Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link

Anurag Kumar, *Senior Member IEEE*

Abstract—**We use a stochastic model to study the throughput performance of various versions of TCP (Tahoe (including its older version that we call OldTahoe), Reno, and NewReno) in the presence of random losses on a wireless link in a local network. We model the cyclic evolution of TCP, each cycle starting at the epoch at which recovery starts from the losses in the previous cycle. TCP throughput is computed as the reward rate in a certain Markov renewal-reward process.**

Our model allows us to study the performance implications of various protocol features, such as, *fast-retransmit* **and** *fast recovery*. **We show the impact of** *coarse timeouts*. **In the local network environment, the key issue is to avoid a coarse timeout after a loss occurs. We show the effect of reducing the** *number of duplicate acknowlegements* **for triggering a fast-retransmit. A large coarse timeout granularity seriously affects the performance of TCP, and the various protocol versions differ in their ability to avoid a coarse timeout when random loss occurs; we quantify these differences. As observed in simulations by other researchers, we show that, for large packet loss probabilities, TCP-Reno performs no better, or worse, than TCP-Tahoe. TCP-NewReno is a considerable improvement over TCP-Tahoe, and reducing the fast retransmit threshold from 3 to 1 yields a large gain in throughput; this is similar to one of the modifications in the recent TCP-Vegas proposal. We explain some of these observations in terms of the variation of fast-recovery probabilities with packet loss probability. Finally, we show that the results of our analysis compare well with a simulation that uses actual TCP code.**

I. Introduction

It is well known that TCP (Transport Control Protocol, the transport layer protocol in the Internet) reacts to all packet losses as if they were caused by congestion; i.e., by stalling for a long timeout period and then dropping its transmission window. Thus, over a cellular wireless channel, in which there can be losses due to link errors and channel handovers, TCP can exhibit very poor performance. Several modifications have been proposed to the TCP loss recovery and congestion control mechanism to improve data throughput in the event of random loss; a simulation study of these has been reported in [6]. To study the performance of TCP over wireless channels, several researchers have recently used experimental test-beds to understand TCP behaviour, and to propose and evaluate possible solutions; see, for example, [1], [2], [4], [8].

Our work, reported here, belongs to a line of research that attempts to develop detailed analytical models of the TCP protocol, in an effort to predict the performance of the various versions that are being proposed. Such modelling and analysis can also be used as a test-bed for evaluating other variations, so that experimentation can be done for the few that are promising. The analysis effort also reveals the reasons for which the various effects are observed in the experimental work. The models we develop are analytical and parametric, so that we can quickly answer "what-if" questions, or evaluate the effect of modifying a particular protocol, or network parameter.

In this paper, we develop models and analyses for studying the bulk throughput of four versions of TCP; namely, OldTahoe (the original protocol from [7]), Tahoe, Reno, and NewReno [15], [6]. We assume a local network scenario, in which a host on a local area network (LAN) is transmitting bulk data to a mobile host connected to the LAN by a wireless link. Our models incorporate important aspects, such as, *slow-start* and *congestion avoidance*, *coarse timers*, *fast-retransmit*, and *fast recovery*. Assuming an uncorrelated random packet loss model on the wireless link, we obtain the data throughput as a function of the packet error probability. Our results show how the throughput degrades, for each version, with increasing packet loss probability. For a given loss probability, our results quantify the performance improvement provided by each version. We show that our analysis provides accurate quantitative evaluation by comparing its results with a simulation based on actual TCP code.

