac{W_{SBS}}{2}+m+1\right)}, \quad (4)$$

with W_{SBS} computed using Equation 2.

C. Model for the Impatient Variant

In this section, we derive a model for the Impatient variant. For this model, we are interested in the timeout case, which occurs when the number of segment losses exceeds some critical value. That is, we restrict our analysis to the case when $m \geq m^*$, where m^* is the number of segment losses required per loss event to cause a coarse timeout before fast recovery terminates with a full ACK. For $m < m^*$, Impatient and Slow-but-Steady behave identically, so Equation 4 can be used.

For $m \geq m^*$, the segment transmissions for the Impatient variant can be viewed as a concatenation of statistically identical *cycles*, where each cycle consists of a congestion avoidance (CA) period, an incomplete fast recovery period, a timeout (TO) period, and a slow start (SS) period, as shown[3] in Figure 1(b).

Consider a cycle as shown in Figure 1(b). Upon occurrence of a loss event, the TCP sender enters fast recovery and reduces its congestion window from W_{IMP} to $\frac{W_{IMP}}{2}$ and sets the slow start threshold, $ssthresh$, to $\frac{W_{IMP}}{2}$. Since Impatient does not schedule a new timer for subsequent partial ACKs, the TCP sender experiences a timeout after $RTT + RTO$. When the timeout occurs, the slow start threshold is first set to half of the current congestion window size (i.e., $ssthresh = \frac{W_{IMP}}{4}$), and then the congestion window size is set to one segment. As a result, TCP re-enters slow start. In the slow start phase, the congestion window size doubles every round until the slow start threshold is reached. To simplify our model, we count the last round of slow start (i.e., its segments and duration) as being part of congestion avoidance. This simplification does not affect the throughput formulation.

The number of segments S_{IMP} sent in an IMP cycle is the sum of the number of segments transmitted during the slow start and congestion avoidance periods. We do not count the new transmissions during the incomplete fast recovery, since TCP NewReno forgets all outstanding data upon a timeout[4]. Therefore,

$$S_{IMP}=(1+2+\cdots+\tfrac{W_{IMP}}{8})+\sum_{i=\frac{W_{IMP}}{4}}^{W_{IMP}} i=\tfrac{15}{32}W_{IMP}^2+\tfrac{7}{8}W_{IMP}-1. \quad (5)$$

From the Bernoulli loss event assumption, it follows that $S_{IMP}=1/p$. Hence, we obtain:

$$W_{IMP}=\frac{-14p+2\sqrt{169p^2+120p}}{15p}. \quad (6)$$

The duration D_{IMP} of an IMP cycle is the sum of the durations of slow start, the congestion avoidance, the incomplete recovery (equal to one RTT), and the timeout periods. Hence,

$$D_{IMP}=\left(log\,\tfrac{W_{IMP}}{8}+1\right)R+\left(\tfrac{3W_{IMP}}{4}+1\right)R+(R+RTO). \quad (7)$$

Thus, the steady-state throughput T_{IMP} of an Impatient NewReno bulk transfer is:

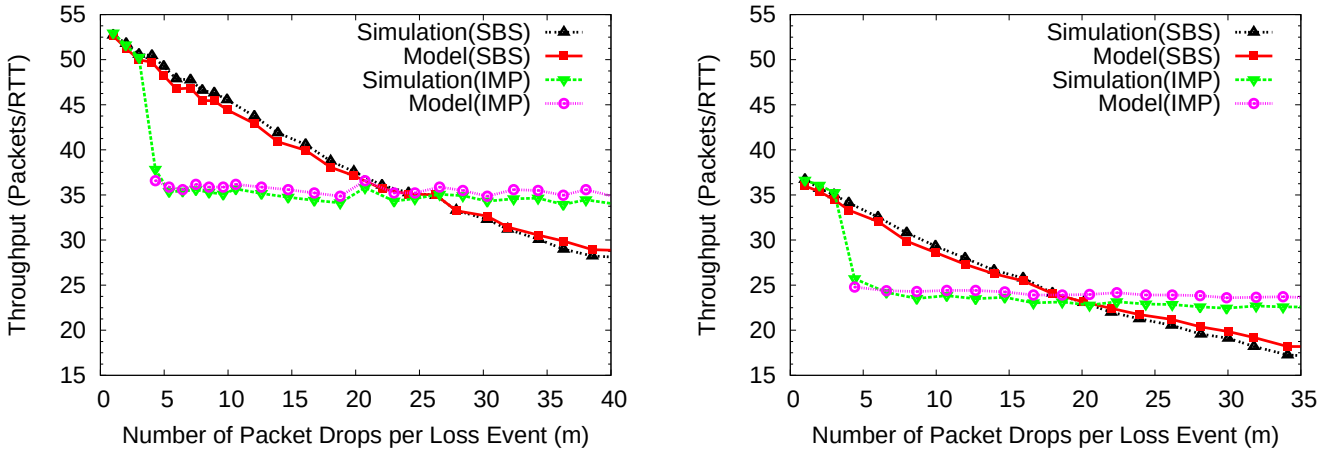
$$T_{IMP}=\frac{S_{IMP}}{D_{IMP}}=\frac{1/p}{\left(log\,\frac{W_{IMP}}{8}+1\right)R+\left(\frac{3W_{IMP}}{4}+1\right)R+(R+RTO)}, \quad (8)$$

