Analytic Models for the Latency and Steady-State Throughput of TCP Tahoe, Reno and SACK

B. Sikdar, S. Kalyanaraman and K. S. Vastola
Dept. of ECSE, Rensselaer Polytechnic Institute
Troy, NY 12180 USA
email: {bsikdar,shivkuma,vastola}@networks.ecse.rpi.edu

Abstract—**Continuing the process of improvements made to TCP through the addition of new algorithms in Tahoe and Reno, TCP SACK aims to provide robustness to TCP in the presence of multiple losses from the same window. In this paper we present analytic models to estimate the latency and steady-state throughput to TCP Tahoe, Reno and SACK and validate our models using both simulations and TCP traces collected from the Internet. In addition to being the first models for the latency of finite Tahoe and SACK flows, our model for the latency of TCP Reno gives a more accurate estimation of the transfer times than existing models. The improved accuracy is partly due to a more accurate modeling of the timeouts, evolution of $cwnd$ during slow start and the delayed ACK timer. Our models also show that under the losses introduced by the droptail queues which dominate most routers in the Internet, current implementations of SACK can fail to provide adequate protection against timeouts and a loss of roughly more than half the packets in a round will lead to timeouts. We also show that with independent losses, SACK performs better than Tahoe and Reno and as losses become correlated, Tahoe can outperform both Reno and SACK.**

I. Introduction

Early TCP implementations used a go-back-n model and required the expiration of a retransmission timer to recover any loss. TCP Tahoe added the *slow-start, congestion avoidance* and *fast retransmit* algorithms to TCP [8]. TCP Reno retained the new algorithms of TCP Tahoe while adding *fast recovery* to the implementations [9]. TCP SACK allows receivers to ACK out of sequence data and is aimed at eliminating the timeouts which arise in TCP Reno due to multiple losses from the same window [1], [2]. The most recent proposal for adding SACK to TCP is in [15].

In this paper we first present analytic models for estimating the latencies and steady state throughput of TCP Tahoe, Reno and SACK. Our models are validated using both traces collected from the Internet as well as simulations. We then compare the performance of these versions of TCP under independent and correlated loss scenarios. Existing analytic models for TCP focus mainly on TCP Reno [4], [7], [11], [13], [17], [18] as it the most widely implemented version of TCP. Also, these models are for the steady state throughput of infinite TCP connections. In [10], models for the steady state throughput of Tahoe, Reno and NewReno are presented assuming an independent loss model without delayed ACKs. In [3] and [19] models for estimating the latency of TCP Reno are presented for correlated and independent losses respectively. Stability issues with TCP Vegas and its fairness and loss properties has been analyzed in [12].

Though TCP Reno has been modeled extensively, no literature exists on modeling the latencies of finite TCP flows using Tahoe or SACK. This paper, in addition to being the first to present models for the latency of TCP Tahoe and SACK, also presents a model for the latency of TCP Reno which gives more accurate results compared to existing models for TCP Reno. This enhanced accuracy can be attributed to the better modeling of the timeouts experienced by a TCP flow as well as a more accurate model for the evolution of $cwnd$ during slow start and its interaction with the delayed ACK timer. We also show a serious drawback in current SACK implementations (which use a variable `pipe`) which can lead to severe performance degradations. We show how TCP SACK can lead to timeouts even on the receipt of 3 duplicate ACKs and derive the precise conditions under which these timeouts occur.

Another important contribution of this work is to show that Tahoe can outperform both Reno and SACK under correlated losses. In [5] simulation scenarios were used show that both Tahoe and SACK outperform Reno in the presence of multiple losses in a window. Also, SACK performs better than Tahoe in the examples considered. However, as our analysis shows, with correlated losses which arise from the droptail queues dominating the routers in the Internet, the performance of the three versions can be quite different and Tahoe can perform better than both Reno and SACK. But, using simulations we show that if the loss scenario is changed to an independent model, both SACK and

Some of the results in this paper were presented at IEEE GLOBECOM'01, San Antonio, TX, USA.

Reno outperform Tahoe. This underlines the fact that a proper determination of the loss model is of utmost importance in determining TCP performance and the portability of such models across different environments is limited.

The rest of the paper is organized as follows. In Section II we present the assumptions made in our models and describe the three versions of TCP in more detail. Section III presents the models for the latencies and their validation while Section IV presents the models for the steady state throughput. In Section V we compare the performance of the three versions of TCP in terms of their latencies and their steady-state throughput. Finally we present a discussion on the results and concluding remarks in Section VI.

II. Background and Assumptions

We follow assumptions on the network and the TCP sender and receiver similar to those in [3], [18], [19]. These assumptions are enumerated below:

1. We account only for the delays arising from TCP's performance and ignore the delays arising at the endpoints from factors like buffer limitations.
2. We assume that the sender transmits full sized segments as fast as its congestion window allows.
3. The receiver advertises a consistent flow control window, W_{max}.
4. We assume that the receiver uses the delayed acknowledgment scheme specified in RFC 2581.
5. As in [3], we do not account for the effects of Nagle's algorithm and silly window avoidance.
6. We model the latency of TCP flows in terms of "rounds" as defined in [18]. A round begins with the transmission of a window of packets and ends on the receipt of an ACK for one of these packets.

We note that successive rounds do not overlap and thus our models are more accurate for wide or metropolitan area networks.

In this paper we assume the correlated loss model of [18] which is better suited for the droptail queues currently prevalent in the Internet. In this model, a packet in a round is lost independently of losses in other rounds. However, losses within a round are correlated and all packets following the first packet to be lost in round are also assumed to be lost. We define p to be the probability that a packet is lost, given that it is either the first packet in its round or the preceding packet in its round is not lost. We also use the term *loss indication* to denote an event, arising either from a timeout or duplicate ACKs which causes the sender to infer a packet loss and result in the retransmission of one or more packets. Note that with correlated losses, multiple lost packets may be recovered from a single loss indication. Thus p also represents the frequency of loss indications. Finally we assume that the time to transmit all the packets is much smaller than the duration of the round and that the duration of the round is independent of the window size. Our models allow for arbitrary amounts of data to be transferred.

A. TCP Tahoe

Early TCP implementations used a go-back-n model with cumulative positive acknowledgments and the expiration of the retransmission timer was required before the flow could retransmit any lost packets. TCP Tahoe added the *slow-start, congestion avoidance* and *fast recovery* algorithms to TCP [8]. With fast retransmit, when a packet is lost, instead of waiting for the retransmission timer to expire, if $tcprexmtthresh$ (usually 3) duplicate ACKs are received, the sender infers a packet loss and retransmits the lost packet. The sender now sets its $ssthresh$ to half the current value of $cwnd$ (maintained in bytes) and begins again in the slow-start mode with an initial window of 1. The slow start phase lasts till the $cwnd$ reaches $ssthresh$ and then congestion avoidance takes over. In this phase the sender increases its $cwnd$ linearly by $MSS * MSS/cwnd$ for every new ACK it receives. Note that with TCP Tahoe, the sender might retransmit packets which have been received correctly. Timeouts are used as the means of last resort to recover lost packets.

B. TCP Reno

TCP Reno has all the features of TCP Tahoe like slow-start, fast retransmit and congestion avoidance. It varies in the subsequent recovery phase which follows a loss and includes the *fast-recovery* algorithm which results in a more efficient transfer. When $tcprexmtthresh$ (usually 3) duplicate ACKs are received, TCP infers a loss and retransmits the lost packet and instead of dropping its window to 1, drops it by half and $ssthresh$ is set to half of $cwnd$. During fast-retransmit, the sender infers each duplicate ACK as one packet having left the network and increases its window by one resulting in a usable window of $\min\{W_{max}, \lceil W_{max}/2 \rceil + ndup\}$ where $ndup$ is the number of duplicate ACKs received. When the retransmitted packet is acknowledged, the sender exits fast-recovery with $ndup = 0$ and $cwnd = ssthresh$. The flow now enters the congestion avoidance mode and $cwnd$ follows a linear increase pattern. Note that TCP Reno can retransmit at most one lost packet per RTT.