Prior research closest to our work is that of Mishra et al [12], and Lakshman and Madhow [11]; additional, related references are also given in these papers. Misra et al only analyse the original protocol ([7]), and provide some supporting simulation and experimental results. They observe a cyclical structure in the TCP transmission process, and identify, and analyse, the Markov chain of congestion window sizes at loss instants. Both of these elements are key to our analysis as well, but we analyse the newer protocols with the fast-retransmit feature. Lakshman and Madhow consider OldTahoe and Reno, and model and analyse one or more TCP connections whose flow is constrained by a common bottleneck link. The link has a finite buffer, and the bandwidth (i.e., the bottleneck bandwidth) round-trip-delay product for each connection is large. There is thus the issue of buffer overflow at the bottleneck link. These authors study buffer sizing to obtain high TCP throughputs, and obtain an approximation to the random loss probability so that random loss does not constrain throughput. They do not, however, include coarse timeouts in their analysis; avoidance of coarse timeouts is, however, the key issue in

Manuscript received December 17, 1996. Approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Prof. Udaya Shankar. This work was done while the author was on sabbatical at the Wireless Information Networks Lab (WINLAB), Rutgers University, Piscataway, N.J., 08854.

Permanent Affiliation: Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore 560 012, India; e-mail: anurag@ece.iisc.ernet.in

the local network environment, and the issue of keeping the round-trip "pipe" full is not as significant as in the wide-area situation. Thus, they do not quantify the benefit of the fast-retransmit feature in reducing the frequency of coarse timeouts.

Our work has been motivated by the experimental work reported in [1] and [2]. These papers study various modifications to the TCP protocol and report experimental results on the improvements in performance observed, when TCP is used in situations such as that shown in Figure 1. In [6] the authors present fragments of simulation sample paths that demonstrate how the various TCP versions differ in the ways they recover from random packet losses. Our stochastic model captures the probabilities of the various events occurring, and hence allows us to quantify how these different responses to loss events affect the long-run throughput of the protocols.

As in [12] and [11], our analysis is also based on a study of the cyclical evolution of the TCP transmission process. Working with a Markov model, we are able to analyse the congestion window process, as well as the duration of the cycle times. Thus we can compute the long run average bulk throughput of the protocol. The newer versions of TCP, namely, Tahoe, Reno and NewReno attempt to avoid coarse timeouts by trying to do a fast retransmit and fast recovery. Working with the complete distribution of the congestion window process at loss instants allows us to obtain the probability that fast recovery succeeds. Unlike [11], our first attempt is to analyse the model as exactly as possible; since our assumptions allow us to work with Markov models throughout, we are able to capture many more details, and the analysis is relatively straight-forward. Our model includes the duplicate acknowledgement threshold used by the fast-retransmit feature, and we can study the effect of reducing this threshold. A threshold of 1 is similar to one of the modifications to Reno proposed in the TCP-Vegas proposals (see [3]).

This paper is organised as follows. In Section II we briefly describe the data transfer part of the TCP protocol, highlighting the various aspects that are important to model. In Sections III and IV we describe the network scenario, and the modelling assumptions/simplifications. In Section V, the detailed mathematical analysis is presented. Section VI contains the numerical results and their discussion. Those primarily interested in the results can skip Section V; the notation and terminology used in Section VI are defined before Section V. Finally, some conclusions are summarised in Section VII.

II. The TCP Protocol

We model only the data transfer part of TCP. Details of the TCP protocol can be found in the various Internet RFCs; see also [15]. The versions of the TCP protocol that we model and analyse all assume the same receiver process.

The TCP receiver: accepts packets out of sequence number order, buffers them in a TCP buffer, and delivers them to its TCP user in sequence. Since the receiver has a finite resequencing buffer, it advertises a maximum window size, W_{max}, at connection setup time, and the transmitter ensures that there is never more than this amount of unacknowledged data outstanding. We assume that the user application at the TCP receiver can accept packets as soon as the receiver can offer them in sequence, and hence the receiver buffer constraint is always just W_{max}. The receiver returns an *acknowledgement (ack)* for every good packet that it receives. An ack packet that acknowledges the first receipt of an error-free, in-sequence packet will be called a *first* ack. The acks are *cumulative*, i.e., an ack carrying the sequence number n acknowledges all data up to, and including, the sequence number $n-1$. If there is data in the resequencing buffer, the acks from the receiver will carry the *next expected* packet number, which is the first among the packets required to complete the sequence of packets in the sequencing buffer. Thus, if a packet is lost (after a long sequence of good packets), then the transmitter keeps getting acks with the sequence number of the first packet lost, if some packets transmitted after the lost packet do succeed in reaching the receiver. These are called *duplicate acks*.

