An Analytic Throughput Model for TCP NewReno

Nadim Parvez Anirban Mahanti Carey Williamson

***Abstract*—This paper develops a simple and accurate stochastic model for the steady-state throughput of a TCP NewReno bulk data transfer as a function of round-trip time and loss rate. Our model builds upon extensive prior work on TCP Reno throughput models but differs from these prior works in three key aspects. First, our model introduces an analytical characterization of the TCP NewReno fast recovery algorithm. Second, our model incorporates a more accurate formulation of NewReno's timeout behaviour. Third, our model is formulated using a flexible two-parameter loss model that can better represent the diverse packet loss scenarios encountered by TCP on the Internet.**

We validated our model by conducting a large number of simulations using the *ns-2* simulator and by conducting emulation and Internet experiments using a NewReno implementation in the BSD TCP/IP protocol stack. The main findings from the experiments are: (1) the proposed model accurately predicts the steady-state throughput for TCP NewReno under a wide range of network conditions; (2) TCP NewReno significantly outperforms TCP Reno in many of the scenarios considered; and (3) using existing TCP Reno models to estimate TCP NewReno throughput may introduce significant errors.

I. Introduction

The Transmission Control Protocol (TCP) [33] provides reliable, connection-oriented, full-duplex, unicast data delivery on the Internet. Modern TCP implementations also include congestion control mechanisms that adapt the source transmission behaviour to network conditions by dynamically computing the *congestion window* size. The goal of TCP congestion control is to increase the congestion window size if there is additional bandwidth available on the network, and decrease the congestion window size when there is congestion. It is widely agreed that the congestion control schemes in TCP provide stability for the "best effort" Internet. These mechanisms increase network utilization, prevent starvation of flows, and ensure inter-protocol fairness [11].

In today's Internet, several variants of TCP are deployed. These variants differ with respect to their congestion control and segment loss recovery techniques. The basic congestion control algorithms namely *slow start*, *congestion avoidance*, and *fast retransmit*, were introduced in TCP Tahoe [18]. In TCP Reno [19], the *fast recovery* algorithm was added. This algorithm uses duplicate acknowledgements (ACKs) to trigger the transmission of new segments during the recovery phase, so that the network "pipe" does not empty following a fast retransmit. TCP NewReno introduced an *improved* fast recovery algorithm that can recover from multiple losses in a single window of data, avoiding many of the retransmission timeout events that Reno experiences [13]. TCP's selective acknowledgement (SACK) option was proposed to allow receivers to ACK out-of-order data [8]. With SACK TCP, a sender may recover from multiple losses more quickly than with NewReno. The aforementioned TCP variants use segment losses to estimate available bandwidth. TCP Vegas uses a novel congestion control mechanism that attempts to detect congestion in the network before segment loss occurs [6]. TCP Vegas, however, is not widely deployed on the Internet today.

Analytic modelling of TCP's congestion-controlled throughput has received considerable attention in the literature (e.g., [3], [5], [7], [10], [16], [22], [23], [25], [27]–[29], [32], [35]–[37]). These analytical models have: (1) improved our understanding of the sensitivity of TCP to different network parameters; (2) provided insight useful for development of new congestion control algorithms for high bandwidth-delay networks and wireless networks; and (3) provided a means for controlling the sending rate of non-TCP flows such that network resources may be shared fairly with competing TCP flows. Most of these throughput models are based on TCP Reno [3], [7], [10], [16], [23], [25], [27]–[29], [32], while some models are based on SACK [36], [37], Vegas [35], and NewReno [22]. A detailed NewReno throughput model, however, seems missing from the literature.

This paper develops an analytic model for the throughput of a TCP NewReno bulk data transfer as a function of round-trip time (RTT) and loss rate. Our work is motivated, in part, by recent measurement studies that indicate that TCP NewReno is widely deployed on the Internet [26], [30]. Furthermore, RFC 3782 indicates that NewReno is preferable to Reno, since NewReno provides better support for TCP peers without SACK [13].

Our TCP NewReno throughput model builds upon the well-known Reno model proposed by Padhye *et al.* [29] but differs with respect to the latter model in three key points. First, we explicitly model the fast recovery algorithm of TCP NewReno. Depending on the segment loss characteristics, a NewReno flow may spend a significant time in the fast recovery phase, sending per RTT an amount of data approximately equal to the slow start threshold. Second, we present an accurate formulation of NewReno's timeout behaviour, including the possibility of incurring a timeout following an unsuccessful fast recovery. Third, our model assumes a flexible two-parameter loss model that can capture both the frequency and the burstiness of losses. In particular, it is important to model both the loss event rate and the burstiness of losses within a loss event because these two characteristics have orthogonal effects on TCP: a loss event triggers either fast recovery or a timeout, whereas the burstiness of losses determines the duration of the fast recovery period.

Our TCP NewReno model differs substantially from Kumar's NewReno model [22]. First, Kumar's model was de-

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 are with the Department of Computer Science, University of Calgary, Canada (email:{parvez, mahanti, carey}@cpsc.ucalgary.ca).

veloped for the local area network environment and did not consider the effect of propagation delay on TCP throughput. Propagation delays cannot be ignored in environments such as the Internet and our model explicitly models the impact of propagation delay. Second, Kumar's model, unlike the model presented in this paper, does not have a closed form. Specifically, throughput estimation using the Kumar model [22] requires use of numerical methods to compute the expected window size and the expected cycle time. In contrast, our model provides a simple closed form for throughput computation. The third (and probably the most significant) difference between the two models is with respect to the modelling of NewReno's fast recovery behaviour. In Kumar's work, the transmission of new segments and the duration of fast recovery is not explicitly modelled; his model only considers the probability of TCP transitioning to fast recovery. The improved fast recovery algorithm is NewReno's key innovation with respect to its parent Reno. We explicitly model TCP NewReno's fast recovery behaviour in detail.

We validated our model by conducting a comprehensive set of simulations using the *ns-2* simulator. In addition, we also validated our model by experimenting with the TCP NewReno implementation in the BSD TCP/IP protocol stack in an emulation environment, and with Internet experiments. Our results show that the proposed model can predict steady-state TCP NewReno throughput for a wide range of network conditions.

The TCP Reno throughput model by Padhye *et al.* [29] has been extensively applied in a number of diverse areas including TCP friendly rate control [12], [24], active queue management [9], and overlay bandwidth management [17], [21]. The extensive application of this Reno model prompted us to consider how accurately this model can estimate TCP NewReno's throughput. Our experimental evaluation quantifies the significant throughput underestimation of the Reno model when used in networks dominated by NewReno flows. To highlight the performance improvements provided by the improved fast recovery algorithm of NewReno, we compare the performance of Reno and NewReno under identical network conditions. Our simulation results illustrate the performance advantages provided by NewReno's improved fast recovery algorithm. We do not present any comparisons with Kumar's NewReno throughput model because that model was developed for a fundamentally different network environment, and furthermore, has not been experimentally validated [22].

The remainder of this paper is organized as follows. Section II presents an overview of NewReno's fast recovery algorithm and our modelling assumptions. The proposed analytic model for TCP NewReno throughput is presented in Section III. The model is validated using simulations in Section IV, network emulations in Section V, and Internet experiments in Section VI. Section VII concludes the paper.

II. Background and Assumptions

A. The NewReno Fast Recovery Algorithm

This section presents an overview of NewReno's improved fast recovery algorithm [13]. All other congestion control components of NewReno, namely slow start, congestion avoidance, and fast retransmit, are identical to that of Reno. The reader is referred to references [19], [39], [40] for a detailed treatment of TCP Reno congestion control.

During congestion avoidance, receipt of four back-to-back identical ACKs (referred to as "triple duplicate ACKs") causes the sender to perform fast retransmit and to enter fast recovery. In fast retransmit, the sender does the following:

1) retransmits the lost segment;
2) sets the slow start threshold $ssthresh$ to $cwnd/2$ (where $cwnd$ is the current congestion window size); and
3) sets $cwnd$ to $ssthresh$ (new) plus 3 segments.

Upon entering fast recovery (FR), 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 that some segments are reaching the destination, and thus can be used to trigger new segment transmissions. The sender can transmit new segments if permitted by its congestion window.

In TCP Reno, receipt of a non-duplicate ACK results in *window deflation*: $cwnd$ is set to $ssthresh$ (i.e., the congestion window size in effect when the sender entered FR), FR terminates, and normal congestion avoidance behaviour resumes. When multiple segments are dropped from the same window of data, Reno may enter and leave FR several times, causing multiple reductions of the congestion window.

TCP NewReno modifies Reno's FR behaviour on receipt of a non-duplicate ACK, by distinguishing between a "full" ACK (FA) and a "partial" ACK (PA). A full ACK acknowledges all segments that were outstanding at the start of FR, whereas a partial ACK acknowledges some but not all of this outstanding data. Unlike Reno, where a partial ACK terminates FR, 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[1] by the partial ACK. Thus NewReno recovers from multiple segment losses in the same window by retransmitting one lost segment per RTT, remaining in FR until a full ACK is received.

On receiving a full ACK, NewReno sets $cwnd$ to $ssthresh$, terminates FR, and resumes congestion avoidance.