C. TCP SACK

TCP Reno's fast-recovery is optimized for the case where a single packet is lost from a window. However

when multiple packets are dropped from the same window, Reno generally cannot recover all the losses without going into a timeout. TCP SACK aims at providing efficient retransmissions in the presence of multiple losses. The SACK option field in TCP SACK ACKs report non-contiguous blocks of data which have been received correctly [15]. In our model we assume that the SACK option has room for three SACK blocks. As in Reno, TCP SACK enters fast-retransmit on the receipt of $tcprexmtthresh$ duplicate ACKs. The sender now retransmits a lost packet and reduces its $cwnd$ by half and exits fast-recovery only when an ACK is received acknowledging all the data that was outstanding when fast-recovery was entered. SACK implementations use a variable `pipe` to estimate the number of packets outstanding in the network [5]. The sender now sends new or retransmitted packets only if the value of `pipe` is less than $cwnd$. When $tcprexmtthresh$ duplicate ACKs are received, `pipe` is initialized to $cwnd - tcprexmtthresh$. The variable *pipe* is now incremented by one with the transmission of any new packet or the retransmission of a lost packet. When a duplicate ACK is received with the SACK option reporting that new data has been received, `pipe` is decremented by one. However, on the receipt of a partial ACK (ACK received during fast-recovery which advances the acknowledgment number field but does not take the sender out of fast-recovery) *pipe* is decremented by two. This ensures that SACK never recovers more slowly that slow-start. Again, timeouts are used as the last means of recovery. There are a number of proposals for congestion control algorithms using SACK [14], [6] but we assume the implementation of [5].

III. Models for TCP Latency

Our approach towards modeling the latency TCP flows is to estimate the transfer time given that the flow experiences a given number of loss indications. When multiplied by the probability that the flow experiences the specified number of losses, this gives us the expected transfer time. We break the modeling in three parts: (1) flows with no losses, (2) flows with a single loss indication and (3) flows with multiple loss indications. The expressions for the latency are derived as a function of the transfer size N (in number of packets), the packet loss probability p and the RTT. We first present the model for estimating the latency in the connection establishment phase. Next, we present a model for the number of packets sent during the slow start phase and then follow it up with the expression for the transfer time when there are no losses. In the absence of losses, all the three versions have identical behavior and the same expression can be used for all three cases.

A. Connection Establishment

A TCP connection begins with a three-way handshake with the initiating host sending a SYN packet. The receiver responds with an ACK for the initiating host's SYN and also sends a SYN packet of its own. When the initiating host receives this SYN/ACK packet it assumes that the connection has been established and confirms this by sending an ACK of its own. During this process, if either host does not receive the ACK it is expecting within a timeout period T_s, it retransmits its SYN and then waits twice as long for an ACK. Following [3], the expected duration of the connection setup time can be approximated as

$$t_{setup} = RTT + 2T_s\left(\frac{1-p}{1-2p} - 1\right) \qquad (1)$$

B. The Slow-start Phase

TCP starts its transmission in the slow-start phase and increases its $cwnd$ by one MSS for every ACK it receives. With delayed acknowledgments, the receives sends one ACK for every two packets that it receives or if the delayed acknowledgment timer expires. In most UNIX based systems the delayed acknowledgment timer is set to 200ms leading to an expected delay of 100ms before the ACK for the first packet in the flow is sent (assuming an initial window of 1). In Windows 95 and Windows NT 4.0 systems this delay is uniformly distributed between 100ms and 200ms. Since the sender generally sends one ACK for two packets, with delayed ACKs, the rate of increase in the $cwnd$ is roughly 1.5 and this is commonly assumed in TCP modeling papers [3]. However, due to the probability that the delayed ACK timer may expire before the end of the current round, the window increase pattern is different in practice and can make significant differences for short transfers. To illustrate this, in Fig. 1 we show two possible ways in which the $cwnd$ may evolve starting from 1 depending on the state of the delayed ACK timer. In the figure, consider the third round which has $cwnd = 3$ and in which packet number 4, 5 and 6 are transmitted. The receiver sends an acknowledgment for the first two packets and delays sending the acknowledgment for packet number 6. If a new packet arrives before the ACK timer expires, packet number 6 is acknowledged along with the new packet and the subsequent $cwnd$ increase pattern is shown in Fig. 1(a). However, if the delayed ACK timer expires before a new packet arrives, an ACK is sent with results in a $cwnd$ of 5 as shown in Fig. 1(b).

To account for such complex behavior of the $cwnd$, we develop a more accurate model which tries to model the expected value of the $cwnd$ for any round. This results in a more accurate representation of the number of packets

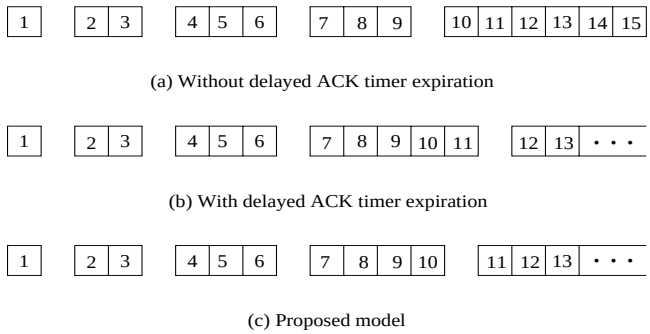


Fig. 1. Window increase patterns for TCP flows in the slow start phase.

transmitted in any round as compared to 1.5^n for the n^{th} round as used in [3]. The number of packets transmitted in the n^{th} round according to our model is given by

$$w(n) = \left\lfloor 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} \right\rfloor \tag{2}$$

Note that our model predicts the number of packets transmitted in the fourth round as 4 (Fig. 1(c)) which is the average of the packets transmitted in examples (a) and (b). The number of packets transmitted in the first k rounds of the slow start phase is then given by

$$\begin{aligned} pkt(k) &= \sum_{n=1}^{k} w(n) = \sum_{n=1}^{k} \left\lfloor 2^{\frac{n-1}{2}} + 2^{\frac{n-2}{2}} \right\rfloor \\ &= \left\lfloor 2^{\frac{k+1}{2}} + 3(2)^{\frac{4k-3}{8}} - 2 - \frac{3\sqrt{2}}{2} \right\rfloor \end{aligned} \tag{3}$$

Also, if the flow starts with a $cwnd$ of one, the receiver waits for a second packet to arrive before sending an ACK. The ACK is subsequently sent when the delayed ACK timer expires and the expected value of this delay is 100ms for UNIX systems and 150ms for Windows [3] and we denote it by t_{dack}.

C. Timeouts and Congestion Avoidance

During the slow-start phase, the window increases exponentially till it transmits all the packets or till it reaches $\min\{ssthresh, W_{max}\}$. However, if the flow experiences a loss, the window reduces and starts increasing again either in the slow-start or congestion avoidance mode. If the first packet to be retransmitted following a timeout is also lost, the retransmission timer backs off exponentially and we have another timeout period of twice the length. The average duration of a timeout, accounting for the exponential backoff of the retransmission timer on the loss of a retransmitted packet, is given by [18]

$$E[TO] = T_o \frac{1 + p + 2p^2 + 4p^3 + 8p^4 + 18p^5 + 32p^6}{1 - p} \tag{4}$$

where T_o is the duration of time the sender waits before retransmitting the first lost packet. Once the flow reaches the congestion avoidance phase, the $cwnd$ increases linearly till it reaches W_{max}, increasing by 1 every two RTTs. The number of rounds required to transmit a packets in the congestion avoidance mode with the initial value of $cwnd = b$ is obtained by solving $a = \sum_{i=1}^{k}(b + \lfloor \frac{i-1}{2} \rfloor)$ for k. The solution for this equation (and accounting for the effect of window limitation) is given by

$$t_{lin}(a,b) = \begin{cases} \left\lceil \frac{a - x(x+1) + b(b-1)}{x+1} \right\rceil + 2x & \text{if } a \le N_{lin} \\ -2(b-1) & \\ \left\lceil \frac{a - W_{max}(W_{max}+1) + b(b-1)}{W_{max}} \right\rceil & \text{otherwise} \\ +2W_{max} - 2(b-1) & \end{cases} \tag{5}$$

where $N_{lin} = W_{max}(W_{max}+1) - b(b-1)$ and denotes the number of packets that can be sent before $cwnd$ reaches W_{max} and $x = \lfloor (-1 + \sqrt{1 + 4(a + b(b-1))})/2 \rfloor$.

D. Flows Without Losses

If the TCP flow does not experience any losses, the behavior of all the three versions of TCP under consideration will be identical. Starting with an initial value (assumed 1), the $cwnd$ increases exponentially till all the packets are transferred or till it reaches W_{max} and stays there till the transfer is complete. The number of round required by the TCP flow to reach a $cwnd$ of W_{max} can be computed from Eqn. (2) and is given by

$$n_{wm} = \left\lfloor 2 \log_2 \left(\frac{2W_{max}}{1 + \sqrt{2}} \right) \right\rfloor \tag{6}$$

The number of packets transmitted so far when $cwnd$ reaches W_{max}, N_{exp}, is then given by

$$N_{exp} = \left\lfloor 2^{\frac{n_{wm}+1}{2}} + 3(2)^{\frac{4n_{wm}-3}{8}} - 2 - \frac{3\sqrt{2}}{2} \right\rfloor + W_{max} \tag{7}$$

The time to transfer N packets, $N < N_{exp}$, can be obtained by solving $N = \sum_{n=1}^{k} w(n)$ for k. The transfer time for N packets is then

$$t_{nl}(N) = \begin{cases} \left\lceil 2 \log_2 \left(\frac{2N + 4 + 3\sqrt{2}}{2\sqrt{2} + 3(2)^{\frac{5}{8}}} \right) \right\rceil RTT, & \text{if } N \le N_{exp} \\ \left\lceil n_{wm} + \left\lceil \frac{N - N_{exp}}{W_{max}} \right\rceil \right\rceil RTT, & \text{otherwise} \end{cases} \tag{8}$$

E. TCP Tahoe

We now consider the cases of flows with losses and begin with TCP Tahoe. When a Tahoe flow does not experience any losses, the time taken to transfer N packets is given by Eqn. (8). We model the case when a Tahoe flow

experiences losses by breaking the analysis in two parts: when there is a single loss indication and when there are multiple loss indications.