The TCP transmitter: At all times t the transmitter maintains the following variables for each connection:

$A(t) =$ the *lower window edge*; all data numbered upto and including $A(t)-1$ has been transmitted and acked. $A(t)$ is nondecreasing; the receipt of an ack with sequence number $n > A(t)$ causes $A(t)$ to jump to n.

$W(t) =$ the *congestion window*. The transmitter can send packets with the sequence numbers $n, A(t) \leq n < A(t) + W(t)$. $W(t) \leq W_{max}$; $W(t)$ increases or decreases as described below.

$W_{th}(t) =$ the *slow-start threshold*; $W_{th}(t)$ controls the increments in $W(t)$ as described below.

Retransmission time-out: The transmitter measures the round-trip times of *some* of the packets that it has transmitted and received ackowledgements for. These measurements are used to obtain a running estimate of the packet *round-trip time (rtt)* on the connection. Each time a new packet is transmitted, the transmitter starts a timeout timer and *resets* the already running timeout timer, if any; i.e., there is a timeout only for the last transmitted packet. The timer is set for a *retransmission time-out (rto)* value that is derived from the rtt estimation procedure. The TCP transmitter process measures time and sets timeouts only in multiples of a *timer granularity*; for example, BSD based systems have a timer granularity of 500ms. Further, there is a minimum timeout duration in most implementations; e.g., 2 timer ticks in BSD, implying an average minimum timeout of 750ms. We will see, in the analysis, that *coarse timers* have a significant impact on TCP performance. For details on rtt estimation, and the setting of rto values, see [5] or [15].

Window adaptation and time-out based recovery: The basic algorithm is common to all TCP versions, and was originally developed and proposed by Van Jacobson [7]; our description follows that of [11]. The normal evolution of the processes $A(t)$, $W(t)$, and $W_{th}(t)$ is triggered by first acks (see definition above) and timeouts as follows.

1. *Slow start:* If $W(t) < W_{th}(t)$, each first ack causes $W(t)$ to be incremented by 1.
2. *Congestion Avoidance:* If $W(t) \geq W_{th}(t)$, each first ack causes $W(t)$ to be incremented by $\frac{1}{W(t)}$.
3. *Timeout* at epoch t causes $W(t^+)$ to be set to 1, $W_{th}(t^+)$ to be set to $\left\lceil \frac{W(t)}{2} \right\rceil$, and retransmissions to begin from $A(t)$.

Packet loss recovery: If a packet is lost, $A(t)$ and $W(t)$ will continue to be incremented until the first ack for the packet just before the lost packet is received. For a particular loss instance, let their final values be denoted by A and M, respectively; we will call M a *loss window*. Then the transmitter will continue to send packets upto the sequence number $A + M - 1$. If some of the packets sent after the lost packet get through, they will result in duplicate acks, all carrying the sequence number A. The last packet transmitted (i.e., $A + M - 1$) will have a rto associated with it.

The TCP versions differ in the way they recover from loss. We provide some details here; in Section IV we will describe how we have modelled these recovery procedures.

TCP OldTahoe: (*timeout recovery [7]*) The transmitter continues sending till packet number $A + M - 1$ and then waits for a coarse timeout.

TCP Tahoe: (*fast retransmit [6]*) There is a transmitter parameter K, a small positive integer; typically $K = 3$. If the transmitter receives the Kth duplicate ack at time t (before the timer expires), then the transmitter behaves *as if* a timeout has occurred and begins retransmission, with $W(t^+)$ and $W_{th}(t^+)$ as given by the basic algorithm.