with W_{IMP} obtained using Equation 6.

[2]During fast recovery, the number of segment transmissions per RTT ranges from 0 to $(\frac{W_{SBS}}{2}-1)$, depending on the value of m. Experiments and analysis suggest that $\frac{W_{SBS}}{4}$ is a reasonable (but conservative) estimate of the average number of transmissions per RTT.

[3]The illustration does not show the incomplete fast recovery period.

[4]Technically, packets transmitted during fast recovery (due to the reception of partial ACKs) could be acknowledged before the RTO occurs, but this depends on the RTO value. Consider the average case, in which the loss event (with m packet losses) starts in the middle of a round transmitting W packets. If $RTO > 2RTT$, then TCP would receive 2 partial ACKs, acknowledging approximately $2W/m$ packets before the timeout occurs. Thus TCP would have $W/2 + 2W/m$ packets acknowledged since the start of the last round. As an approximation, we assume that the number of packets successfully acknowledged before the timeout is W.



(a) $p = 0.05\%$ (b) $p = 0.10\%$

Fig. 2. Model Accuracy and Performance with Bernoulli Loss Events

IV. Model Validation and Results

This section validates the proposed NewReno throughput models using the *ns-2* network simulator [1]. The results reported in this section also identify the regimes in which Slow-but-Steady outperforms Impatient, and vice versa.

A. Network and Traffic Models

The results reported here are for a simple dumbbell network topology with a single common bottleneck between all sources and sinks. Each source/sink pair is connected to the bottleneck link via a high bandwidth access link. The propagation delays of the access links are varied to simulate the desired round-trip delay between a source/sink pair. The flows that are being actively monitored are referred to as the "foreground" flows, with all other traffic considered "background" flows.

All experiments have two foreground NewReno flows – one Slow-but-Steady and one Impatient. The receiver-side buffers of the foreground flows are large enough that the buffer space advertisements do not constrain the congestion window size. Experiments are reported for both Drop-Tail and RED management policies at the bottleneck queue. For RED queue management, the $minthresh$ and the $maxthresh$ are set to $1/3$ and $2/3$ of the corresponding queue size [10].

Experiments with background traffic consider a mix of long duration FTP transfers and short duration HTTP sessions. The background HTTP sessions are simulated using a model similar to that in [17], [25]. Specifically, each HTTP session consists of a unique client/server pair. The client sends a single request packet across the (reverse) bottleneck link to its dedicated server. The server, upon receiving the request, uses TCP to send the file to the client. Upon completion of the data transfer, the client waits for a period of time before issuing the next request. These waiting times are exponentially distributed and have a mean of 500 ms. The file sizes are drawn from a Pareto distribution with mean 48 KB and shape 1.2 to simulate the observed heavy-tailed nature of HTTP transfers [4].

Background FTP and HTTP sessions use TCP NewReno (Slow-but-Steady) with a maximum congestion window size of 64 KB. The packet size is 1 KB. All packets are of identical size except HTTP request packets and possibly the last packet of each HTTP response. The round-trip propagation delays of the background flows are uniformly distributed between 20 ms and 460 ms, to model WAN delays similar to those reported in the Internet measurement literature [2], [15].

From the simulations, the necessary input parameters for the analytical models are obtained. Each experiment simulates 1000 seconds. Results are reported using data from the last 750 simulated seconds of the simulation.