E.1 Single Loss

We first present some expressions which will be used frequently in the derivations. To find the $cwnd$ of the round when the i^{th} packet is transmitted, $cwnd_i^0$, we first find the number of rounds it takes to transmit i packets, $r(i)$, assuming there is no effect of window limitation. $r(i)$ can be calculated as in Eqn. (8) and is given by

$$r(i) = \left\lceil 2\log_2\left(\frac{2i+8.243}{7.455}\right)\right\rceil \tag{9}$$

If $r(i) \geq n_{wm}$, the number of rounds it takes for $cwnd$ to reach W_{max}, we know that $cwnd = W_{max}$. For all other cases, $cwnd < W_{max}$ and is given by Equation (2). Then, $cwnd_i^0$, the $cwnd$ of the round when the i^{th} packet is transmitted, is given by

$$cwnd_i^0 = \begin{cases} W_{max} & \text{if } r(i) > n_{wm} \\ \left\lfloor 2^{\frac{r(i)-1}{2}} + 2^{\frac{r(i)-2}{2}} \right\rfloor & \text{otherwise} \end{cases} \tag{10}$$

The sequence number of the last packet transmitted in this round, $n_{max}(i)$, can be obtained using Equation (3) and is given by

$$n_{max}(i) = \begin{cases} N_{exp} + \left\lceil \frac{\max(0, i-N_{exp})}{W_{max}} \right\rceil W_{max} & \text{if } r(i) > n_{wm} \\ \left\lfloor 2^{\frac{r(i)+1}{2}} + 3(2)^{\frac{4r(i)-3}{8}} - 2 - \frac{3\sqrt{2}}{2} \right\rfloor & \text{otherwise} \end{cases} \tag{11}$$

From our correlated loss model, if packet i is lost, then all the packets following it in the same round, i.e., $i+1$ to $n_{max}(i)$ are also lost and we denote the number of losses by $nloss$. Thus, we have $nloss = n_{max}(i) - i + 1$. Also, in this round $cwnd_i^0 - nloss = i - n_{max}(i) + cwnd_i^0 - 1$ packets are transmitted correctly before the i^{th} packet. Since the receiver sends one ACK for every two packets that it receives, we can thus expect ACKs for half of these packets, each of which increases $cwnd$ by one. We denote the $cwnd$ in the round which now follows by $cwnd_i^1$ and the number of packets transmitted in it by $nrnd_i^1$. Since we recieve ACKs for roughly half of the packets sent correctly in the previous round ($cwnd_i^0 - nloss$), $cwnd_i^1$ can increase to at most $cwnd_i^0 + \lceil (cwnd_i^0 - nloss)/2 \rceil$ unless limited by W_{max}. However, for the round corresponding to the $cwnd$ of $cwnd_i^1$, the $nloss$ packets lost in the previous round are already backlogged. Thus at most $cwnd_i^1 - nloss$ new packets can be sent in this round resulting in $nrnd_i^1 = cwnd_i^1 - nloss$. On the receipt of three duplicate ACKs or on the expiration of the retransmission timeout, the sender retransmits the lost packets and sets its $ssthresh$ to $n = \max\{2, \lceil cwnd_i^1/2 \rceil\}$. Let the duration of the slow start phase following a loss indication in TCP Tahoe be denoted by $r(n)$ and the number of packets transmitted in these rounds be denoted by k'. Note that since k' corresponds to the number of packets sent in the slow-start phase, we can use Eqn. (3) to evaluate it. Each of the quantities defined above is then given by

$$\begin{aligned} nloss &= n_{max}(i) - i + 1 \\ cwnd_i^1 &= \min\{W_{max}, cwnd_i^0 + \lceil (cwnd_i^0 - nloss)/2 \rceil\} \\ nrnd_i^1 &= cwnd_i^1 - nloss \\ n &= \max\{2, \lceil cwnd_i^1/2 \rceil\} \\ k' &= \left\lfloor 2^{\frac{r(n)+1}{2}} + 3(2)^{\frac{4r(n)-3}{8}} - 2 - \frac{3\sqrt{2}}{2} \right\rfloor \end{aligned} \tag{12}$$

and $r(n)$ is obtained from Eqn. (6) with W_{max} replaced by n. Following the round where the loss occurs, we have another round of size $nrnd_i^1$ whose packets result in duplicate ACKs. Now if $nrnd_i^1 \geq 3$, the sender will get at least three duplicate ACKs for packet i and enter the slow start mode after retransmitting packet i. Otherwise, not enough duplicate ACKs are received and the connection times out. In the slow start phase which follows both the fast-retransmit and the timeout, k' packets are transmitted before congestion avoidance takes over. When the congestion avoidance phase begins, $nrnd_i^1 + k'$ additional packets have been correctly transferred. The transfer time for the remaining $a = N - nrnd_i^1 - k' - i + 1$ packets to be transmitted in the linear increase phase can be found using Eqn. (5) with $b = n$. Also, the time to transmit the first $i-1$ packets can be obtained using the expression for the no loss case. The time to transmit N packets with a loss indication at packet i is then given by

$$t_{sl}(N) = \begin{cases} [t_{nl}(i) + r(n) + 1 & \text{if } nrnd_i^1 \geq 3 \\ \quad + t_{lin}(a,b)]\, RTT & \\ [t_{nl}(i) + r(n) + t_{lin}(a,b) + & \text{else} \\ \quad I(nrnd_i^1 > 0) + E[TO]]\, RTT & \end{cases} \tag{13}$$

The term $I(nrnd_i^1 > 0)$ is a consequence of the fact the we have an additional round following the one in which the packets were lost only if at least some packets from the original round were transmitted successfully. The expected duration to transmit the N packets with a single loss indication can now be obtained by averaging over the N possible values of i.

E.2 Multiple Losses

We now consider the case when the the flow experiences multiple loss indications. Let there be M loss indications

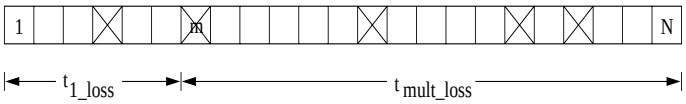


Fig. 2. Modeling a flow by breaking it up into the section which has the first loss and the remaining section with the rest of the losses.

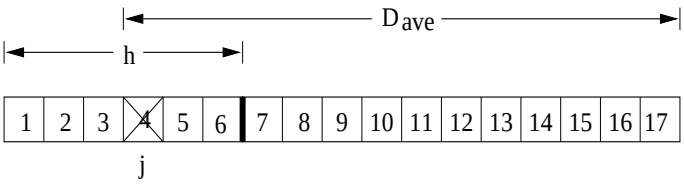


Fig. 3. The parameters h and j and their relation to D_{ave}.

in a flow of N packets, the second of which occurs at packet number m. Please refer to Fig. 2 for an illustration. The first $m-1$ packets have one loss indication and the time to transmit these packets can be obtained from the results of the previous subsection. The final $N-m+1$ packets contain $M-2$ loss indications and the average number of packets between two successive loss indications is given by

$$D_{ave} = \frac{N-m+1}{M-1} \tag{14}$$

To obtain the transfer time for the last $N-m+1$ packets, we first compute the average time to transmit D_{ave} packets. After the first loss, we approximate the possible range of values of $cwnd$ when the subsequent losses occur by $1, \cdots, \lceil \frac{-1+\sqrt{1+16D_{ave}}}{2} \rceil\}$. This approximation is based on the following: Firstly, since the average number of packets between two successive loss indications is D_{ave}, we approximate the upper limit of the number of packets between two successive loss indications by $2D_{ave}$. We now note that after the loss is inferred, Tahoe starts with a window of 1 and even though it is in the slow start phase, the number of packets transmitted in the first 4 rounds are 1, 2, 3 and 4 respectively as shown in Fig. 1(c) and from Eqn. (2) and beyond that the increase is not linear. We also note that it is quite likely that after 4 rounds, the flow enters the congestion avoidance mode where the window increase is linear. If we assume that all the $2D_{ave}$ packets are transmitted following the pattern just mentioned, the number of rounds, n, required to transmit the $2D_{ave}$ packets is obtained by solving $n(n+1)/2 = 2D_{ave}$ for n. The solution $n = \frac{-1+\sqrt{1+16D_{ave}}}{2}$ results in the approximate upper limit for the range of the $cwnd$ for the multiple loss case. To account for the effect of window limitation on the possible values of $cwnd$, we limit its range to $\min\{W_{max}, \lceil \frac{-1+\sqrt{1+16D_{ave}}}{2} \rceil\}$. Also, to keep the analysis tractable, we assume that each of these possible values of $cwnd$ and the position of the lost packet within a $cwnd$ are equally likely when the loss occurs.