TCP Reno: (*fast retransmit; fast (but conservative) recovery [15]*). Fast-retransmit is implemented, as in TCP Tahoe, but the subsequent recovery phase is different. Suppose the Kth duplicate ack is received at the epoch t_0. Loss recovery then starts. Recalling the definitions of A and M above, the transmitter sets

$$W(t_0^+) = \left\lceil \frac{M}{2} \right\rceil + K \text{ and } W_{th}(t_0^+) = \left\lceil \frac{M}{2} \right\rceil$$

The addition of K takes care of the fact that K more packets have successfully left the network. The Reno transmitter then retransmits *only* the packet with sequence number A, the Reno philosophy being conservative – if only one packet is lost then this retransmission will produce an ack for all the other packets, whereas if more packets are lost we had better be sure that we are not really experiencing congestion loss. For the ith duplicate ack received, at say t_{ack_i}, until recovery completes

$$W(t_{ack_i}^+) = W(t_{ack_i}) + 1$$

We explain the rest of the Reno recovery via an example, where we take $K = 3$. Suppose $A = 15$, $M = 16$ and packet 15 is lost. The transmitter continues sending packets $16, 17, 18, \ldots, 30$; suppose packet 20 is also lost. The receiver returns acks (all asking for packet 15) for packets $16, 17, 18, 19, 21, \ldots, 29, 30$. Note that the ack for packet 14 would have been the *first* ack asking for packet 15. When the ack for packet 18 is received (i.e., the 3rd duplicate ack) the transmitter sets $W = (M/2) + 3 = 11$, and $W_{th} = (M/2) = 8$. A is still 15; thus packet 15 is retransmitted. Meanwhile acks for packets $19, 21, \ldots, 30$ are also received, and W grows to $11 + 11 = 22$. Since $A = 15$, with $W = 22$, the transmitter is allowed to send packets 15 to 36; hence retransmission of packet 15 is followed by transmission of packets 31 to 36. Receipt of packet 15 results in a first ack asking for packet 20[1] (thus first-acking packets 15,16,17,18,19) and A jumps to 20. If packets 31 to 36 succeed in getting through then 3 duplicate acks asking for packet 20 also are obtained, and 20 is retransmitted. This results in a first ack that covers all the packets upto packet number 36. At this time t_1, the congestion window is reset as follows, and a new transmission cycle starts

$$W(t_1^+) = \left\lceil \frac{M}{2} \right\rceil (= W_{th}(t_1^+))$$

Thus Reno slowly recovers the lost packets, and there is a chance that, owing to insufficient duplicate acks, the recovery stalls and a timeout has to be waited for. After a timeout, the basic timeout recovery algorithm is applied.

TCP NewReno: (*fast-retransmit; fast-recovery [6]*) When K duplicate acks are received, the first lost packet is resent, but, unlike Reno, upon receipt of a partial ack after the first retransmission, the next lost packet (as indicated by the partial ack number) is retransmitted. Thus, after waiting for the first K duplicate acks, the remaining lost packets are recovered in as many round-trip times. If less than K duplicate acks are received then a timeout is inevitable. Consider the following example to see the difference with Reno. Suppose that after a loss $A = 7$ and $M = 8$, packets $7, 8, \ldots, 14$ are sent, and packets 7,8 and 11 are lost. The transmitter receives 3 duplicate acks for packets 9,10, and 12 (asking for packet 7). Fast-restransmit is done (the same as in Reno); i.e., $W = 4 + 3 = 7$, $W_{th} = 4$, and packet 7 is sent. The 2 acks for packets 13 and 14 cause W to become 9. Assuming that packet 7 now succeeds, its ack (the first ack asking for 8) would cause A to become 8; the transmitter can now send packets 15 and 16 also. NewReno would now resend packet 8, whereas Reno would wait for 3 duplicate acks; these cannot come since only 2 more packets have been sent after the retransmission of packet 7. Thus, in case of multiple losses, Reno has a higher probability of resorting to a coarse time-out.