B. Assumptions

This section outlines our assumptions regarding the application, the sender/receiver, and the network. Except for the segment loss model, all our assumptions are similar to those in prior work (e.g., [7], [16], [25], [29], [35], [36]).

1) Application Layer: Our model focuses on the steady-state throughput for TCP bulk transfers. We consider an application process that has an infinite amount of data to send from a source node to a destination node.

2) TCP Sender and Receiver: Our model assumes that the sender is using the TCP NewReno congestion control algorithm. The sender always transmits full-sized (i.e., MSS) segments whenever the congestion window allows it to do so.

[1]This window reduction strategy is referred to as *partial window deflation*. In *full window deflation*, $cwnd$ is set to $ssthresh$ when partial ACKs are received. The current NewReno proposal in RFC 3782 recommends the partial window deflation option.

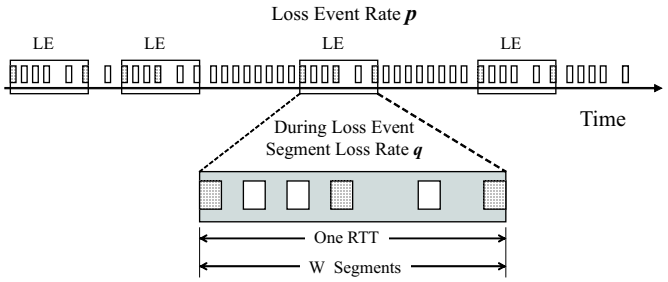


Fig. 1. The Two-Parameter Segment Loss Model

We assume that the receiver's buffer space advertisements do not limit the congestion window size, and that an ACK is sent for each received segment. These assumptions can be relaxed at the cost of somewhat more complex models using arguments similar to those in prior work [16], [29].

Similar to assumptions in other bulk transfer models, our analysis ignores TCP's three-way connection establishment phase and initial slow start phase because the congestion avoidance algorithm dominates during the TCP data transfer.

3) Latency Model: The latency of the TCP transfer is measured in terms of "rounds". 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 also last one RTT. The only exception is the round that terminates fast recovery and switches to congestion avoidance: its duration could be shorter than one RTT.

As in prior work, we assume that the round duration exceeds the time required to transmit segments in a round, and that round duration is independent of the congestion window size. Segment transmission may be bursty or arbitrarily spaced within the round.

4) Loss Model: Our work introduces a novel two-parameter segment loss model that can reflect both the frequency and the burstiness of loss events. We define a loss event (LE) to begin with the first segment loss in a round that eventually causes TCP to transition from the congestion avoidance phase to either the fast recovery phase or the timeout phase.

For a congestion window size of $W^{'}$, all losses within the next $W^{'}$ segments (starting from the first loss) are considered part of the same LE. This hierarchical relationship between an LE and losses within an LE is illustrated in Figure 1. Note that an LE can start at any segment, but once it starts, it spans at most one RTT (equivalently, $W^{'}$). A Bernoulli process with parameter p is used to characterize the occurrence of LEs. Segments transmitted during an LE are assumed to be lost independently with probability q (i.e., parameter q captures the "burstiness" of the losses within an LE).

Many throughput models in the literature assume a restricted version of the foregoing loss model (e.g., [7], [16], [35], [36]). These models assume that following the first segment loss in a round, all subsequent segments transmitted in that round are lost. This approximation works well for routers with DropTail queueing. In other contexts, such as routers with RED active queue management, and wireless networks where losses occur because of the physical channel characteristics, this approximation is less applicable.

TABLE I
MODEL NOTATION

Parameter	Definition
p	Loss event rate
q	Segment loss rate within a loss event
R	Average round-trip time
RTO	Average duration of first timeout in a series of timeouts
W	Average of the peak congestion window size

Estimation of the two parameters p and q depends on the application of the model. For example, estimation techniques outlined in [12] can be used when TCP friendly rate control is needed for non-TCP flows. In some scenarios, however, estimation of the two parameters might be difficult (e.g., non-invasive sampling of the network [16]). Our experience with validating the model suggests that estimating the loss event rate p should suffice, as the approximation $q = p$ yields reasonably accurate estimates of TCP throughput in most scenarios. In Section IV-H, we explore this approximation, and also show how q may be estimated from measured characteristics of the flow.

III. THE ANALYTIC MODEL

This section develops the stochastic throughput model for TCP NewReno bulk data transfer. The model is developed in two steps. In Section III-A, the model is developed assuming that all loss events are identified by triple duplicate ACKs. Subsequently, in Section III-B, an enhanced model is developed that handles both triple duplicate ACKs and timeouts. The model notation is summarized in Table I.

A. Model without Timeout (NoTO)

In this section, we assume that all loss events are identified by triple duplicate ACKs, so that no timeouts occur. The model developed here is referred to as the "NoTO" model.

Ignoring the initial slow start phase, it follows from the arguments given in [29], [35] that the evolution of the congestion window can be viewed as a concatenation of statistically identical *cycles*, where each cycle consists of a congestion avoidance period (where the window size increases by one segment per round from $W/2$ to W), followed by detection of segment loss and a fast recovery period. Each of these cycles is called a Congestion Avoidance/Fast Recovery (CAFR) period.

The throughput of the flow can be computed by analyzing one such CAFR cycle. Let S_{CAFR} be the expected number of segments successfully transmitted during a CAFR period. Let D_{CAFR} denote the expected time duration of the period. Then the average throughput of the flow is:

$$T_{NoTO} = \frac{S_{CAFR}}{D_{CAFR}}. \quad (1)$$

Before determining the expectations of the variables in Equation 1, let us consider the illustration in Figure 2. Figure 2 shows the segment transmissions per round in two adjacent and identical CAFR periods. We focus on the ith such CAFR period, and use this example to illustrate the different events

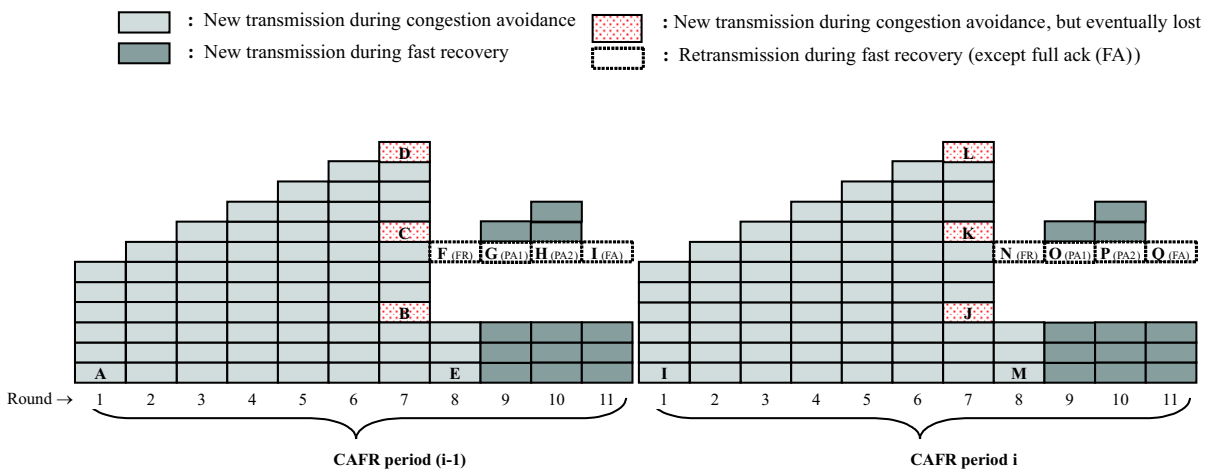


Fig. 2. Segment Transmissions in Two Adjacent and Identical CAFR Periods

in a CAFR period. Each CAFR consists of congestion avoidance and fast recovery. The first round of a CAFR period corresponds to the start of congestion avoidance (marked I in Figure 2). During congestion avoidance, the congestion window opens linearly, increasing by one (vertically) the number of segments transmitted per round. We note that the time gap between two horizontally adjacent rectangles in the same CAFR period, on average, equals the RTT. In round $W/2+1=7$ in Figure 2, three (non-contiguous) transmitted segments are lost. The first of these lost segments (marked J in Figure 2) is detected in the following round upon receipt of triple duplicate ACKs, resulting in termination of congestion avoidance and a fast retransmit (marked N in Figure 2). TCP then enters fast recovery.

We use the term *drop window* to refer to the window's worth of segments starting from the first lost segment in round $W/2+1$ to the segment transmitted just before the receipt of the first duplicate ACK. Suppose that m segments are lost in the drop window. As shown in Figure 2 (and Figure 3), fast recovery continues for m RTTs with TCP sending up to approximately $W/2$ new segments per RTT. TCP exits fast recovery and resumes normal congestion avoidance behaviour when a full ACK (FA) is received.