Now consider the flow with $cwnd = h$ where the loss indication occurs at the j^{th} packet of the round (see Figure 3). We want to find the time to transmit D_{ave} packets correctly following the j^{th} packet. For analytic tractability, we now assume that the flow was in the congestion avoidance mode when this loss indication occurred, as it will subsequently be following the first loss indication. The number of packets lost in this round is now given by $h-j+1$ while $j-1$ packets are transmitted successfully. Each successful packet slides the window forward and we have another $j-1$ packets transmitted in the subsequent round. When the loss is detected either though duplicate ACKs or timeout, $ssthresh$ is set to $n = \max\{2, \lceil h/2 \rceil\}$. In case of a timeout we need $r(n)$ rounds with k' packets transmitted in them till the flow enters the congestion avoidance mode again. Using the same definition for the quantities $nloss$, $cwnd_j^0$, $cwnd_j^1$, $nrnd_j^1$, $r(n)$ and k' as for the single loss case, we have

$$\begin{aligned} cwnd_j^0 &= h \\ nloss &= h-j+1 \\ cwnd_j^1 &= h \\ nrnd_j^1 &= cwnd_j^1 - nloss = j-1 \end{aligned} \tag{15}$$

and $r(n)$ and k' are obtained from Eqn. (9) and (12) respectively. Now if $nrnd_j^1 \geq 3$, we get enough duplicate ACKs to lead to a fast retransmit. In the slow start phase that follows both the timeouts and the fast retransmit, we transmit another k' packets leaving $a = D_{ave} - k' - j + 1$ packets to be transmitted in the congestion avoidance phase. The time to transmit D_{ave} packets after the loss indication is thus

$$t_{M\_loss}(D_{ave}) = \begin{cases} [r(n)+1 & \text{if } nrnd_i^1 \geq 3 \\ +t_{lin}(a,n)]\, RTT & \\ [t_{lin}(a,n) + E[TO] + & \text{otherwise} \\ r(n) + I(j-1>0)]\, RTT & \end{cases} \tag{16}$$

The expected duration to transmit the D_{ave} packets can now be obtained by averaging Eqn. (16) for the possible value of h and j. The time to transmit the N packets with M loss indications is then

$$t_{ml}(N) = E\{t_{sl}(m-1)\} + (M-2)E\{t_{M\_loss}(D_{ave})\} \tag{17}$$

where the expectation operation is carried over all possible values of m and the number of losses M.

File Size	$p = 0.01$		$p = 0.05$		$p = 0.10$	
	sim.	ana.	sim.	ana.	sim.	ana.
1 KB	0.28	0.28	0.47	0.47	0.94	1.04
4 KB	0.39	0.37	0.75	0.76	1.56	1.47
8 KB	0.72	0.68	1.23	1.27	1.78	1.82
16 KB	0.82	0.81	1.69	1.73	2.73	2.68

File Size	$p = 0.01$		$p = 0.05$		$p = 0.10$	
	sim.	ana.	sim.	ana.	sim.	ana.
1 KB	0.36	0.35	0.63	0.64	1.02	1.17
4 KB	0.63	0.60	1.08	1.12	1.81	1.74
8 KB	0.99	0.94	1.48	1.52	2.62	2.69
16 KB	1.42	1.37	2.16	2.21	3.47	3.49

TABLE I
TCP TAHOE TRANSFER TIMES. $RTT = 100ms$ (TOP) AND $RTT = 200ms$ (BOTTOM), $MSS = 1.4KB$, $W_{max} = 24$.

E.3 The Expected Transfer Time

Combining the results of the previous sections, the expected transfer time for a flow of N packets is given by

$$T_{transfer}(N) = t_{setup} + t_{dack} + (1-p)^N t_{nl}(N) + p(1-p)^{N-1} E\{t_{sl}(N)\} + t_{ml}(N) \quad (18)$$

where t_{setup}, $t_{nl}(N)$, $t_{sl}(N)$ and $t_{ml}(N)$ are defined in Eqns. (1), (8), (13) and (17) respectively.

In Table I we present the comparison of the results from our model with those from simulations using *ns*. A 2-state Markov error model was used to model the correlated loss process. We note the close agreement between the analytic and simulation results.

F. TCP Reno

TCP Reno adds the fast-recovery algorithm to TCP Tahoe which can results in substantial improvements in the presence of single packet losses. However with multiple packet losses, TCP Reno has to resort to timeouts to recover the losses and its performance degrades. When the Reno flow does not experience any losses, the transfer time is obtained using Eqn. (8). We now present the models for the single and multiple loss cases.

F.1 Single Loss

We again consider a flow of N packets where the loss indication occurs at the i^{th} packet. Again, since we have correlated losses, packets $i+1$ to $n_{max}(i)$ are also lost. Then, following the notation used for the Tahoe model we again obtain

$$\begin{aligned} nloss &= n_{max}(i) - i + 1 \\ cwnd_i^1 &= \min\{W_{max}, cwnd_i^0 + \lceil (cwnd_i^0 - nloss)/2 \rceil\} \\ nrnd_i^1 &= cwnd_i^1 - nloss \end{aligned}$$

where $cwnd_i^0$ from Eqn. (10). However, since Reno can recover multiple lost packets (with one recovery per round), we can have another round of packets with $cwnd$ denoted by $cwnd_i^2$ before a timeout may be detected. If $nrnd_i^1 \geq 3$, on the receipt of the third duplicate ACK, the $cwnd$ drops to half of its current value and then increments by one for each additional duplicate ACK. Thus $cwnd_i^2$ is given by

$$cwnd_i^2 = \min\{W_{max}, \lceil cwnd_i^1/2 \rceil + nrnd_i^1\} \quad (19)$$

The number of packets still unacknowledged at the end of first round following the round with the losses is $cwnd_i^1$. Thus if $cwnd_i^2 > cwnd_i^1$, the number of additional packets sent in the next round along with the retransmitted packet is then

$$nrnd_i^2 = cwnd_i^2 - cwnd_i^1 \quad (20)$$

The value of the $cwnd$ when the loss is inferred is $cwnd_i^1$. It is known that a single packet loss in a window of less than 4 ($tcprexmtthresh + 1$), two or more losses in windows between 4 and 8 ($tcprexmtthresh + 1$ and $2(tcprexmtthresh + 1)$) and three or more losses for higher windows lead to timeouts in TCP Reno [16]. For all other cases, the losses can be recovered using fast retransmissions. Let us first consider the simpler case when the losses are recovered using fast retransmissions. Since $nrnd_i^1 + nrnd_i^2$ new packets along with the $nloss$ retransmitted packets are sent before the congestion avoidance phase begins, we only have $a = N - nloss - nrnd_i^1 - nrnd_i^2 - i + 1$ packets to send in congestion avoidance phase. Also, since each recovered loss reduces $ssthresh$ by half, we have $n = \max\{2, \lceil cwnd_i^1/2^{nloss} \rceil\}$. Since we need one round to recover each of the $nloss$ losses the time to transmit the N packets is then

$$t_{sl}(N) = [t_{nl}(i) + nloss + 1 + t_{lin}(a, n)]\, RTT \quad (21)$$

If the flow times out while recovering the losses, we can have between 0 and 3 additional rounds of packets where the first couple of losses may be recovered. If the first packet of the round is lost then we do not have any additional rounds and the flow directly times out. If $nrnd_i^1$ is one or two, then we have another round with $nrnd_i^1$ packets before the flow times out. In this case all losses are recovered using timeouts and $ssthresh$ is set to $cwnd_i^1/2$. For other cases we have at least one packet which is recovered using fast recovery. In this round $nrnd_i^2$ additional packets are sent along with the retransmitted packet and $ssthresh$ becomes $cwnd_i^1/4$ if the flow times out now. Also, if $nrnd_i^2 \geq 3$ we have a third additional round with another fast retransmit. The flow inevitably times out now with $ssthresh$ set to $cwnd_i^1/8$. We denote the number

of additional packets transmitted before the timeout by k'' and during the slow start phase following the timeout by k'. We also denote the number of rounds before the timeout starts by t_{TO} and the duration of the slow start phase by $r(n)$. After some algebraic simplifications we then have