Spurious retransmissions: Consider TCP OldTahoe. The retransmission of the first lost packet may result in an ack that acknowledges all the packets until the next packet lost in the loss window. This would cause the lower window edge $A(t)$ to advance, the congestion window would be increased to 2 and the next lost packet and its successor

[1] This is called a *partial ack* since all the packets transmitted in the original loss window were not acked.

would be transmitted, *even if this successor packet had got through in its first transmission.* Thus some of the good packets in the loss window may be retransmitted when retransmission starts; this phenomenon can be seen in the sample path fragments shown in [6].

III. Network Scenario and Model

A. The Scenario

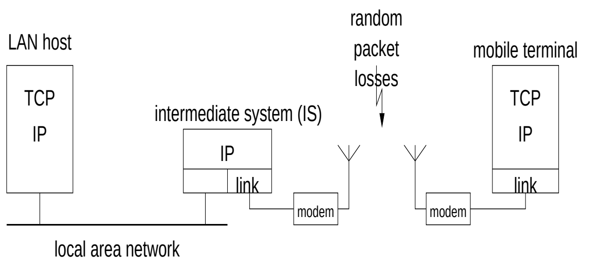

Fig. 1. A LAN host with a TCP connection to a mobile host.

The scenario (Figure 1) is motivated by the many recent experimental studies of TCP performance over wireless mobile links; see, for example, [1], [2], [4], [8]. In this paper, we do not consider the additional issue of mobility (see, e.g., [4]); hence we refer to the wireless link as simply a "lossy" link.

B. The Network Model

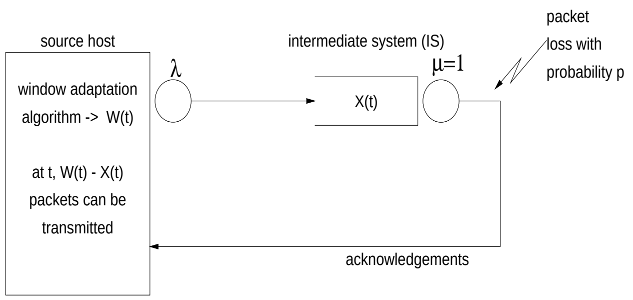

Fig. 2. Model for the TCP connection. $X(t)$ is the number of TCP packets queued at the IS at time t.

Figure 2 shows the schematic of a model for the system shown in Figure 1. *We model only one direction of flow of packets:* from the LAN host to the mobile terminal. Since this is a local networking situation, *propagation delays are assumed to be negligible.* The transmission time of a TCP packet from the LAN host (resp., the lossy link) is assumed to be exponentially distributed with mean λ^{-1} (resp., μ^{-1}); we take $\mu = 1$; i.e., time is normalised to the mean packet transmission time on the lossy link. During a bulk transfer over a TCP connection, we would expect that the packets would predominantly be of a fixed length. Note, however, that a multiple access protocol operates on the LAN as well as on the lossy link (in particular, for CDMA (Code Division Multiple Access) or CDPD (Cellular Digital Packet Data) (see [13, page 455 and 519])). Thus, the randomness in packet transmission times in our model can be taken to account for the variability in the time taken to transmit a head-of-the-line packet at the transmitter queues. The service time at the source host model can also be used to model the protocol processing time at the TCP transmitter. The exponential assumption yields Markov processes, and hence facilitates the analysis. The exponential assumption is used only for the computation of certain mean cycle times; we have found (see [9]) that fairly crude approximations for these suffice.

The queue at the intermediate system (IS) is the queue of packets waiting to be transmitted onto the lossy link. Since we are primarily interested in the effect of random losses in the wireless link, we do not study buffer overflows at the IS; i.e., we assume that the buffer at the IS is large enough to hold the largest possible window worth of packets. We assume that acknowledgements (acks) from the mobile receiver arrive instantaneously to the LAN host. Since ack packets are relatively much smaller than data packets (40 byte acks vs. 560 to 1500 byte packets), this is a reasonable assumption.