From our assumptions regarding statistically identical CAFR periods, we extrapolate and consider the case where two adjacent CAFR periods are exactly identical, as shown for example in Figure 2. From Figure 2 we see that S_{CAFR} can be expressed as the sum of: 1) the expected number of segments α transmitted between the end of one LE and the start of the next LE (e.g., between D and J in Figure 2); and 2) the expected number of segments δ transmitted between the first loss and the last loss (e.g., between J and L in Figure 2) of a loss event. It follows from the assumptions regarding loss events that the expected value of α is $1/p$ [29], [35]. Therefore,

$$S_{CAFR}=\frac{1}{p}+\delta. \tag{2}$$

Next, we derive δ. For m uniformly spaced drops in a window of size W, the expected number of segments transmitted between the first and the last loss in the same CAFR period (e.g., between J and L in Figure 2) is:

$$\delta=W-\frac{W}{E[m]}. \tag{3}$$

The expected value of m can be obtained as follows. Let $A(W,m)$ denote the probability of m segment losses from a drop window of size W. By definition, the first segment in the drop window is always lost. Furthermore, we have assumed that segments in the drop window are lost independently of other segments with probability q. Therefore,

$$A(W,m)=C_{m-1}^{W-1}\,(1-q)^{W-m}q^{m-1}. \tag{4}$$

Since we have assumed that all losses are identifiable by triple duplicate ACKs, we know that $m\leq W-3$. Hence[2],

$$E[m]=\sum_{m=1}^{W-3}mA(W,m)\approx 1+(W-1)q\approx 1+Wq. \tag{5}$$

Substituting $E[m]$ into Equation 3, we obtain:

$$\delta=\frac{W^2q}{1+Wq}. \tag{6}$$

Finally, substituting δ into Equation 2 we obtain:

$$S_{CAFR}=\frac{1}{p}+\frac{W^2q}{1+Wq}. \tag{7}$$

To compute W in terms of p and q, we need an alternate expression for S_{CAFR}. From Figure 2, note that S_{CAFR} can be expressed as the sum of: 1) the expected number of segments S_{LI} transmitted in the linear increase phase (from round 1 to round $W/2+1$); 2) the expected number of segments S_β transmitted from the start of round $W/2+2$ (marked M in Figure 2) until triple duplicate ACKs terminate congestion avoidance (N in Figure 2); and 3) the expected number of segments S_{FR} transmitted during fast recovery (from N to Q in Figure 2). Therefore,

$$S_{CAFR}=S_{LI}+S_\beta+S_{FR}. \tag{8}$$

[2]This approximation holds when q is small. All subsequent approximations also assume that q is small.

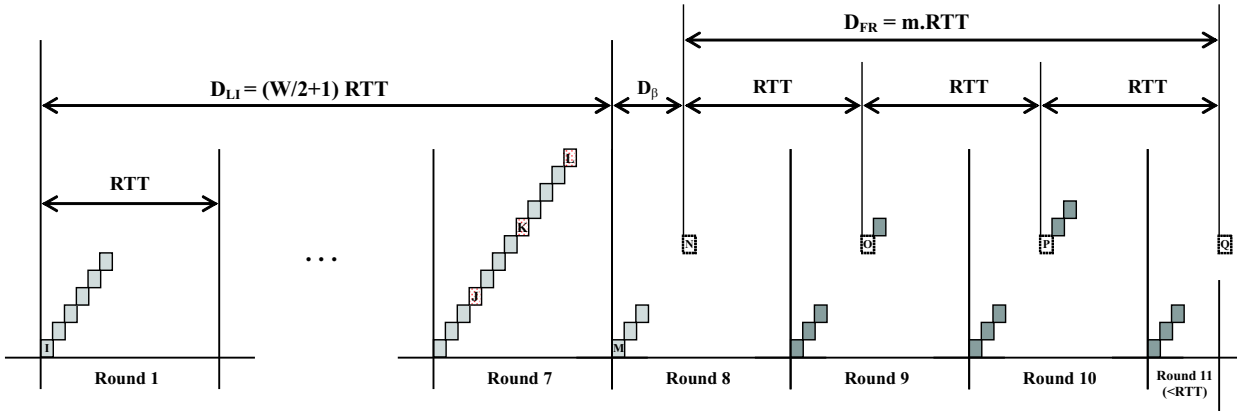


Fig. 3. Time View of a CAFR Period

We will determine S_{FR} first. The time view of a CAFR period shown in Figure 3 may be helpful in following the ensuing discussion. When TCP detects a segment loss and enters fast recovery, the expected number of outstanding segments is W. With m drops from the window, the source receives $W-m$ duplicate ACKs during the first RTT of fast recovery. Each duplicate ACK increases the congestion window by one segment, so at the end of the first RTT the congestion window size will be $\frac{3}{2}W-m$. This inflated congestion window allows TCP to send $\frac{W}{2}-m$ new segments during the first RTT of fast recovery, provided $m \leq \frac{W}{2}$. The second RTT starts with the reception of the first partial ACK (PA1). Immediately following the receipt of the partial ACK, TCP retransmits the next lost segment and also transmits one new segment. During this second RTT of fast recovery, $\frac{W}{2}-m$ additional duplicate ACKs will arrive, increasing the congestion window size by the same amount. This window increase allows the transmission of $\frac{W}{2}-m$ new segments as well. In total, TCP transmits $\frac{W}{2}-m+1$ segments in the second RTT. For m segment losses, fast recovery requires exactly m round-trip times to recover all the lost segments with TCP transmitting $\frac{W}{2}-m+j-1$ new segments in the jth RTT of fast recovery. Generalizing we obtain:

$$S_{FR}^{m\leq\frac{W}{2}} = \sum_{j=1}^{m}\left(\frac{W}{2}-m+j-1\right) = \frac{m}{2}\left(W-m-1\right). \quad (9)$$

If $m > \frac{W}{2}$, TCP will not transmit any new data during the first RTT of fast recovery, because the congestion window size $\frac{3}{2}W-m$ at this time is smaller than the amount of outstanding data W. With each partial ACK, the congestion window size increases by one segment. Thus, TCP needs $m-\frac{W}{2}$ partial ACKs to inflate the congestion window size to the number of outstanding segments W. Therefore, on arrival of the $(m-\frac{W}{2}+1)$th partial ACK, TCP can transmit one new segment. In the next RTT, TCP will transmit two new segments, and so on. In general:

$$S_{FR}^{m>\frac{W}{2}} = \sum_{k=m-\frac{W}{2}+1}^{m-1}\left(\frac{W}{2}-m+k\right) = \frac{W^2}{8}-\frac{W}{4}. \quad (10)$$

Using Equations 4, 9, and 10, the expected number of new segments transmitted during fast recovery is:

$$S_{FR} = \sum_{m=1}^{\frac{W}{2}} A(W,m)\ S_{FR}^{m\leq\frac{W}{2}} + \sum_{m=\frac{W}{2}+1}^{W-3} A(W,m)\ S_{FR}^{m>\frac{W}{2}}$$

$$\approx \frac{W^2}{2}\left(q-q^2\right) + \frac{W}{2}\left(1-5q+3q^2\right) - \left(1-2q+q^2\right). \quad (11)$$

We next determine S_{LI} for Equation 8. Immediately following receipt of a full ACK, fast recovery is terminated and the congestion window is reset to $W/2$ (e.g., I in Figure 2). This also ends the current cycle and normal congestion avoidance begins. In this phase, the congestion window increases by one segment per round until it reaches the assumed peak value of W in round $W/2+1$. It therefore follows that:

$$S_{LI} = \sum_{i=\frac{W}{2}}^{W} i = \frac{3}{8}W^2 + \frac{3}{4}W. \quad (12)$$

To determine S_β for Equation 8, we consider its two extreme boundary cases. If the first loss occurs at the start of round $W/2+1$, then the number of segments $S_\beta^{'}$ transmitted in the next round until termination of congestion avoidance is 0. Similarly, $S_\beta^{'} = W-1$ if the first loss occurs at the end of round $W/2+1$. Therefore, we approximate[3] S_β with its median value $W/2$.

Substituting the expressions for S_{LI}, S_{FR}, and S_β into Equation 8 and simplifying, we obtain:

$$S_{CAFR} =$$

$$\left(\tfrac{3}{8}+\tfrac{q}{2}-\tfrac{q^2}{2}\right)W^2 + \left(\tfrac{7}{4}-\tfrac{5q}{2}+\tfrac{3q^2}{2}\right)W - \left(1-2q+q^2\right) \quad (13)$$

Equating the right-hand sides of Equation 7 and Equation 13, and neglecting high-order terms, we can express the value of W in terms of p and q as:

$$W = \frac{10pq-5p+\sqrt{p(24+32q+49p)}}{p(3+4q)}. \quad (14)$$

[3]This approximation introduces a small amount of error into our model.

To obtain the expected time duration of a CAFR period, we again refer to the time view of a CAFR period, shown in Figure 3. From this illustration, we note that:

$$D_{CAFR} = D_{LI} + D_{\beta} + D_{FR}, \tag{15}$$

where D_{LI} is the expected duration of a linear increase period, D_{β} is the expected delay from the start of round (W/2+2) to the end of congestion avoidance, and D_{FR} is the expected duration of the fast recovery phase. The duration of the linear increase phase is:

$$D_{LI} = \left(\frac{W}{2} + 1\right) R. \tag{16}$$

For m segment losses in the drop window, fast recovery requires m round-trip times. Therefore,

$$D_{FR} = E[m]R \approx (1 + Wq)R \tag{17}$$

Using arguments similar to those used for determining S_{β}, we approximate using $D_{\beta} = \frac{R}{2}$. Substituting D_{LI}, D_{β}, and D_{FR} into Equation 15, we obtain:

$$D_{CAFR} = \left(\frac{W}{2} + Wq + \frac{5}{2}\right) R \tag{18}$$

Finally, substituting Equation 7 and Equation 18 into Equation 1, we obtain:

$$T_{NoTO} = \frac{\frac{1}{p} + \frac{W^2 q}{1+Wq}}{\left(\frac{W}{2} + Wq + \frac{5}{2}\right) R}, \tag{19}$$

where W can be computed from Equation 14.