$$k'' = \begin{cases} nrnd_i^1 & \text{if } nrnd_i^1 < 3 \\ \max\{cwnd_i^1, cwnd_i^2\} - nrnd_1^1 - 1 & \text{if } nrnd_i^2 < 3 \\ \max\{cwnd_i^1, cwnd_i^2\} - nrnd_1^1 & \text{otherwise} \end{cases} \quad (22)$$

$$t_{TO} = \begin{cases} I(nrnd_i^1 > 0) & \text{if } nrnd_i^1 < 3 \\ 2 & \text{if } nrnd_i^2 < 3 \\ 3 & \text{otherwise} \end{cases} \quad (23)$$

$$n = \begin{cases} \max\{2, \lceil cwnd_i^1/2 \rceil\} & \text{if } nrnd_i^1 < 3 \\ \max\{2, \lceil cwnd_i^1/4 \rceil\} & \text{if } nrnd_i^2 < 3 \\ \max\{2, \lceil cwnd_i^1/8 \rceil\} & \text{otherwise} \end{cases} \quad (24)$$

with $r(n)$ and k' given by Eqns. (9) and (12) respectively. Since we only have to transmit $a = N - k' - k'' - i + 1$ packets in the congestion avoidance phase, the time to transmit N packets is then given by

$$t_{sl}(N) = [t_{nl}(i) + t_{TO} + r(n) + E[TO] + t_{lin}(a, n)]\, RTT \quad (25)$$

and its expected value can be obtained by averaging over the N possible values of i.

F.2 Multiple Losses

We follow the same approach as used in modeling Tahoe flows with multiple losses and use the same expressions for finding D_{ave} and the range of possible $cwnd$ values. We break up a flow of N packets with M losses into the first part with $m-1$ packets with a single loss indication and the rest into $M-2$ chunks of D_{ave} packets. We again limit the possible range of values of $cwnd$ when the subsequent losses occur by $\min\{w_{max}, \lceil \frac{-1+\sqrt{1+16D_{ave}}}{2} \rceil\}$. Note that while the upper limit of this range was derived in Section III-E.2 assuming that the flow starts with a $cwnd$ of 1 following the loss indication, this is not always true for TCP Reno and SACK. However, this approximation works well as can be seen from the results and is used due to its analytic tractability.

Now consider the flow with $cwnd = h$ where the loss indication occurs at the j^{th} packet of the round. Using the same definition for the quantities $nloss$, $cwnd_j^0$, $cwnd_j^1$, $nrnd_j^1$ as for Tahoe flows, we again have

$$\begin{aligned} cwnd_j^0 &= h \\ nloss &= h - j + 1 \\ cwnd_j^1 &= h \\ nrnd_j^1 &= cwnd_j^1 - nloss = j - 1 \end{aligned}$$

As in the single loss case, if $nrnd_j^1 \geq 3$ we have another round where one loss is recovered using a fast retransmit. The $cwnd$ of this round, $cwnd_j^2$ is again given by Eqn. (19) and if $cwnd_i^2 > cwnd_i^1$, the number of additional packets sent in this round, $nrnd_1^2$ is given by Eqn. (20). The same conditions now apply to determine whether the flow has to resort to timeouts to recover all the losses. Consider the case when all the losses are successfully recovered using fast retransmits. In the congestion avoidance mode which follows, we need to transmit $a = D_{ave} - nloss - nrnd_j^1 - nrnd_j^2$ packets and the linear increase phase begins with a $cwnd$ of $n = \max\{2, \lceil cwnd_i^1/2^{nloss} \rceil\}$ since $ssthresh$ is halved by every fast retransmission. The time to transmit the D_{ave} packets can now be expressed as

$$t_{M\_loss}(D_{ave}) = [nloss + 1 + t_{lin}(a, n)]\, RTT \quad (26)$$

If the flow resorts to timeouts to recover the losses, we can again have between 0 and 3 rounds before the timeout starts and the conditions leading to these cases are the same as for the single loss case. Using the same notation for k'', k', t_{TO} n and $r(n)$ as in the single loss case, we can then use Eqns. (22), (12), (23), (24) and (9) respectively to calculate each of them. In the congestion avoidance phase we are now left with $a = D_{ave} - k' - k''$ to be transmitted. The time to transmit the D_{ave} packets can now be expressed as

$$t_{M\_loss}(D_{ave}) = [E[TO] + t_{TO} + r(n) + t_{lin}(a, n)]\, RTT \quad (27)$$

The expected duration to transmit the D_{ave} packets can now be obtained by averaging Eqns. (26) and (27) over all possible value of h and j. The time to transmit the N packets with M loss indications is then

$$t_{ml}(N) = E\{t_{sl}(m-1)\} + (M-2)E\{t_{M\_loss}(D_{ave})\} \quad (28)$$

F.3 The Expected Transfer Time

Combining the results of the previous sections, the expected transfer time for a flow of N packets is given by

$$\begin{aligned} T_{transfer}(N) &= t_{setup} + (1-p)^N t_{nl}(N) + t_{dack} + \\ &\quad p(1-p)^{N-1} E\{t_{sl}(N)\} + t_{ml} \end{aligned} \quad (29)$$

where t_{setup}, $t_{nl}(N)$ and $t_{ml}(N)$ are defined in Eqns. (1), (8) and (28) respectively and $t_{sl}(N)$ is defined in Eqns. (21) and (25).

In Tables II we present the comparison of the results from our model with those from *ns* simulations with a 2-state Markov loss model. Again we have a close agreement between the analytic and simulation results. For TCP

File Size	$p = 0.01$		$p = 0.05$		$p = 0.10$	
	sim.	ana.	sim.	ana.	sim.	ana.
1 KB	0.27	0.26	0.53	0.54	0.96	0.96
4 KB	0.42	0.41	0.83	0.85	1.63	1.62
8 KB	0.63	0.62	1.18	1.14	2.05	2.08
16KB	0.80	0.80	1.34	1.37	2.66	2.68

File Size	$p = 0.01$		$p = 0.05$		$p = 0.10$	
	sim.	ana.	sim.	ana.	sim.	ana.
1 KB	0.38	0.36	0.70	0.65	1.00	1.06
4 KB	0.65	0.63	1.06	1.07	2.08	2.10
8 KB	0.97	0.87	1.62	1.64	2.47	2.52
16 KB	1.41	1.31	2.16	2.17	3.72	3.71

TABLE II
TCP RENO TRANSFER TIMES. $RTT = 100ms$ (TOP) AND $RTT = 200ms$ (BOTTOM), $MSS = 1.4KB$, $W_{max} = 24$.

Reno we also validate our model by comparing the delays predicted by our model with measurements of actual TCP transfers over the Internet. These measurements were done for files with randomly generated sizes transferred between machines at RPI Troy, NY and Ohio State University, Columbus OH, MIT, Boston MA, University of California, Los Angeles CA and University of Pisa, Pisa Italy. To avoid repetitive graphs, we show the results for transfers to Ohio and Los Angeles in Fig. 4. The results for the others are similar. We also plot the results from the model given in [3]. For short transfers, our results are clearly much closer to the measured average and the improvement in the accuracy is in the order of 15-20%. For large transfers there is no appreciable difference between the two models.

G. TCP SACK

TCP SACK is specially aimed at eliminating timeouts in the presence of multiple losses. However, we now show that with correlated losses, SACK can fail to achieve this goal and timeouts can occur quite frequently. For our analysis, we assume the SACK implementation of [5]. We use the same definitions from the previous subsections and obtain $cwnd_i^0$ from Eqn. (10) and $nloss$, $cwnd_i^1$ and $nrnd_i^1$ from Eqn. (12).

We first identify the cases in which TCP SACK can lead to a time out. It is easy to see that if $nrnd_i^1 < 3$ (i.e. $tcprexmtthresh$) the sender does not receive enough duplicate acknowledgments and resorts to timeouts. However, in the presence of correlated losses, there is another factor which leads to most of the timeouts in current SACK implementations and this is the variable `pipe`. Unlike the fast retransmits of Reno, in TCP SACK, `pipe` controls the transmission of new or retransmitted packets during fast recovery. The sender sends new or retransmitted packets only if the value of `pipe` is less than $cwnd$. Let us consider a SACK flow where the loss indication occurs at packet i. When the third duplicate ACK is received and the first packet is retransmitted, `pipe` is initialized to $cwnd_i^1 - 3$ and $cwnd$ is set to $cwnd_i^1/2$. For each of the additional duplicate ACKs resulting from the rest of the $nrnd_i^1 - 3$ packets `pipe` is decremented by one resulting in

$$\texttt{pipe} = cwnd_i^1 - nrnd_i^1 = nloss \tag{30}$$

If $\texttt{pipe} \geq cwnd$ no more packets are transmitted. When the partial ACK corresponding to the first retransmitted packet arrives, `pipe` is decremented by two. Now if `pipe` is still greater than equal to $cwnd$ the flow will timeout. The condition for timeouts resulting due to the implementation of `pipe` can then be expressed as

$$nloss - 2 \geq \lceil cwnd_i^1/2 \rceil \tag{31}$$

This roughly translates to: if more than half of the packets in a round are lost, SACK is unable to recover them without a timeout. Thus with correlated losses, Tahoe can out perform SACK since it is the most conservative and starts retransmitting all lost packets on the receipt of 3 duplicate ACKs. This is verified in Section V.