Packet loss model: With probability p a packet transmitted by the IS transmitter is lost; the losses are independent, i.e., we have a Bernoulli loss model. Since ack packets are much smaller (40 bytes), they have a much smaller loss probability and we ignore ack loss in this analysis. Typically, wireless links are subject to fading phenomena, which results in burst losses, rather than the uncorrelated losses that we have assumed here. This important extension to the present analysis has been done in [10].

IV. Modelling Assumptions for the TCP Versions

In order to obtain analytically tractable models, we have made some simplifications that we now discuss. It is important to note that these simplifications do not alter the essential nature of the protocols. We obtain parametrised, analytically tractable models that can quickly yield quantitative comparisons between the various protocol versions, and can be used for evaluating the effects of the various parameters.

A. General Assumptions

Model for congestion avoidance: For the calculation of the mean durations of certain transient periods in our analysis, we have modelled the congestion avoidance phase probabilistically: i.e., if $W(t) \geq W_{th}(t)$, each ack from the receiver causes $W(t)$ to be incremented by 1 with *probability* $\frac{1}{W(t)}$; thus the expected window increase for each acknowledgement is $\frac{1}{W(t)}$. Until a loss occurs, let $X(t) (\leq W(t))$ denote the number of packets in the IS queue. Then the LAN host transmitter can send $W(t) - X(t)$ packets at the rate λ. With our assumptions (i.e., exponential service times and probabilistic window increment) $(X(t), W(t))$ is a Markov chain, and we can use a mean absorption time analysis to obtain the mean of the time till a loss occurs. See the beginning of Section V-A for comments on the probabilistic window increment assumption and how it affects our results.

Model for retransmission timeouts: As described in Section II above, the retransmission timeout is based on measured round-trip times of some of the transmitted packets. In our model, we approximately obtain the retransmission timeout as follows. After a packet loss there can be some packets still in the IS buffer, and the transmitter may still be able to send some more packets. The transmitter starts a retransmission timer after sending the last packet in the current window; *ideally* the timeout should be set for the epoch at which the ack for this last transmitted packet is expected. Suppose that at the loss epoch i packets remained in the IS queue, and j more packets could be sent from the TCP transmitter. Then we identify the "exact" timeout epoch as the epoch at which the last of these $i+j$ packets is transmitted out of the IS queue; this would have been the epoch at which the ack for the last transmitted packet would have reached the TCP transmitter. We find the mean time from the packet loss instant until the exact timeout epoch. The coarse timeout instant is obtained by adding half the timer granularity (rto_gran) to the exact timeout epoch. The actual timeout is taken as the maximum of the coarse timeout and the minimum timeout (rto_min).

Model for exponential timeout-backoff: If the first retransmitted packet is also lost then the coarse timeout is taken to be twice the minimum timeout; this is adequate since the probability of 3 successive timeouts is very small (just .001 for a 10% packet loss probability); note that exponential backoff can become a serious issue if losses are severe and correlated (see, e.g., [8])

B. Models for OldTahoe, Tahoe, Reno, and NewReno

From the protocol description given in Section II, it is clear that there is a cyclical behaviour in the protocols: periods of transmission of good packets (slow-start or congestion avoidance) alternate with periods of loss recovery. Assume that, at $t=0$ the connection starts with $W(0)=1$ and $W_{th}(0)=\lceil \frac{W_{max}}{2} \rceil$. The protocol starts at time 0 in slow start. Let $t_0=0$, and, for $k \geq 1$, let t_k denote the kth epoch at which a new transmission cycle starts, following the recovery phase after a packet loss. For $k \geq 1$, we call the interval $(t_{k-1}, t_k]$ the kth cycle. In the kth cycle, let ℓ_k be the epoch at which the first packet is lost in the cycle (this is an end of a packet transmission epoch from the IS). Further, for $k \geq 1$, let $U_k = W(\ell_k)$ denote the transmitter's window size at which packet loss takes place. Note that, since propagation delays and reverse path transmission delays are being ignored, $W(\ell_k)$ accounts for all the window increments upto and including the last good packet before the lost packet. Thus, with reference to our earlier definition, U_k is the loss window in the kth cycle. We take $U_0 = W_{max}$; this is consistent with the initial conditions assumed above.