B. Full Model (Full)

This section extends the foregoing model to include timeouts as loss indications. We refer to this as the "Full" model.

We again view the congestion window evolution as a concatenation of statistically identical cycles. Each cycle consists of several CAFR periods followed by a CATOSS period, where a CATOSS period is the concatenation of congestion avoidance (CA), timeout (TO), and slow start (SS) periods, as shown in Figure 4. Therefore, the throughput of a TCP NewReno flow can be expressed[4] as:

$$T_{Full} = \frac{(1 - p_{TO}) S_{CAFR} + p_{TO}(S_{CA} + S_{TO} + S_{SS})}{(1 - p_{TO}) D_{CAFR} + p_{TO}(D_{CA} + D_{TO} + D_{SS})} \tag{20}$$

where p_{TO} is the probability that a loss event leads to a timeout. S_X is the expected number of successful segment transmissions in a period of type X, and D_X is the expected duration of a period of type X. Obviously, D_{CA}=$D_{LI} + D_{\beta}$. Intuitively, S_{CA}=$S_{LI} + S_{\beta}$. However, we use $S_{CA} = S_{LI}$ instead, since TCP forgets outstanding data after timeout.

TCP NewReno may experience a timeout either from the congestion avoidance phase or from the fast recovery phase. The former transition occurs when TCP does not receive enough duplicate ACKs to trigger fast retransmit/fast recovery,

[4]This expression ignores the duration of an incomplete fast recovery phase, as well as any new segments transmitted therein.

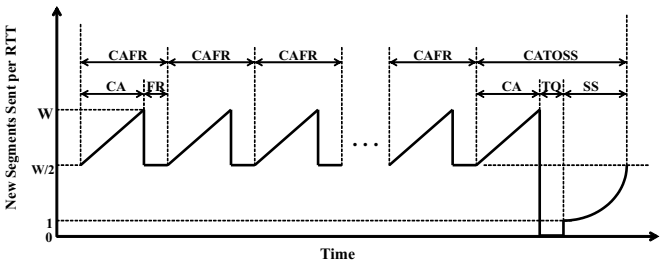


Fig. 4. Segment Transmissions in a Cycle (multiple CAFRs followed by CATOSS)

while the latter transition occurs when retransmitted segments are lost during the fast recovery phase. We express p_{TO} as:

$$p_{TO} = p_{DTO} + p_{IFR} \tag{21}$$

where p_{DTO} is the probability of directly transitioning to timeout from congestion avoidance and p_{IFR} is the probability of a timeout due to an unsuccessful fast recovery.

We determine p_{DTO} as follows. TCP experiences direct timeout when more than $W - 3$ segments are lost from a drop window of size W. Recalling the definition of $A(W, m)$ in Equation 4, we get:

$$p_{DTO} = \sum_{m=W-2}^{W} A(W, m). \tag{22}$$

When TCP NewReno loses no more than $W - 3$ segments from a drop window of size W, it enters fast recovery. On entering fast recovery, a timeout will occur if any segments retransmitted during fast recovery are lost. We approximate this condition by assuming that if a new loss event occurs during fast recovery, then the segment retransmitted in that RTT of fast recovery is also lost[5], thus triggering timeout. For m losses in the drop window, NewReno needs m round-trip times, sending approximately $W/2$ segments (including retransmissions) per RTT. The probability that the ith segment is lost given that the previous $i - 1$ segments arrived at the destination is $(1 - p)^{i-1}p$. Therefore, it follows from our assumptions that:

$$p_{IFR} = \sum_{m=1}^{W-3} A(W, m) \left[p + (1-p)p + \cdots + (1-p)^{\frac{mW}{2}-1} p\right]$$

$$= \sum_{m=1}^{W-3} A(W, m) \left[1 - (1-p)^{\frac{mW}{2}}\right]. \tag{23}$$

Substituting Equations 22 and 23 in Equation 21, we get:

$$p_{TO} = 1 - \sum_{m=1}^{W-3} A(W, m) \left[(1-p)^{\frac{mW}{2}}\right]. \tag{24}$$

Derivation of the expected duration of timeout is similar to [29]. Furthermore, during timeout TCP does not transmit

[5]While we do not explicitly model successive occurrences of FR, this assumption implicitly captures its effect by increasing the probability of timeout.

any new segments. Thus,

$$S_{TO} = 0 \quad \text{and} \tag{25}$$

$$D_{TO} = RTO\frac{1+p+2p^2+4p^3+8p^4+16p^5+32p^6}{1-p}. \tag{26}$$

In the slow start phase, the initial window size is 1 and the window size is doubled every round until the slow start threshold ($W/2$) is reached. In the last round of slow start, TCP transmits W/2 segments and enters congestion avoidance. We count the duration and segments of the last round of slow start as being part of congestion avoidance. Hence,

$$S_{SS} = 1 + 2 + 4 + \cdots + \frac{W}{4} = 2^{1+log\frac{W}{4}} - 1 \quad \text{and} \tag{27}$$

$$D_{SS} = \left(log\frac{W}{4} + 1\right) R. \tag{28}$$

Statistically, the numerator of Equation 20 can be replaced by $\frac{1}{p} + \frac{W^2q}{1+Wq}$ [35]. Substituting Equations 18, 26, 28, and D_{CA} into Equation 20, we obtain:

$$T_{Full} = \frac{\frac{1}{p}+\frac{W^2q}{1+Wq}}{N.R+p_{TO}\left((1+2p+4p^2)RTO+\left(1+log\frac{W}{4}\right)R\right)}, \tag{29}$$

where $N = \left(\frac{W}{2} + \frac{3}{2} + (1-p_{TO})(1+Wq)\right)$, and W can be computed from Equation 14.

IV. Model Validation

This section validates the proposed NewReno throughput model using the *ns-2* network simulator [1]. The results reported in this section also illustrate the performance advantages of NewReno over Reno. Finally, we quantify the ineffectiveness of existing TCP Reno models in predicting TCP NewReno throughput.

A. Network Model and Traffic Models

Before discussing the simulation results, we present the basic setup used in the ns-2 simulations. Specifically, we describe the network model and the various traffic models used. To conserve space when presenting the results, we describe only the setup changes with respect to the default settings discussed here.

Most of the results reported here are for a simple dumbbell network topology with a single common bottleneck[6] 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. We refer to the flows that are being actively monitored as the "foreground" flows, with all other traffic being designated as "background" flows.

All experiments have two foreground flows: one NewReno flow and one Reno flow. The receive buffers for the foreground flows are sufficiently provisioned such that their buffer space advertisements do not limit the congestion window size. The experiments vary the bottleneck bandwidths (e.g., 5 Mbps to 45 Mbps), the round-trip delays of the flows (e.g., 20 ms to 460 ms), the bottleneck queue management policies (e.g., DropTail and RED), and the load/mix of background traffic (e.g., mix of long duration FTP transfers, short duration HTTP sessions, and constant bit rate UDP flows). For RED queue management, the $minthresh$ and the $maxthresh$ are set to $1/3$ and $2/3$ of the corresponding queue size limit.

The background HTTP sessions are simulated using a model similar to that in [24], [35]. 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 HTTP and FTP sessions use TCP NewReno 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 these background flows are uniformly distributed between 20 ms and 460 ms, consistent with measurements reported in the literature [2], [20].

The background UDP flows are constant bit rate UDP flows with rate 1 Mbps each. The packet size is 1 KB and the one-way propagation delay for each UDP flow is 35 ms.

The results reported here are for the "Full" TCP NewReno model with q set to p, unless stated otherwise. The widely used TCP throughput model by Padhye *et al.* [29], henceforth referred to as the PFTK model, is used as a representative TCP Reno throughput model. From the simulations, the necessary input parameters for the analytical models are obtained. All the losses in a single window of data are counted as one loss event. The loss event rate p is taken to be the ratio of the total number of loss events to the total number of segment transmissions, in the period of interest. The average round-trip time R was measured at the sender, and RTO was approximated as $4R$.

In simulations where multiple long duration flows share a single bottleneck link, systematic discrimination has been observed against some connections [14], [15]. Such *phase* effects, however, rarely arise in experiments that consider a mix of long and short duration flows, with heterogeneous round-trip propagation delays [15]. As a precautionary measure, the experiments reported here start all flows at slightly different times. The background flows start at uniformly distributed times between 0 and 2 seconds, and the foreground flows start at uniformly distributed times between 5 and 7 seconds, all measured in simulation time since the start of a run. Each experiment simulates 1000 seconds of run. Results are reported using data from the last 750 simulated seconds.