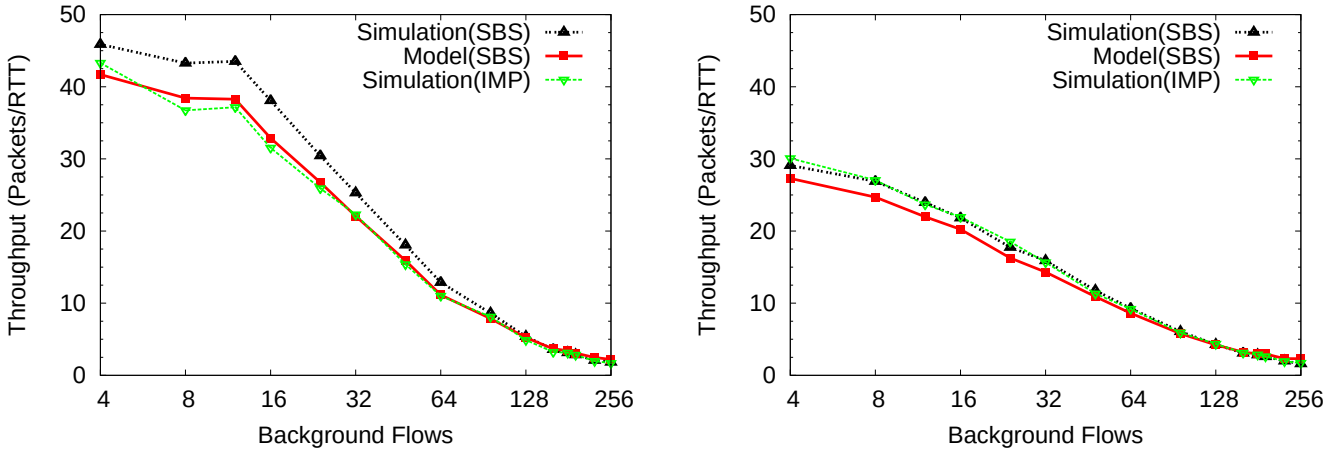
B. Bernoulli Loss Events

Before validating the model in the presence of background traffic, validation is carried out in isolation. The configuration considered here consists of two foreground NewReno flows with 70 ms RTT traversing a 45 Mbps bottleneck link. A separate packet drop module was placed on the receiver's access link of each flow to simulate "non-congestion" packet losses, as might be observed, for example, with wireless access technologies [12]. The packet drop module has two parameters, namely p and m.

Figure 2 shows the results for varying m ($m > 0$) with two different values[5] of p. Figure 2(a) shows the throughput results from the simulations and the analytic models when $p = 0.05\%$ and m ranges between 1 and 40. Figure 2(b) shows results for $p = 0.10\%$ with m ranging from 1 to 35.

For both Impatient and Slow-but-Steady, the proposed models accurately track the simulation throughput over the entire range of m values. The prediction error of a model is defined as $|simulation - model|/simulation$, where $simulation$ and $model$ refer to the throughput values from the simulation and model, respectively. For the Slow-but-Steady model, the prediction error ranges from 0.15% to 3.11% when $p = 0.05\%$,

[5]The models work well even for values of p as large as 5%. See Table II in Section IV-C.



(a) Drop-Tail (b) RED

Fig. 3. Model Accuracy and Performance with Background FTP/HTTP Traffic

and from 0.20% to 6.29% when $p = 0.10\%$. For the Impatient model, the prediction error ranges from 0.45% to 3.34% for $p = 0.05\%$, and from 0.74% to 5.54% when $p = 0.10\%$.

The Slow-but-Steady model slightly underestimates throughput when m is small and overestimates when m is large. This trend can be attributed to the approximations used for estimating the segment transmissions during fast recovery. Qualitatively similar results were obtained from experiments with other values of p, ranging up to 5%.

Figure 2 indicates three distinct performance regions: a region where Slow-but-Steady is superior; a region where Impatient is superior; and an *equality region* where Impatient and Slow-but-Steady provide comparable performance. The equality region prevails for $m < m^*$, where $m^* = 4$ in our experiments.

Slow-but-Steady is superior for small to moderate values of m. While Slow-but-Steady spends a significant amount of time in fast recovery, the penalty is not as bad as a coarse timeout. In Figure 2(a), the SBS superior region spans m values between 4 and 25, and in Figure 2(b) it spans between 4 and 20.

The Impatient variant is superior when m is large, and the penalty of a timeout is less than a lengthy fast recovery. The IMP superior region occurs for $m \geq 26$ in Figure 2(a), and for $m \geq 21$ in Figure 2(b).