TCP OldTahoe and Tahoe: Both protocols start each new cycle in slow-start. Consider the kth cycle. If the first loss occurs at the epoch ℓ_k (see Figure 3), then OldTahoe will recover only after a coarse time-out, which is estimated as described in Section IV; in Figure 3 the coarse time-out epoch is shown as the later t_k.

Tahoe, on the other hand, may succeed in a fast-retransmit. Out of the $U_k - 1$ packets that follow the lost packet, if K succeed, then Tahoe will not wait for a coarse time-out, and will start retransmitting from the first lost packet. But (with a fast LAN transmitter) the TCP transmitter would already have sent the remaining packets in the loss window, and the first retransmission will be queued up behind the $U_k - 1$ packets that are sent after the lost packet. Thus the retransmitted packet will not get transmitted on the lossy link until the last of these $U_k - 1$ packets is transmitted on the lossy link. But this is just the so-called exact time-out epoch that we have described in Section IV-A above, and is shown as the earlier t_k in Figure 3. Note that, owing to the Bernoulli loss model, the probability of fast-retransmit depends only on U_k.

For both OldTahoe and Tahoe, the value of U_k determines the values of $W(t_k^+)$ and $W_{th}(t_k^+)$, as follows,

$$W(t_k^+) = 1 \text{ and } W_{th}(t_k^+) = \left\lceil \frac{U_k}{2} \right\rceil,$$

and hence determines (probabilistically) the value of U_{k+1}.

TCP Reno and NewReno: Let F_{U_k} be the probability that the packet loss in the kth cycle is followed by a fast recovery. The calculation of F_U is shown in Section V-A.1. Thus, the value of U_k, and whether or not fast-recovery occurs, determines the values of $W(t_k^+)$ and $W_{th}(t_k^+)$, as follows: with probability F_{U_k}

$$W(t_k^+) = W_{th}(t_k^+) = \left\lceil \frac{U_k}{2} \right\rceil,$$

and the next cycle starts with congestion avoidance; with probability $1 - F_{U_k}$,

$$W(t_k^+) = 1 \text{ and } W_{th}(t_k^+) = \left\lceil \frac{U_k}{2} \right\rceil,$$

and the next cycle starts in the slow-start phase. It follows that, for TCP Reno and NewReno too, U_k determines (probabilistically) the value of U_{k+1}. If fast-retransmit occurs, we assume (see the motivation above for doing this for Tahoe) that the next cycle starts at the exact timeout epoch (see Figure 4), otherwise the transmitter times out; we compute the coarse timeout as for OldTahoe. This model for the time to recovery, and the way we calculate the probability of fast recovery, are approximations for Reno and NewReno, but the essential distinctions between Tahoe, Reno and NewReno are captured; i.e., Tahoe may do a fast-retransmit but always recovers by slow-start, whereas Reno and NewReno may both do a fast-retransmit, but fast-recovery occurs with a higher probability for NewReno.