B. Bernoulli Packet Loss

Before validating the model with background traffic, validation is carried out in isolation. The configuration considered here consists of two foreground flows traversing a 20 Mbps bottleneck link. A Bernoulli packet drop module was placed on the access link of each foreground flow. The bottleneck

[6]Section IV-F considers a scenario with multiple bottlenecks.

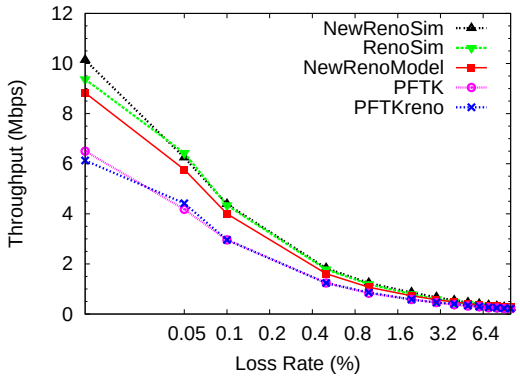


Fig. 5. Model Accuracy with Bernoulli Packet Loss

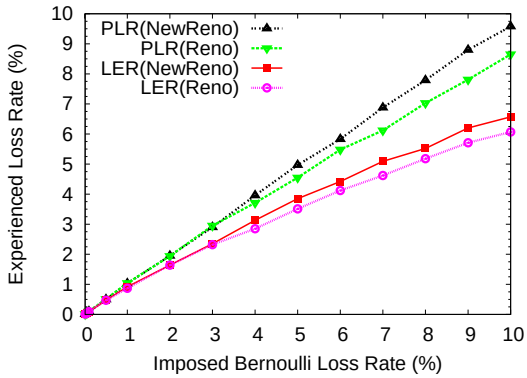


Fig. 6. Packet Loss Rate (PLR) and Loss Event Rate (LER) for the Imposed Bernoulli Loss Rate

router's buffer was sufficiently provisioned such that there were no congestion-induced packet losses. Experiments varied the drop probability of the loss module from 0.01% to 10%.

Figure 5 shows the throughput from the simulations and the models from representative experiments with round-trip propagation delay of the foreground flows set to 75 ms. For the imposed Bernoulli loss rates, the corresponding observed packet (segment) loss rates (PLR) and loss event rates (LER) for both foreground flows are shown in Figure 6.

Several observations can be drawn from the results in Figure 5. The results show that the proposed NewReno throughput model (`NewRenoModel` in the figures) is able to track accurately the simulation throughput over the entire range of loss rates considered. The prediction error of our model, defined as $|simulation - model|/simulation$, ranges from 0.00% to 15.12% with an average error of 8.97%. Furthermore, we note that if the PFTK model (`PFTK` in the figure) is naively used to estimate NewReno throughput (based on the packet loss ratio experienced by the foreground NewReno flow), the relative errors range from 16% to 35.90%.

The PFTK model, when applied to TCP Reno throughput (`PFTKreno` in the figures, based on the observed packet loss ratio for the foreground Reno flow), is also less accurate in this scenario, with prediction errors in the 4.76% to 34.58% range. The higher prediction errors in the PFTK model can be attributed to the omission of the fast recovery algorithm from their model, and the (correlated) bursty packet loss

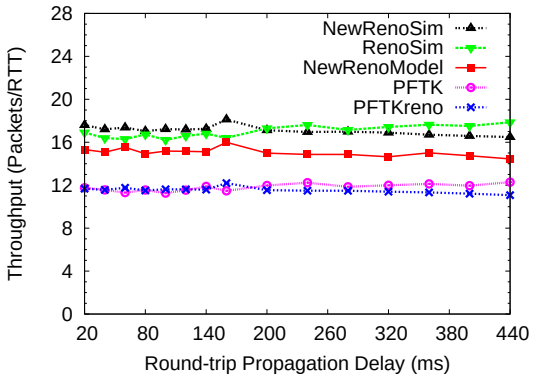


Fig. 7. Model Accuracy with Varying RTT

assumptions of the model. Note that with the Bernoulli packet drop module, most packet losses are isolated single packet drops that can be recovered using a single fast recovery phase. In fact, we observe that for low packet loss rates (e.g., less than 5%) most losses are single isolated drops, and thus the throughput of Reno and NewReno are similar in these cases.

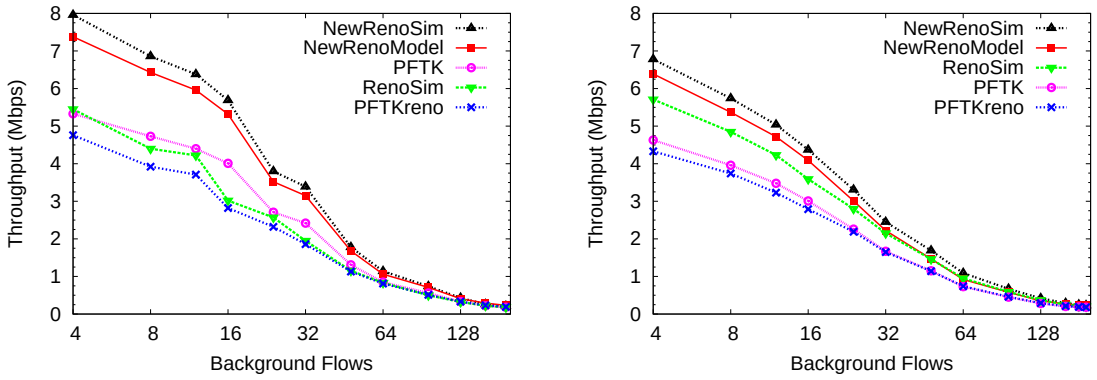
Figure 7 demonstrates the prediction accuracy of the models with the round-trip propagation delay of the foreground flows varied from 20 ms to 440 ms. The simulation setup is similar to the previous experiment except that the bottleneck bandwidth was set to 15 Mbps and the Bernoulli packet drop rate was fixed at 0.5%. For a fixed Bernoulli packet drop rate, the throughput per RTT (expressed as packets/RTT) is nearly constant. The results show that the NewReno model closely follows the simulation throughput, with an average prediction error of 12.02%. We observe that the PFTK model, as expected, is unable to track NewReno throughput and the average prediction error was 32.58%. We also note that the PFTK model predicts Reno throughput rather poorly in these experiments, with an average prediction error of 30.62%.

C. HTTP/FTP Background Traffic

The simulation results reported in this section are for a 15 Mbps bottleneck link with a queue of capacity 150 packets. The background traffic consists of a mix of 75% HTTP and 25% FTP flows. The total number of background flows is varied from 4 to 200. Both foreground flows have a round-trip propagation delay of 50 ms.

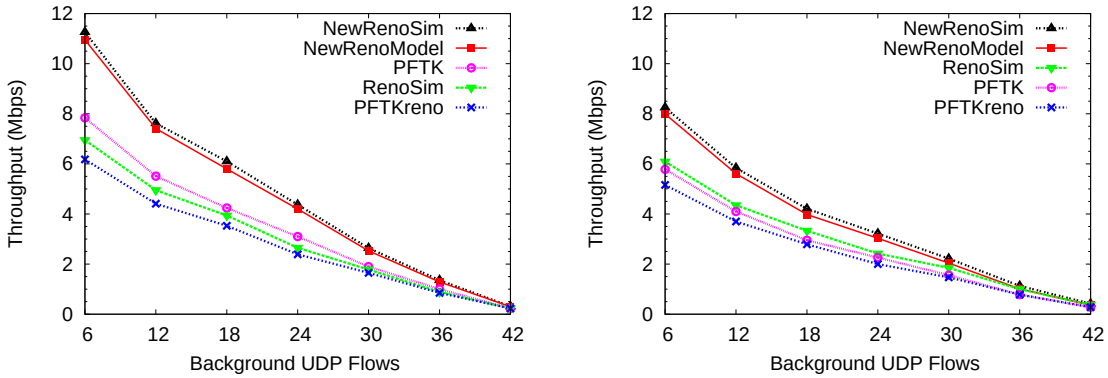
Figure 8 shows the simulated throughputs of NewReno and Reno as well as the throughputs from the analytic models. Figure 8(a) is for a DropTail bottleneck router, while Figure 8(b) is for a RED bottleneck router. Comparing the results in Figure 8(a) to the results for Bernoulli packet losses, we observe that NewReno throughput can be up to 50% higher than that of Reno. This is because the cross traffic generates bursty packet losses at the DropTail router buffer. NewReno is able to recover efficiently from these losses using its improved fast recovery algorithm. The performance differences between Reno and NewReno decrease when RED queues are used, as can be seen in Figure 8.

From Figure 8, we also note that the proposed analytic model tracks the simulated throughput for the range of back-



(a) DropTail (b) RED

Fig. 8. Model Accuracy with Background HTTP/FTP Traffic



(a) DropTail (b) RED

Fig. 9. Model Accuracy with Background UDP Traffic

ground traffic considered. The prediction error of our analytic model ranges from 4.05% to 11.54% in the case of experiments with DropTail queues, and ranges from 0% to 16.7% in the case of RED queue management policy. The foreground NewReno flow experienced packet losses in 0.06% to 11.0% range (corresponding to loss event rate between 0.02% and 4.72%) in the DropTail experiments, and packet losses in the 0.07% to 9% range (corresponding to loss event rates between 0.05% and 6%) in the RED experiments. The results also show that the PFTK model significantly underestimates NewReno throughput. The prediction errors in this case range from 14.29% to 33.04% with DropTail queues and 21.74% to 31.71% with RED queues.