C. FTP/HTTP Background Traffic

The next experiment considers a 15 Mbps bottleneck link with a queue of capacity 150 packets. The background traffic consists of a mix of 25% FTP and 75% HTTP flows. The total number of background flows is varied from 4 to 256. There are two foreground flows: one Impatient NewReno flow and one Slow-but-Steady NewReno flow. Both foreground flows have a round-trip propagation delay of 50 ms.

Figure 3 shows the simulated throughputs of the NewReno variants along with the Slow-but-Steady model predictions. Figure 3(a) is for a Drop-Tail bottleneck queue, while Figure 3(b) is for a RED queue.

In both graphs in Figure 3, the throughputs for the foreground flows decrease (as expected) as the number of competing background flows increases. The loss event rate and the packet loss rate also increase with the number of background flows. Table II shows the loss event rate, packet loss rate, simulation throughput, and model prediction for Slow-but-Steady in the Drop-Tail experiments.

Figure 3(a) and Table II show that the analytic model for Slow-but-Steady tracks the simulation throughput reasonably well when the number of background flows is between 4 and 200. In this range, the loss event rate ranges from 0.07% to 5.31%, and the number of drops per loss event is between 2.11 and 4.16.

For 4 to 200 background flows, the model prediction is below the simulation throughput by 9.78% on average. One reason is our somewhat conservative estimate of the average number of segment transmissions per RTT in fast recovery. Simulation and analysis indicate that the average number of segment transmissions per RTT during fast recovery is approximately $\frac{W_{SBS}}{2}$ for the m values experienced, but we used $\frac{W_{SBS}}{4}$ in the throughput model.

With 200 or more background flows, timeouts dominate, and the model overestimates throughput since it does not capture direct timeout or timeout due to losses of retransmitted segments (which occur in the simulation). The largest prediction error observed is 20.64% with 256 background flows (see Table II).

In this experiment, the Impatient variant experiences 1.93 to 2.93 packet drops per loss event, on average. The number of drops is below $m^* = 4$. Since the Impatient throughput model does not apply in this regime, the corresponding model results are not shown in Figure 3.

Figure 3(b) shows that the performance differences between the two TCP variants diminish when a RED queue is used. The reason is that RED queues reduce the burstiness of packet drops. In our experiments, most loss episodes consisted of a single packet drop. The ratio of packet loss rate to loss

TABLE II

MEASURED CHARACTERISTICS FOR SBS FLOW UNDER DROP-TAIL

Background Flows	Loss Event Rate	Loss Rate	Simulation (Pkts/RTT)	Prediction (Pkts/RTT)
4	0.07%	0.20%	45.88	41.69
8	0.08%	0.30%	43.28	38.42
12	0.08%	0.30%	43.52	38.27
16	0.10%	0.42%	38.08	32.88
24	0.15%	0.58%	30.44	26.74
32	0.21%	0.81%	25.29	22.06
48	0.38%	1.20%	18.06	15.91
64	0.66%	1.99%	12.89	11.15
96	1.24%	3.40%	8.61	7.87
128	2.43%	5.74%	5.38	5.23
160	4.06%	8.97%	3.58	3.67
180	4.74%	10.25%	3.14	3.42
192	5.31%	11.22%	2.85	3.08
224	7.28%	14.97%	2.08	2.47
256	8.29%	16.34%	1.83	2.21

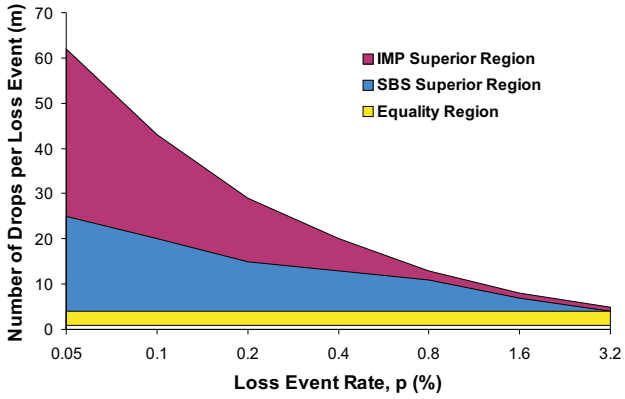


Fig. 4. Fast Recovery Regions for NewReno Variants

event rate is about 1.22 with 4 background flows, and about 1.40 with 256 background flows. The Slow-but-Steady model tracks the simulation throughput reasonably well up to 200 background flows, with an average prediction error of 7.07%.