G.1 Single Loss

We again consider a flow of N packets with a single loss indication occurring at packet i. The variables $cwnd_i^0$ $cwnd_i^0$, $nloss$, $nrnd_1$ and n are described as in the previous subsections and are given by Eqn. (12). Once the duplicate ACKs corresponding to the packets sent in the round following the round with losses are received, the current value of `pipe` is given by Eqn. (30).

Let us first consider the case when the flow resorts to a timeout to recover the losses, which happens if $nrnd_i^1 < 3$ or the if the condition of Eqn. (31) is satisfied. If $nrnd_i^1 > 0$ we have at least one round following the round with losses and if $nrnd_i^1 \geq 3$ we have one additional round where one packet is recovered with a fast retransmit. A fast retransmit results in the $cwnd$ being halved and the value of $cwnd$ and the number of packets transmitted in the next round, $nrnd_i^2$ are given by

$$\begin{aligned} cwnd_i^2 &= \lceil cwnd_i^1/2 \rceil \\ nrnd_i^2 &= 1 + \max\{0, cwnd_i^2 - \texttt{pipe}\} \end{aligned} \tag{32}$$

Also, when the loss is detected, $ssthresh$ is set to $n = \min\{2, \lceil cwnd_i^1/2 \rceil\}$ and the number of packets transmitted in the slow start phase, k' is obtained from Eqn. (12). The number of packets remaining to be transmitted in the

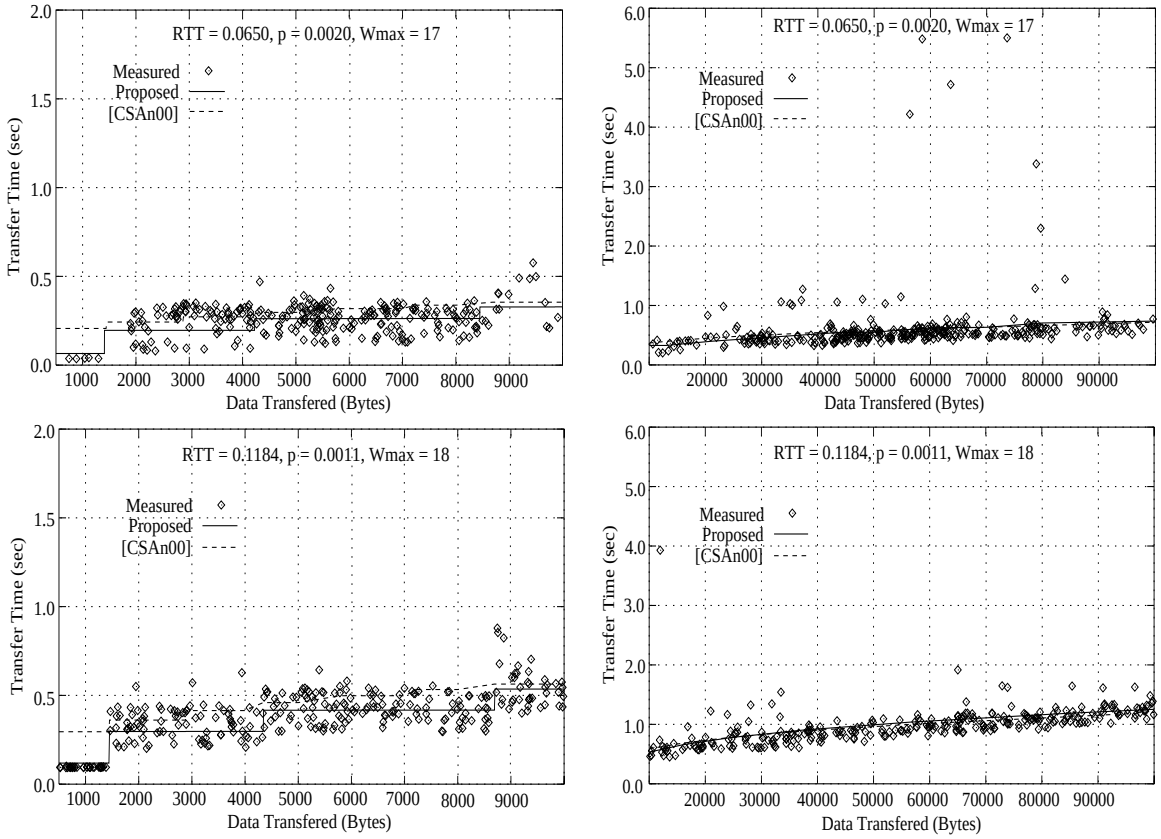


Fig. 4. Comparison of latency of TCP Reno transfer between Troy, NY and Columbus, OH (top) and Los Angeles, CA (bottom).

congestion avoidance mode is thus $a = N - nrnd_i^1 - nrnd_i^2 - k' - i + 1$. The time to transmit N packets with one loss indication resulting in a timeout is thus given by

$$t_{sl}(N) = \Big[t_{nl}(i) + E[TO] + I(nrnd_i^1 > 0) + I(nrnd_i^1 > 3) + t_{lin}(a,n)\Big] RTT \quad (33)$$

In the case that all the losses are recovered without any timeout, in each round the partial ACKs decrease `pipe` by two which ensures that SACK does not recover more slowly than slow start. Since $nrnd_i^2$ packets are retransmitted in the round following the receipt of the duplicate ACKs, the number of losses which remain to be transmitted, rl, is given by

$$rl = nloss - nrnd_i^2 \quad (34)$$

Also, as `pipe` decrements by two for each partial ACK and we start with an initial value of $nrnd_i^2$ packets each of which decrements `pipe` by two, the number of rounds spent in this exponential phase till all the losses are recovered, $r(n)$ and the packets transmitted in these $r(n)$ rounds, k', can be easily shown to be given by

$$\begin{aligned} r(n) &= \left\lceil \log_2\left(\frac{rl}{nrnd_i^2} + 1\right)\right\rceil \\ k' &= nrnd_i^2(2^{r(n)} - 1) \end{aligned} \quad (35)$$

We then have $a = N - nrnd_i^1 - nrnd_i^2(2)^{r(n)} - i + 1$ packets to be transmitted in the congestion avoidance mode resulting in a total transfer time of

$$t_{sl}(N) = [t_{nl}(i) + 2 + r(n) + t_{lin}(a,n)]RTT \quad (36)$$

The expected duration to transmit the N packets with a single loss indication can now be obtained by averaging over the N possible values of i.

G.2 Multiple Losses

Following the same approach as used for Tahoe and Reno and using the same definitions, we can again use Eqn. (15) to obtain $nloss$, $cwnd_j^0$, $cwnd_j^1$, $nrnd_j^1$. The value of `pipe` after the receipt of the duplicate ACKs can again be found using Eqn. (30). Again, if $nrnd_j^1 > 0$ we have at least one round following the round with losses and if $nrnd_j^1 \geq 3$ we have one additional round where one packet is recovered with a fast retransmit. The $cwnd$ of this round, $cwnd_j^2$, and the number of packets transmitted in this round, $nrnd_j^2$ are now similarly obtained using

$$\begin{aligned} cwnd_j^2 &= \lceil cwnd_j^1/2 \rceil \\ nrnd_j^2 &= 1 + \max\{0, cwnd_j^2 - \texttt{pipe}\} \end{aligned} \quad (37)$$

We can now use the same conditions as in the single loss case to determine the cases in which the SACK flow

File Size	$p = 0.01$		$p = 0.05$		$p = 0.10$	
	sim.	ana.	sim.	ana.	sim.	ana.
1 KB	0.31	0.30	0.62	0.62	0.96	1.09
4 KB	0.40	0.41	0.96	0.98	1.88	1.91
8 KB	0.65	0.62	1.12	1.08	2.00	1.04
16 KB	0.82	0.80	1.58	1.56	2.81	2.88

File Size	$p = 0.01$		$p = 0.05$		$p = 0.10$	
	sim.	ana.	sim.	ana.	sim.	ana.
1 KB	0.37	0.36	0.59	0.61	1.05	1.20
4 KB	0.59	0.59	1.08	1.05	1.81	1.79
8 KB	0.99	0.91	1.50	1.53	2.53	2.56
16 KB	1.43	1.40	2.22	2.36	3.58	3.63