Our results also show that the PFTK model more accurately predicts TCP Reno throughput when the bottleneck queue management policy is DropTail. The average prediction error is 6.33% in the DropTail experiments and 20.97% in the RED experiments. With RED queues, the burstiness of packet losses decrease allowing some of the packet losses to be recovered by the Reno fast recovery algorithm. As stated earlier, the PFTK model assumes that packet losses are bursty (i.e., losses within a round are correlated) and also does not model fast recovery.

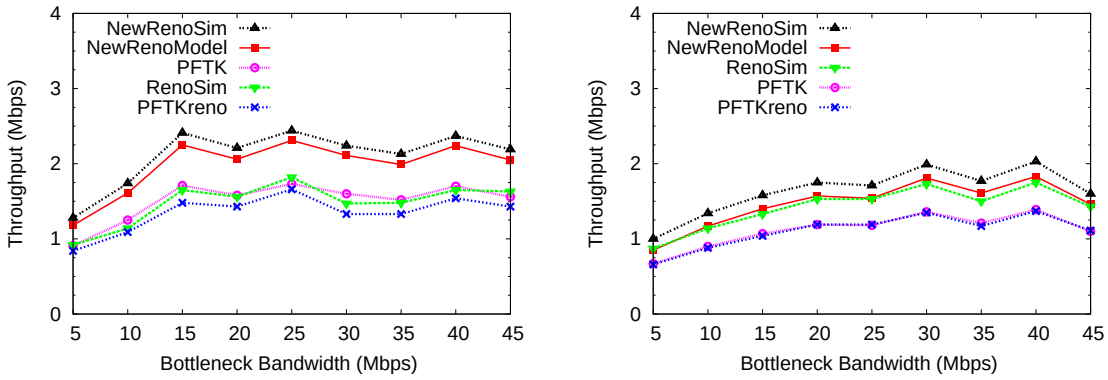
Our results indicate that modelling the Reno fast recovery phase is important in these scenarios. Furthermore, our two-parameter loss model provides robust results for both DropTail and RED packet loss scenarios.

D. UDP Background Traffic

This section considers the impact of background traffic that is predominantly constant bit rate UDP. The experiments reported here are for a 45 Mbps bottleneck link with a queue limit of 250 packets. The background traffic consists of a fixed number of HTTP/FTP background flows (24 HTTP sessions and 8 FTP sessions) and a varying number of constant bit rate UDP flows. The two foreground flows, namely NewReno and Reno, each have a round-trip propagation delay of 70 ms.

The results in Figure 9 again show that TCP NewReno can significantly outperform TCP Reno under similar network conditions. We also observe that the proposed analytic model closely tracks NewReno throughput for both RED (average prediction error is 7.35%) and DropTail (average prediction error is 3.84%) experiments. Similar to the results reported in the earlier sections, the PFTK model is unable to track NewReno throughput; the prediction errors range from 25.81% to 30.49% for DropTail, and from 29.79% to 33.33% for RED.

By comparing the results in Figure 8(b) and Figure 9(b), it can be observed that the PFTK model tracks TCP Reno throughput reasonably well for RED queues when constant bit rate UDP flows are the major contributor to background load. This might be expected since the uncontrolled UDP flows cause bursty losses at the bottleneck RED queues, making the loss behaviour similar to that of a DropTail queue.



(a) DropTail (b) RED

Fig. 10. Model Accuracy with System Scaling

E. System Scaling

The next experiment studies the robustness of our model to the scaling of network model and workload parameters.

Figure 10 shows the simulated throughput of the foreground flows and the results from the analytic models for a range of bottleneck bandwidths. Here, the initial experimental setup had a 5 Mbps bottleneck link with a buffer of 50 packets and 20 background flows. The background flows consist of 10% FTP flows and 90% HTTP sessions. At each step, all system resources and the background loads are scaled upwards. Thus, for each new configuration, the bottleneck capacity is increased by 5 Mbps, the queue size by 50 packets, the number of HTTP sessions by 18, and the number of background FTP sessions by 2. The foreground NewReno and Reno flows each have a round-trip propagation delay of 70 ms.

The results show that NewReno throughput is 30% to 50% higher than Reno throughput under identical network conditions. It can be observed that our NewReno analytic model accurately tracks the throughput observed in the simulations for a wide range of bandwidths, with both DropTail (average prediction error is 6.48%) and RED queue management policies (average prediction error is 10.67%). Similar to observations made in earlier sections, predicting NewReno throughput with the PFTK throughput model has higher prediction errors: an average prediction error of 28.66% for the DropTail experiments, and 31.91% for the RED experiments. Furthermore, the results reiterate that the PFTK analytic model is more accurate for Reno if DropTail queues are in use (average estimation error 8.79%). However, when RED queue management is employed at the bottleneck buffer, the PFTK model is less accurate, with prediction errors of 21.71% to 24.14%.

F. Multiple Bottlenecks

This section reports validation results from an experiment setup with multiple bottlenecks. The network topology used here consists of two dumbbell networks connected in series at the bottlenecks. Each bottleneck link had a capacity of 15 Mbps with a buffer space for 125 packets. Identical background load was applied to each bottleneck link (i.e., each background flow traverses only a single bottleneck). Each background traffic mix consists of 75% HTTP flows and 25% FTP flows. We varied the total number of flows from 4 to 200.

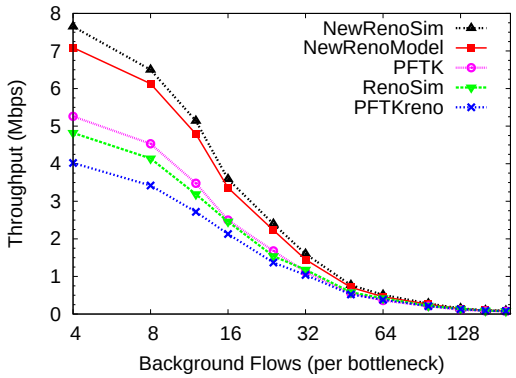
Figure 11 shows the throughput from the simulations and the results from the analytic models. In the DropTail experiments, the foreground NewReno flow experiences between 0.05% and 15.7% packet losses, with corresponding loss event rates ranging between 0.01% and 8.0%. As shown in Figure 11, our NewReno throughput model closely tracks the simulation throughput over the entire range of background load simulated. The prediction errors in these experiments range between 0.00% and 9.80%.

Compared to the DropTail experiment results, the results from the experiments with RED queues show somewhat higher prediction errors. The prediction errors in this setting range between 0.00% and 16.67% with an average prediction error of 10.02%. We note that the NewReno flow experienced slightly higher packet loss in the RED experiments, with packet loss rates ranging from 0.05% to 17.5%, and the corresponding loss event rate ranging from 0.03% to 11.2%. Similar to earlier results, we observe that the prediction errors increase significantly if the PFTK model is used to estimate NewReno throughput. In the multiple bottleneck experiments, the average prediction error of the PFTK model (when tracking NewReno throughput) is approximately 28% under both RED and DropTail queue management policies.

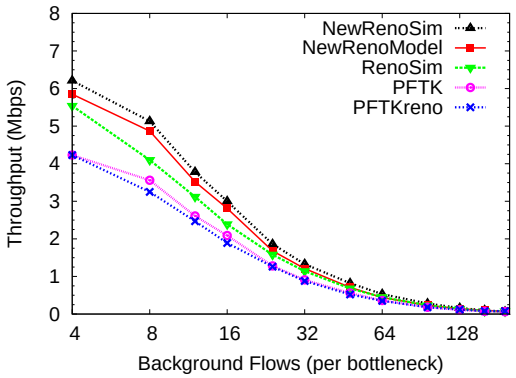
G. Impact of Timeouts

This section presents sample results that illustrate the differences between the "NoTO" model and the "Full" model. In general, we observed that the NoTO model is adequate when there is little contention for the bottleneck resource, whereas the Full model is more applicable otherwise.

In Figure 12(a), an expanded view of Figure 5 is presented, highlighting model performance for Bernoulli packet losses in the 6% to 10% range. At a high Bernoulli loss rate, TCP's retransmit timeout mechanism dominates and the NoTO Model, as expected, is somewhat inaccurate. The average prediction error of the NoTO model for this range of imposed packet loss rates is 15.26%. As the results show, the Full model follows the throughput much more accurately. The average prediction error of the Full model in this range is only 4.11%.