With Drop-Tail queues, we observed bursty packet losses. The average number of packet drops per loss event ranged from 2.11 to 4.16. In most cases, the variants operate in the equality region ($1 \leq m < 4$), although in a few cases, they operate in the SBS superior region. The protocols never operate in the IMP superior region, since there are few packet drops per loss event.

When the number of background flows ranges from 4 to 64, the average throughput difference (in simulation) between the Slow-but-Steady and the Impatient variant is approximately 13.59%. As the background load increases (from 64 to 256 flows), the loss event rate experienced by the foreground flows increases. At higher loss event rates, the maximum attainable congestion window size decreases substantially, thus increasing the probability of timeout events. Since the treatment of timeouts is identical in Slow-but-Steady and Impatient, the throughput difference between them decreases. In this range, the average throughput difference between Slow-but-Steady and Impatient is approximately 5.54%.

V. DISCUSSION

In this section, we use the analytic models to obtain further insight into the performance of the NewReno variants. Specifically, we focus on finding the range of m values for the three performance regions, namely the SBS superior region, the IMP superior region, and the equality region.

Figure 4 shows the different regions for a range of loss event rates. The following observations are evident from this graph:

- The equality region is independent of the loss event rate. This region is bounded by $1 \leq m \leq m^*$, where $m^* = 4$ in this plot.
- The SBS superior region lies above the equality region, representing moderately higher loss scenarios. This region is upper bounded by a threshold value m_t, where m_t can be computed by analytic intersection of T_{SBS} and T_{IMP} for a particular p. Note that m_t must satisfy $m_t \leq (W_{SBS} - 3)$. Analytic computation produces $m_t = 27$ for $p = 0.05\%$ and $m_t = 21$ for $p = 0.10\%$, consistent with the simulation results reported in Figure 2.
- The IMP superior region lies above the SBS superior region. It prevails for more extreme loss scenarios, but is also bounded. The upper bound for the IMP superior region is at $(W_{IMP} - 3)$, which is the maximum number of segment losses that can be tolerated, while still triggering fast recovery.

Figure 4 can be used to determine the network conditions under which a certain TCP variant is preferable. We make such an argument for each of the two TCP NewReno variants.

The Impatient variant is superior when the loss event rate is low, but the number of packet drops per loss event is high (e.g., at least 26 drops at $p = 0.05\%$). These network conditions are rather rare, and did not arise in our simulations. These conditions may apply for some network scenarios, such as high bandwidth-delay product networks, satellite networks, or high performance grid computing applications.

Quantitatively, IMP is superior at $p = 0.05\%$ when $m_t/W_{IMP} \geq \frac{1}{3}$. That is, IMP is preferred if at least one-third of the packets in the (large) window are dropped. Similarly, IMP is superior at $p = 0.25\%$ if $m_t/W_{IMP} \geq \frac{1}{2}$. While these results show that there exist conditions under which IMP is superior, we believe that such scenarios may be rare on the Internet.

When the loss event rate is even higher, the Impatient variant offers little advantage. In particular, at a high loss event rate ($p > 0.8\%$), the congestion window becomes small, and the IMP superior region becomes very thin (and it requires almost a full window of packets to be dropped). TCP is unlikely to operate consistently here in practice, since most packet drops lead directly to a timeout.

The SBS superior region is broader, and more likely to arise in practice. For example, as long as Slow-but-Steady loses less than half of the segments from a window at $p = 0.25\%$, it

outperforms Impatient.

Although our models do not capture direct timeout, we still believe that the results in Figure 4 are useful and accurate. At a loss event rate of 3% or higher, the IMP superior and the SBS superior regions essentially vanish. Based on the results in Table II, we believe that our model predictions are reasonably accurate up to loss event rates of 5%.

VI. Conclusions

This paper compares the throughputs of two different variants of TCP NewReno. It provides analytic throughput models for each variant, expressing steady-state throughput in terms of RTT, RTO, and losses (i.e., loss event rate and the burstiness of packet losses).

Our NewReno throughput models are formulated using a flexible two-parameter loss model that can better capture the dynamics of loss events for TCP. We have validated our models with extensive *ns-2* simulation experiments. Our results show that the proposed models can predict steady-state TCP NewReno throughput for a wide range of loss scenarios.