TABLE III
TCP SACK TRANSFER TIMES. $RTT = 100ms$ (TOP) AND $RTT = 200ms$ (BOTTOM), $MSS = 1.4KB$, $W_{max} = 24$.

will timeout. We first consider the case when the flow times outs. When the loss is detected, $ssthresh$ is set to $n = \min\{2, \lceil cwnd_i^1/2 \rceil\}$ and the number of packets transmitted in the slow start phase, k' is again obtained from Eqn. (12). The number of packets remaining to be transmitted in the congestion avoidance mode is then $a = D_{ave} - nrnd_j^1 - nrnd_j^2 - k'$ and the time to transmit D_{ave} packets is thus

$$t_{M\_loss}(D_{ave}) = \Big[E[TO] + r(n) + I(nrnd_j^1 > 0) + I(nrnd_j^1 > 3) + t_{lin}(a,n)\Big] RTT \quad (38)$$

In the case that the losses are recovered without any timeouts, the number of losses which remain to be transmitted after the first round of retransmissions is $rl = nloss - nrnd_j^2$. Again, due to the reduction of `pipe` by two for every partial ACK, the number of retransmission increases as a power of two in every round and the time to transmit the remaining losses, $r(n)$, and the number of packets transmitted in these rounds, k' are given by

$$\begin{aligned} r(n) &= \left\lceil \log_2\left(\frac{rl}{nrnd_j^2} + 1\right)\right\rceil \\ k' &= nrnd_j^2(2^{r(n)} - 1) \end{aligned} \quad (39)$$

Now, only $a = D_{ave} - nrnd_j^1 - nrnd_j^2(2)^{r(n)}$ packets remain to be transmitted in the congestion avoidance mode and the transfer time for D_{ave} packets is given by

$$t_{M\_loss}(D_{ave}) = [2 + r(n) + t_{lin}(a,n)]\, RTT \quad (40)$$

The expected duration to transmit the D_{ave} packets can now be obtained by averaging Eqns. (38) and (40) for the possible value of h and j. The time to transmit the N packets with M loss indications is given by

$$t_{ml}(N) = E\{t_{sl}(m-1)\} + (M-2)E\{t_{M\_loss}(D_{ave})\} \quad (41)$$

where the expectation operation is carried over all possible values of m and the number of losses M.

G.3 The Expected Transfer Time

Combining the results of the previous sections, the expected transfer time for a flow of N packets is given by

$$T_{transfer}(N) = t_{setup} + (1-p)^N t_{nl}(N) + t_{dack} \\ p(1-p)^{N-1}E\{t_{sl}(N)\} + t_{ml}(N) \quad (42)$$

where t_{setup}, $t_{nl}(N)$ and $t_{ml}(N)$ are defined in Eqns. (1), (8) and (41) respectively and $t_{sl}(N)$ is defined in Eqns. (33) and (36).

In Table III we present the comparison of the results from our model with those from simulations using *ns*. Again, our results match very well with the simulation results.

IV. STEADY-STATE THROUGHPUT

In this section, we model the steady state throughput of infinite Tahoe, Reno and SACK flows by extending the models of the previous section. As in other steady state throughput modeling papers, we assume that the sender always has an unlimited amount of data to send. To extend the models of the previous section to infinite flows, we first note that with infinite packets, the probability that the flow experiences a single or no loss indications is 0 for $p > 0$. Also, for a packet loss probability of p, the average number of packets transmitted between two successive loss indications, d, is given by $1/p$. For the infinite flows, the time between two successive losses can be obtained using the equation for $t_{M\_loss}(d)$ for each of the three TCP versions. Using the average value of $t_{M\_loss}(d)$ and the average distance between two successive losses, we can now calculate the steady state throughput of the flow. The possible values of the $cwnd$ in this case is again similar for the multiple loss cases of the previous section. We assume the $cwnd$ to vary uniformly between 1 and c_{wm}, where c_{wm} is given by

$$c_{wm} = min\left\{\left\lceil(-1 + \sqrt{1 + 16/p})/2\right\rceil, W_{max}\right\} \quad (43)$$

The expected time to transfer d packets can now be obtained by averaging $t_{M\_loss}(d)$ over all possible values of the $cwnd$ and all possible positions of the loss indication within the round. Since the possible values of $cwnd$ vary uniformly between 1 and c_{wm}, each of these cases (tuple

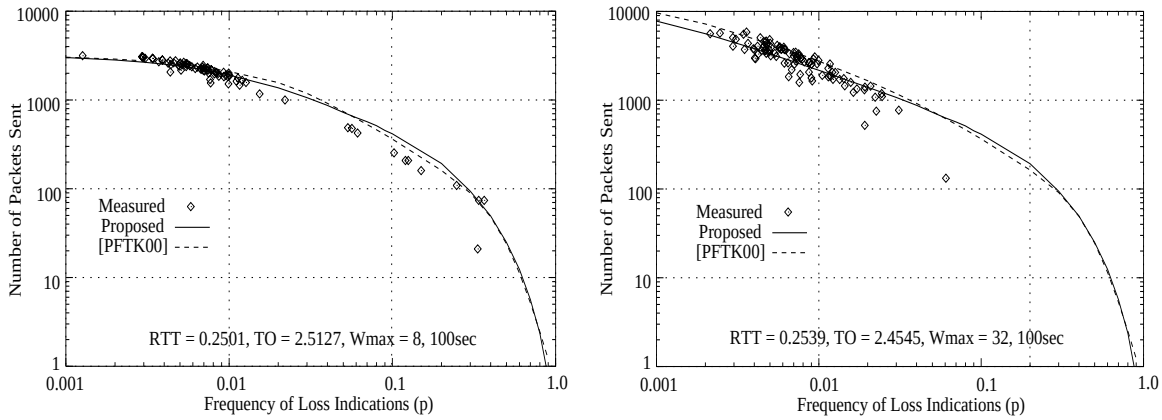


Fig. 5. Steady state throughput of TCP Reno connections.

corresponding to the $cwnd$ value and the position of the loss indication), resulting in a total of $c_{wm}(c_{wm}+1)/2$ cases, is equally likely. Thus each case occurs with a probability of $2/(c_{wm}(c_{wm}+1))$. The expected time, as a function of the loss probability, is then given by

$$t_{ss}(p) = \frac{2}{c_{wm}(c_{wm}+1)} \sum_{h=1}^{c_{wm}} \sum_{j=1}^{h} t_{M\_loss}(d) \qquad (44)$$

with the conditions for determining whether a loss indication results in a timeout or not is determined as in the previous section and $t_{M\_loss}(d)$ is given by Eqn. (16) for Tahoe, Eqns. (26) and (27) for Reno, and Eqns. (38) and (40) for SACK. Since d packets corresponding to $dMSS$ bytes are transmitted on an avegare every $t_{ss}(p)$ seconds, the steady state throughput in bytes per second of a TCP connection is thus

$$R = \frac{dMSS}{t_{ss}(p)} = \frac{MSS}{t_{ss}(p)p} \qquad (45)$$

We note that though the expression for the steady state throughput does not have the same closed form as those in [18], their numerical values are almost the same. In Fig. 5 we show the results of our model extended to calculate the steady state throughput. The results are for TCP Reno and we compare our results with the model and traces reported in [18]. We see that our results are almost exact over the whole range of loss probabilities.