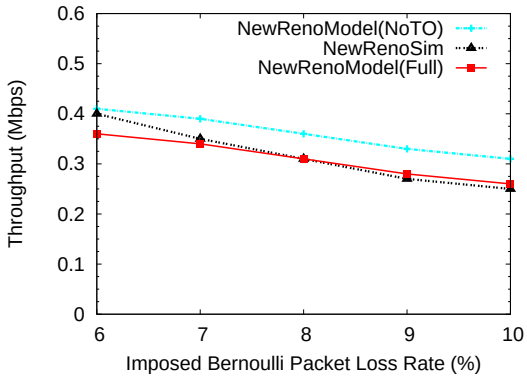


(a) DropTail

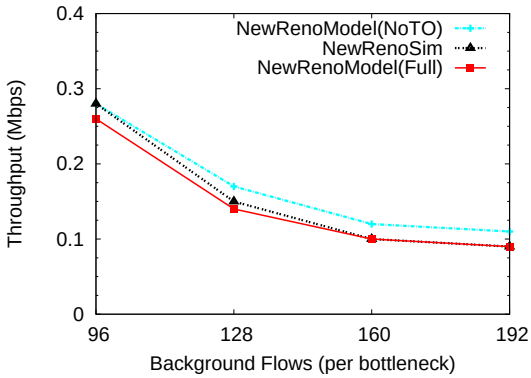


(b) RED

Fig. 11. Model Accuracy with Multiple Bottlenecks



(a) Bernoulli Packet Loss



(b) Multiple Bottleneck (DropTail)

Fig. 12. Comparison of the NoTO NewReno Model and the Full NewReno Model

The NoTO model is comparatively more accurate than the Full model for Bernoulli packet loss rates in the range of 0.01% to 5%. For this range of loss rates, the average prediction error is 7.64% with the NoTO model and 11.67% with the Full model.

Figure 12(b) presents an expanded view of Figure 11(a), focusing on 96 to 192 background flows. In this range, the packet loss rate experienced by the foreground NewReno flow ranges from 4.7% to 15.7%, with corresponding loss event rates ranging between 2.3% and 7.8%. The Full model tracks the simulated throughput reasonably well with an average prediction error of 3.45%. The performance of the NoTO model is comparatively poorer and its average prediction error is 13.89%. Similar to Bernoulli packet drop experiments, we observe that the NoTO model predicts NewReno throughput more accurately when there is less contention at the bottleneck link. For example, with 4 to 96 background flows, the average prediction error of the NoTO model is less than 6%.

H. Impact of Bursty Packet Losses

This section illustrates the flexibility of our novel two-parameter loss model. We first develop an approximation for computing q from the measured characteristics of the flow. Then we use simple simulations to quantify the inaccuracies that may result from assuming $q = p$ in the presence of bursty packet losses, and note conditions where the aforementioned assumption is reasonable.

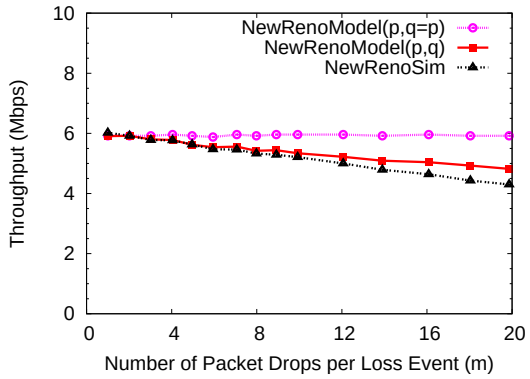


Fig. 13. Model Accuracy with Bursty Losses (Constant $p = 0.05\%$)

Given the average loss rate $\tilde{q}$ observed over the entire duration of the transfer, and the loss event rate p, a relation between $\tilde{q}$, q, p, and W can be obtained as follows. The expected number of segment losses per loss event is $m = \tilde{q}/p$. Using Equation 5, we obtain:

$$q \approx \frac{\tilde{q}/p - 1}{W - 1}, \tag{30}$$

where W is computed from Equation 14 using $q = \tilde{q}$.

The simulation results reported here are for a single foreground NewReno flow traversing a 45 Mbps bottleneck link.

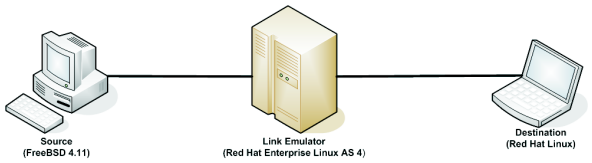


Fig. 14. Testbed for Emulation Experiments

No background flows are present, and the round-trip propagation delay of the NewReno flow is 70 ms. A specialized drop module that takes as input two parameters p and m was placed on the access link of the TCP Sink node. This drop module schedules Bernoulli loss events at rate p; whenever a loss event occurs, m back-to-back packets are dropped.

Figure 13 shows the simulated throughput of the NewReno flow and the results from the analytic model. In the experiments, m was varied from 1 to 20 while keeping the loss event rate fixed at 0.05%. We ensured that the TCP congestion window was large enough to allow m back-to-back packet drops per loss event. Analytic results are shown for $q = p$, and q approximated using Equation 30. When the approximation $q = p$ is used, the average prediction error is 17.35%. Determining q using Equation 30, however, reduces the average prediction error to 5.00%.

The results again demonstrate the accuracy of our analytic model, and show that the two-parameter loss model is particularly useful in scenarios that involve bursty packet losses. In a separate paper [31], we use the q parameter (and a fixed loss event rate p) to study the effect of bursty packet losses on two variants of NewReno, namely Slow-but-Steady (SBS) and Impatient (IMP). Contrary to RFC 3782, we find that the SBS variant offers superior throughput to IMP in all but the most extreme packet loss scenarios (e.g., 26 or more segment losses per window [31]). Similar experiments (not shown here) clearly demonstrate the superiority of partial window deflation versus full window deflation in TCP NewReno.

V. Emulation Experiments

We validated our TCP NewReno throughput model using a real TCP source and a real TCP sink on an emulated wide area network in our laboratory. This section describes the experimental testbed used for emulation, the emulated network configuration, and the experimental results.

A. Testbed Configuration

The testbed consists of three physical machines on a 100 Mbps private Ethernet LAN, as shown in Figure 14. One machine serves as the TCP source node, with another as the TCP destination node, and the third as the network emulator.

The TCP source node was a 1.8 GHz Intel Pentium 4 machine with 512 MB of RAM, running the FreeBSD 4.11 operating system. We verified that the NewReno implementation in the FreeBSD kernel conformed to the TCP NewReno description in RFC 3782. In addition, we instrumented the FreeBSD kernel to collect statistics required for model validation such as the number of timeout (TO) events, the number of fast recovery (FR) events, the total transfer duration (in seconds), the total bytes successfully transferred (Bytes), and the fine-grained RTT. The TCP destination node was a 2.8 GHz Intel Xeon with 1 GB of RAM. This machine was running Linux 2.6.8 as the operating system.

TABLE II
Summary of Emulation Experiments

Loss Rate	TO	FR	RTT (ms)	Duration (sec.)	Bytes
0.50%	5	556	55.56	500.54	169,942,625
1.00%	21	765	57.31	502.33	116,768,593
1.50%	38	841	58.48	502.62	89,441,537
2.00%	68	864	59.06	501.92	72,544,825
2.50%	113	775	59.65	502.11	55,202,129
3.00%	129	777	60.23	502.92	47,554,586

We used `iperf`[7] for generating TCP bulk data transfers. This software, freely available from NLANR, is used for measuring TCP and UDP performance. In our experiments, we ran `iperf` in the TCP-mode to generate traffic representing bulk data transfer.

We used the Internet Protocol and Traffic Network Emulator (IP-TNE) [38] to emulate a wide area network. IP-TNE is a high-performance internetwork emulation tool that uses a parallel discrete-event simulation kernel. In our experiments, all `iperf` traffic between the TCP source and TCP destination traverses the virtual (simulated) wide area network. IP-TNE transfers IP packets as needed between the real and the simulated network, and models packet transmissions in the emulated wide area network. IP-TNE was running on a 3.2 GHz Intel Xeon machine with 4 GB of RAM; the operating system on this machine was Red Hat Enterprise Linux Academic Server Edition 4.

B. Emulated Network

The experiments reported here use a simple dumbbell network topology. There is a single bottleneck link of capacity 10 Mbps between the TCP source node and the TCP sink node. The TCP source and destination nodes are each connected to the bottleneck link by a 100 Mbps access link. In the experiments, the round-trip propagation delay of the emulated network is 50 ms.

All routers in the emulated network use FIFO queueing, with DropTail queue management. We installed a Bernoulli packet loss module on the access link of the TCP destination node to drop packets at a predetermined rate. The buffer at the bottleneck router was sufficiently provisioned such that there were no congestion-induced packet losses. This setup is simple, but allows us to compare the emulation results with those from *ns-2* simulations.

C. Results

In our emulation experiments, we varied the packet drop rate from 0.5% to 3%, in steps of 0.5%. Table II summarizes statistics obtained from the emulation experiments. Note that summing the number of FR and TO events represents the total number of loss events experienced by the TCP flow.

[7] http://dast.nlanr.net/Projects/Iperf/

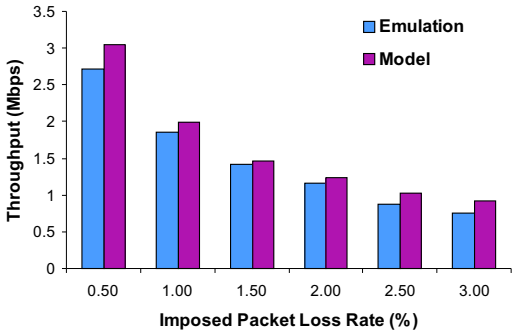


Fig. 15. Model Accuracy in WAN Emulation

The segment size (or *Segsize*) for all transfers is 1448 byte excluding TCP and IP headers. As in [29], we use the expression $\frac{FR+TO}{Bytes/Segsize}$ to estimate the loss event rate p. This computed loss event rate and the measured RTT are used as inputs to our "Full" TCP NewReno throughput model. In our computation of the model estimated throughputs, we used the approximation $q = p$.