Using our models, we derived a fast recovery region plot that explicitly shows the operating conditions under which different TCP variants are superior. Our results show that the Impatient variant is superior only under very extreme network conditions (i.e., low loss event rates, but many packet drops per loss event). The Slow-but-Steady variant is superior to Impatient, or comparable to Impatient, in all other operating regimes, although RED queues diminish the performance differences observed. We thus recommend Slow-but-Steady as the preferred variant of TCP NewReno, contrary to RFC 3782.

Acknowledgements

Financial support for this work was provided by the Natural Sciences and Engineering Research Council (NSERC) of Canada, as well as by the Informatics Circle of Research Excellence (iCORE) in the Province of Alberta. The authors thank Naimul Basher for his help with graph plotting, and the anonymous ICC 2006 reviewers for their very careful reading of the initial version of this paper.

References

[1] The NS Project. The Network Simulator: ns-2. http://www.isi.edu/nsnam/ns.

[2] M. Allman. A Web Server's View of the Transport Layer. *ACM Computer Communications Review*, 30(5):10–20, October 2000.

[3] E. Altman, K. Avrachenkov, and C. Barakat. A Stochastic Model of TCP/IP with Stationary Random Losses. In *Proc. of ACM SIGCOMM*, pages 231–242, Stockholm, Sweden, August 2000.

[4] M. Arlitt and C. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Transactions On Networking*, 5(5):631–645, October 1997.

[5] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. of ACM SIGCOMM*, pages 24–35, New York, USA, August 1994.

[6] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proc. of IEEE INFOCOM*, pages 1742–1751, Tel-Aviv, Israel, March 2000.

[7] K. Fall and S. Floyd. Simulation-Based Comparisons of Tahoe, Reno, and Sack TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.

[8] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks. *ACM Computer Communication Review*, 21(5):30–47, 1997.

[9] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, April 2004.

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

[11] M. Goyal, R. Guerin, and R. Rajan. Predicting TCP Throughput from Non-Invasive Network Sampling. In *Proc. of IEEE INFOCOM*, pages 180–189, New York, USA, June 2002.

[12] A. Gurtov and S. Floyd. Modelling Wireless Links for Transport Protocols. *ACM Computer Communication Review*, 34(2):85–96, April 2004.

[13] V. Jacobson. Congestion Avoidance and Control. In *Proc. of ACM SIGCOMM*, pages 314–329, Stanford, USA, August 1988.

[14] V. Jacobson. Berkeley TCP evolution from 4.3-Tahoe to 4.3 Reno. In *Proc. of the 18th Internet Engineering Task Force*, Vancouver, Canada, August 1990.

[15] H. Jiang and C. Dovrolis. Passive Estimation of TCP Round-trip Times. *ACM Computer Communication Review*, 32(3):75–88, July 2002.

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

[17] A. Mahanti, D. Eager, and M. Vernon. Improving Multirate Congestion Control Using a TCP Vegas Throughput Model. *Computer Networks*, 48(2):113–136, June 2005.

[18] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, 27(3):67–82, July 1997.

[19] A. Medina, M. Allman, and S. Floyd. Measuring the Evolution of Transport Protocols in the Internet. *Computer Communications Review*, 35(2):37–51, April 2005.

[20] A. Misra and T. Ott. The Window Distribution for Idealized TCP Congestion Avoidance with Variable Packet Loss. In *Proc. of IEEE INFOCOM*, pages 1564–1572, New York, USA, March 1999.

[21] V. Misra, W. Gong, and D. Towsley. Stochastic Differential Equation Modeling and Analysis of TCP-Windowsize Behavior. In *Proc. of IFIP PERFORMANCE*, Istanbul, Turkey, October 1999.

[22] J. Padhye, V. Firioiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. of ACM SIGCOMM*, pages 303–314, Vancouver, Canada, September 1998.

[23] J. Padhye and S. Floyd. On Inferring TCP Behavior. In *Proc. of ACM SIGCOMM*, pages 287–298, San Deigo, USA, August 2001.

[24] V. Paxson. Empirically Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, 1994.

[25] C. Samios and M. Vernon. Modeling the Throughput of TCP Vegas. In *Proc. of ACM SIGMETRICS*, San Diego, USA, June 2003.

[26] B. Sikdar, S. Kalyanaraman, and K. Vastola. An Integrated Model for the Latency and Steady-State Throughput of TCP Connections. *Performance Evaluation*, 46(2-3):139–154, September 2001.

[27] W. Stevens. *TCP/IP Illustrated Vol. 1: The Protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 1994.

[28] W. Stevens and G. Wright. *TCP/IP Illustrated Vol. 2: The Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 1995.