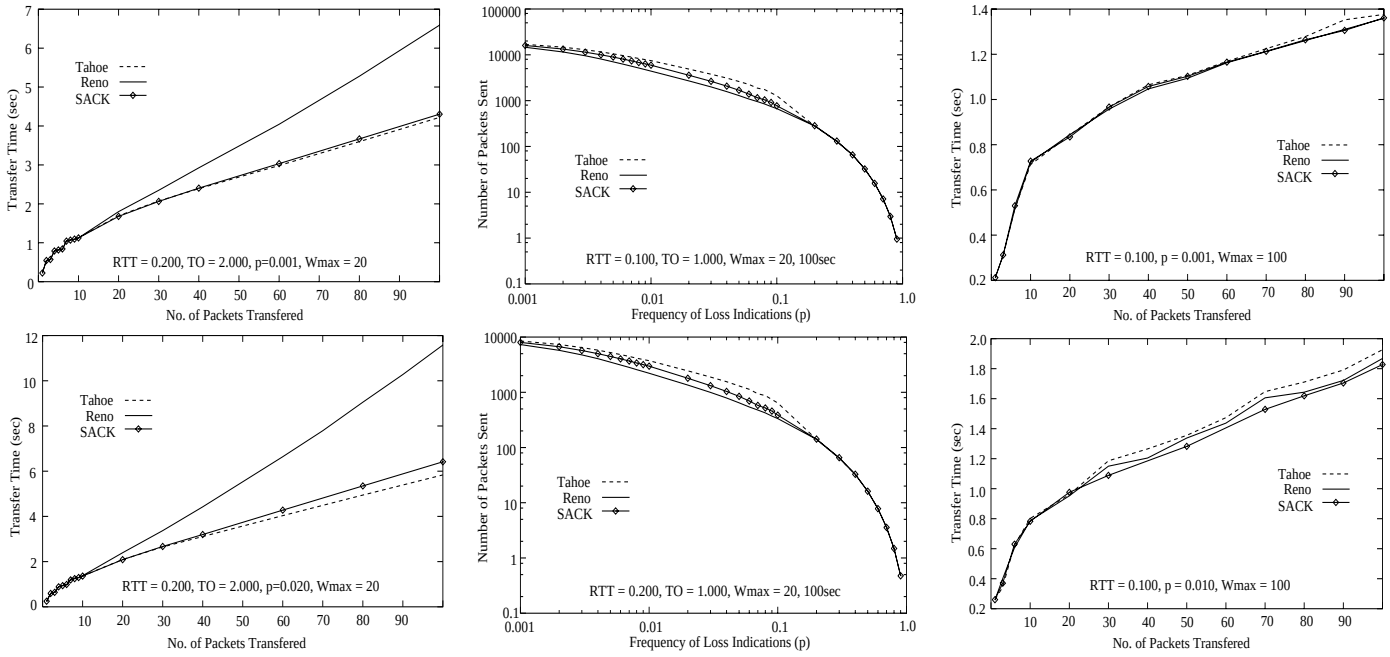
V. Comparison of Tahoe, Reno and SACK

In this section we compare the performance of the three versions of TCP in terms of both their latency and steady-state throughput and identify some conditions and reasons due to which one version may outperform the other. In Fig. 6(a) and 6(b) we plot the expected transfer times and the steady state throughputs of the three versions for different loss probabilities, transfer sizes and RTTs. We note that in all cases, Reno performs worse than the others which is due to the fact that it resorts to timeouts in the presence of multiple losses. The result of importance here is that Tahoe outperforms SACK in the presence of correlated losses. This is explained by considering the fact that with correlated losses, the probability that a loss event leads to the loss of more than half of the packets of that round is close to 0.5. Thus SACK flows timeout quite frequently, degrading their performance. However, Tahoe has a very conservative retransmission policy and assumes that all outstanding packets following a lost packet are also lost. The immediate retransmission of all the losses in the slow start mode leads to considerable savings and more than makes up for any unnecessary retransmissions.

The difference in the performance of these versions of TCP under the influence of a different loss model is highlighted in Fig. 6(c) which shows the simulation results of the transfer times using *ns*. For these simulations, we used an independent loss model where all packet losses are assumed to be independent and there is no correlation between the losses in the same round. This scenario is more likely for queueing disciplines like RED. With independent losses, SACK performs better than both Tahoe and Reno while Reno performs better than Tahoe. The examples considered in [5] which show similar results also have independent losses. SACK performs better with independent losses as now the probability of the loss of more than half the packets from a window is reduced enormously.

VI. Conclusion and Discussions

In this paper we presented analytic models for estimating the latencies and the steady-state throughput of TCP Tahoe, Reno and SACK. While TCP Reno has been modeled extensively, no models existed for estimating the latency of TCP Tahoe and SACK. Additional, the estimates of the transfer times predicted by our TCP Reno model more accurate than existing models (in the range of 15-



(a) Correlated losses (model) (b) Correlated Losses (model) (c) Independent Losses (sim.)

Fig. 6. Transfer times and steady state throughput for Tahoe, Reno and SACK for correlated and independent losses.

20%), particularly for short transfers less than 10KB while for larger flows the difference are not significant. This can be attributed to the better modeling of the timeouts experienced by a TCP flow as well as a more accurate model for the evolution of $cwnd$ during slow start and the delayed ACKs. Our models are based on the assumption of correlated or bursty losses currently prevalent in the Internet due to the dominance of droptail queues in the routers. The models were validated using both simulations as well as traces of TCP transfers over the Internet.

TCP SACK is specially geared towards providing smaller retransmission times in the presence of multiple losses from the same window. However, we show that with correlated losses losses SACK is unable to provide adequate protection against the occurrence of timeouts. If a SACK flow with a current window of $cwnd$ loses $2 + cwnd/2$ packets then current implementations of SACK will timeout. Hence with correlated losses, TCP Tahoe can perform better than SACK because of its conservative retransmission policy. Though Tahoe may unnecessarily retransmit some correctly received packets, this is more than made up the avoidance of timeouts through the direct retransmission of all the packets which are outstanding at the instant the loss is detected.

It is also of interest to note that the performance of the three versions of TCP changes when an independent loss scenario (unrealistic in current Internet scenarios) is used. With independent losses, the probability of multiple losses from the same window reduces and SACK is able to recover the losses without going into a timeout in most cases. Thus in these cases SACK perform better than both Tahoe and Reno while Reno performs better that Tahoe. Since independent loss models do not reflect reality, it is apparent that such assumptions will lead to incorrect conclusions. These observations underline the fact that buffer management policies can make significant differences in protocol performance.

Though there are various possibilities to provide enhanced robustness against timeout to SACK, the most apparent one is to change the implementation of `pipe` to allow retransmission of lost packets on the receipt of partial ACKs even though `pipe` may be less than $cwnd$. Since SACK times out in the presence of heavy losses compared to the window size (where the losses need not be back to back) this change would eliminate timeouts in these circumstances also. Also, with droptail queues here to stay in the foreseeable future, providing protection against bursty losses is necessary to ensure the benefits promised by SACK.

REFERENCES

[1] R. Braden and V. Jacobson, “TCP extensions for long delay paths,” *RFC 1072*, October 1988.

[2] R. Braden, V. Jacobson and L. Zhang, “TCP extensions for high-speed paths,” *RFC 1185*, October 1990.

[3] N. Cardwell, S. Savage and T. Anderson, “Modeling TCP latency,”

Proceedings of IEEE INFOCOM, pp. 1742-1751, Tel Aviv, Israel, March 2000.

[4] C. Casetti and M. Meo, "A New Approach to Model the Stationary Behavior of TCP Connections," *Proceedings of IEEE INFOCOM*, pp. 367-375, Tel Aviv, Israel, March 2000.

[5] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *Computer Communications Review*, vol. 26, no. 3, pp. 5-21, July 1996.

[6] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky and A. Romanow, "An extension to the Selective Acknowledgment (SACK) option for TCP," Internet draft draft-floyd-sack-00.txt, August 1999.

[7] J. Heidemann, K. Obraczka and J. Touch, "Modeling the performance of HTTP over several transfer protocols," *IEEE/ACM Trans. on Networking*, vol. 5, no. 5, pp. 616-630, Oct 1997.

[8] V. Jacobson, "Congestion avoidance and control," *Proceedings of ACM SIGCOMM*, pp. 314-329, Stanford, CA, August 1988.

[9] V. Jacobson, "Modified TCP congestion avoidance algorithm," Email to the end2end-interest mailing list, `ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt`.

[10] A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link," *IEEE/ACM Trans. on Networking*, vol. 6, no. 4, pp. 485-498, Aug 1997.

[11] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. on Networking*, vol. 5, no. 3, pp 336-350, Jun 1997.

[12] S. H. Low, L. L. Peterson and L. Wang, "Understanding TCP Vegas: A duality model," *Proceedings os ACM SIGMETRICS*, pp. 226-235, Boston, MA, June 2001.

[13] M. Mathis, J. Semke, J. Mahdavi and T. Ott, "The macroscopic behavior of the TCP congestion Avoidance Algorithm," *Computer Communications Review*, vol. 27, no. 3, pp 67-82, July 1997.

[14] M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control," *Proceedings of ACM SIGCOMM*, pp. 281-291, Stanford, CA, August, 1996.

[15] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP selective acknowledgment options," *RFC 2018*, April 1996.

[16] R. Morris, "Scalable TCP congestion control," *Proceedings of IEEE INFOCOM*, pp. 1176-1183, Tel-Aviv, Israel, March 2000.

[17] T. Ott, J.H.B. Kemperman and M. Mathis, "The stationary behavior of ideal TCP congestion avoidance," Unpublished Manuscript, Aug 1996. `ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps`

[18] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Trans. on Networking*, vol. 8, no. 2, pp. 133-145, April 2000.

[19] B. Sikdar, S. Kalyanaraman and K. S. Vastola, "An integrated model for the latency and steady state throughput of TCP connections," *Performance Evaluation*, vol. 46, no. 2-3, pp. 139-154, September 2001.

[20] W. R. Stevens, "TCP/IP illustrated volume 1," Addison Wesley, 1994.