In the absence of loss (p=0.00%), the NewReno flow fully utilizes the 10 Mbps bottleneck link. The achieved throughput is 9.59 Mbps excluding TCP/IP header overhead, and 9.93 Mbps including TCP/IP overhead.

Figure 15 shows the (emulated) throughput attained by the TCP flow, along with the throughput predicted by our model. All these throughput calculations exclude the TCP/IP header overhead. At 1.5% imposed loss rate, the emulation throughput is 1.42 Mbps and the model prediction is 1.46 Mbps, correspondingly the prediction error is 3.04%. The maximum estimation error observed is 21.26% (at an imposed packet loss rate of 3.0%), and the average prediction error is 11.29%.

In general, our model predicts the TCP NewReno throughput successfully in the experiments considered.

VI. Internet Experiments

As a final step for model validation, we conducted several experiments on the Internet. With help from selected colleagues around the globe, we measured the throughput achieved for 5 MB file transfers from our BSD Unix server site in North America to 6 different client locations: USA, Canada, UK, Australia, Bangladesh, and Japan. For space reasons, we only present results from the latter experiment, which had the worst-case prediction error observed.

To validate our model predictions at different loss rates, we added controlled levels of packet loss to our experiments using DummyNet [34]. We varied the imposed packet loss rate (PLR) from 0.50% to 3.00%, leaving bandwidth and delay unchanged. Actual losses always exceed the imposed PLR.

Table III shows the results from the Japan experiment. The (Full) NewReno model predicts the observed throughputs reasonably well, with an average prediction error of 11.96%. These model predictions use the assumption $q = p$. The native network path (i.e., with zero imposed PLR) is lossy, experiencing a loss event rate (LER) of 0.98%, and a PLR of 6.21%. The prediction error for this case is high at 31.29%, because the average number of segment losses per loss event ($m = \frac{6.21}{0.98} = 6.4$) is relatively large, and the assumption $q = p$ is violated. Using $q = \frac{m-1}{W-1}$ in the model (denoted with '*' in Table III) reduces the prediction error to 0.94%.

TABLE III
Experimental Results from Calgary to Japan

Imposed PLR	TO	FR	Actual LER	RTT (ms)	Duration (sec)	Expt (Kbps)	Model (Kbps)
0.00%	7	27	0.98%	229	106.34	379	497 382*
0.50%	5	39	1.27%	245	120.12	335	394
1.00%	9	58	1.93%	237	134.84	299	297
1.50%	24	87	3.19%	253	195.73	206	202
2.50%	59	100	4.57%	256	277.23	145	147
2.00%	26	76	2.93%	300	302.13	133	173
3.00%	59	119	5.12%	260	290.15	139	141

VII. Conclusions

This paper presents an analytic model for the bulk data transfer performance of TCP NewReno. The model expresses steady-state throughput in terms of RTT and loss rate.

Our NewReno throughput model has three important features. First, we explicitly model the fast recovery algorithm of TCP NewReno, which is important since a NewReno flow may spend a significant amount of time in the fast recovery phase. Second, we also consider the possibility of incurring a timeout following an unsuccessful fast recovery phase. Third, our analytical model uses a flexible two-parameter loss model that better captures the dynamics of TCP loss events on the Internet.

We validated our model with extensive *ns-2* simulation experiments. We also validated our model using a real TCP NewReno implementation. Our results show that the proposed model can predict steady-state TCP NewReno throughput for a wide range of network conditions, unlike existing Reno models. The results also illustrate the performance advantages of NewReno over Reno because of NewReno's improved fast recovery algorithm.

Several avenues remain for future work. These include the development and validation of models for other TCP variants and high delay-bandwidth product networks, modeling TCP's transient state for short-duration TCP transfers, and the application of TCP modeling to peer-to-peer networks.

Acknowledgements

The authors thank Derek Eager for his feedback on the model, Sally Floyd for her comments on different aspects of TCP NewReno, Martin Arlitt for his constructive comments on an earlier version of this paper, and Sean Boyden for his help with the emulation experiments. We also thank the TeleSim Research Group for making IP-TNE available to us.

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] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *Proc. of IEEE INFOCOM*, pages 631–640, Anchorage, USA, April 2001.
[6] 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.
[7] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proc. of IEEE INFOCOM*, pages 1742–1751, Tel-Aviv, Israel, March 2000.
[8] K. Fall and S. Floyd. Simulation-Based Comparisons of Tahoe, Reno, and Sack TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.
[9] V. Firoiu and M. Borden. A Study of Active Queue Management for Congestion Control. In *Proc. of IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.
[10] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks. *ACM Computer Communication Review*, 21(5):30–47, 1997.
[11] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.
[12] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. of ACM SIGCOMM*, pages 43–56, Stockholm, Sweden, August 2000.
[13] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, April 2004.
[14] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.
[15] S. Floyd and E. Kohler. Internet Research Needs Better Models. In *Proc. of First Workshop on Hot Topics in Networking*, Princeton, USA, October 2002.
[16] M. Goyal, R. Guerin, and R. Rajan. Predicting TCP Throughput from Non-Invasive Network Sampling. In *Proc. of IEEE INFOCOM*, Hiroshima, Japan, March 2002.
[17] Q. He, C. Dovrolis, and M. Ammar. On the Predictability of Large Transfer TCP Throughput. In *Proc. of ACM SIGCOMM*, Philadelphia, USA, August 2005.
[18] V. Jacobson. Congestion Avoidance and Control. In *Proc. of ACM SIGCOMM*, pages 314–329, Stanford, CA, USA, August 1988.
[19] 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.
[20] H. Jiang and C. Dovorolis. Passive Estimation of TCP Round-trip Times. *ACM Computer Communication Review*, 32(3):75–88, July 2002.
[21] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination using an Overlay Mesh. In *Proc. of ACM SOSP*, Bolton Landing, USA, October 2003.
[22] A. Kumar. Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link. *IEEE/ACM Transactions on Networking*, 6(4):485–498, August 1998.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] J. Padhye, V. Firioiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. of ACM SIGCOMM*, Vancouver, Canada, September 1998.
[30] J. Padhye and S. Floyd. On Inferring TCP Behavior. In *Proc. of ACM SIGCOMM*, pages 287–298, San Deigo, USA, August 2001.
[31] N. Parvez, A. Mahanti, and C. Williamson. TCP NewReno: Slow-but-Steady or Impatient? In *Proceedings of IEEE ICC 2006*, Istanbul, Turkey, June 2006.
[32] V. Paxson. Empirically Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Trans. on Networking*, 2(4):316–336, 1994.
[33] J. Postel. Transmission Control Protocol. RFC 793, September 1980.
[34] L. Rizzo. Dummynet and Forward Error Correction. In *Proc. of Freenix*, New Orleans, USA, June 1998.
[35] C. Samios and M. Vernon. Modeling the Throughput of TCP Vegas. In *Proc. of ACM SIGMETRICS*, San Diego, USA, June 2003.
[36] 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.
[37] B. Sikdar, S. Kalyanaraman, and K. Vastola. Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK. *IEEE/ACM Transactions on Networking*, 11(6):959–971, December 2003.
[38] R. Simmonds, R. Bradford, and B. Unger. Applying Parallel Discrete Event Simulation to Network Emulation. In *Proc. ACM Parallel and Distributed Simulation*, pages 15–22, Bologna, Italy, May 2000.
[39] W. Stevens. *TCP/IP Illustrated Vol. 1: The Protocols*. Addison-Wesley, Boston, USA, 1994.
[40] W. Stevens and G. Wright. *TCP/IP Illustrated Vol. 2: The Implementation*. Addison-Wesley, Boston, USA, 1995.

Nadim Parvez is a Ph.D. candidate in the Department of Computer Science at the University of Calgary. He holds a B.E. in Computer Science and Engineering from the Bangladesh University of Engineering Technology and a M.Sc. in Electrical and Computer Engineering from the University of Manitoba. His research interests include Internet protocols, TCP modelling, peer-to-peer systems, and wireless sensor networks.

Anirban Mahanti is an Assistant Professor in the Department of Computer Science at the University of Calgary. He holds a B.E. in Computer Science and Engineering from the Birla Institute of Technology (at Mesra), India, and a M.Sc. and a Ph.D. in Computer Science from the University of Saskatchewan. His research interests include Internet protocols, P2P applications, multicast, and scalable multimedia streaming.

Carey Williamson is an iCORE Chair in the Department of Computer Science at the University of Calgary, specializing in *Wireless Internet Traffic Modeling*. He holds a B.Sc.(Honours) in Computer Science from the University of Saskatchewan, and a Ph.D. in Computer Science from Stanford University. His research interests include Internet protocols, wireless networks, network traffic measurement, network simulation, and Web performance.