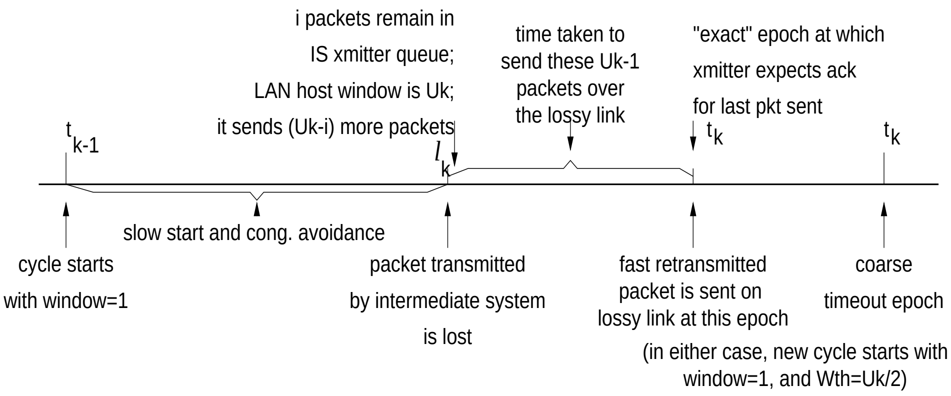

Fig. 3. Events during a TCP-OldTahoe or Tahoe transmission cycle; the next cycle starts at either of the two epochs marked t_k, depending on whether or not fast-retransmit takes place.

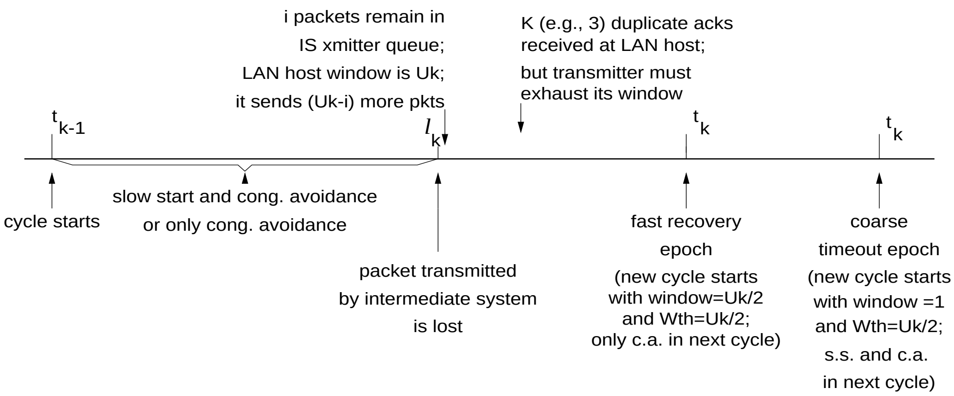

Fig. 4. Events during a TCP Reno or NewReno transmission cycle with fast-recovery; the next cycle starts at either of the two epochs marked t_k, depending on whether or not fast-recovery succeeds.

See the first paragraph of Section V-A.2 for a further discussion of this approximation approach.

From the above observations, it follows that for all the TCP versions the process $\{U_k\}$ is a discrete time Markov chain over the state space $\{1, 2, \ldots, W_{max}\}$. We can obtain the stationary distribution of this chain.

The mean number of packets successfully transmitted in each cycle, before the first lost packet, is just $\frac{1-p}{p}$ (recall that p is the probability of packet loss). When a loss occurs at the lossy link, there would be some packets queued in the lossy link transmitter, and the TCP transmitter at the LAN host will continue sending packets till the congestion window (U_k) is exhausted (even if fast-retransmit is implemented, by the time 3 duplicate acks are received, a fast sender would already have exhausted the window). Some of the packets that are transmitted on the lossy link, *after* the first lost packet, will get through, and will not need to be resent. If U represents the stationary random variable for the process $\{U_k\}$, then the mean number of good packets transmitted, out of the $(U-1)$ sent subsequent to the first loss in a cycle, is $(EU-1)(1-p)$.

Thus, for each TCP version, we have a Markov renewal reward process [16], embedded at the epochs $t_0, t_1, t_2, \ldots$, the reward being the number of good packets put through in each cycle. For $ver \in \{$OldTahoe, Tahoe, Reno, NewReno $\}$, let C_{ver} denote the mean cycle time for a TCP version, and let U_{ver} denote the stationary congestion window process at the loss epochs. Then the throughput T_{ver} is given by

$$T_{ver} = \frac{\frac{1-p}{p} + (1-p)(EU_{ver} - 1)}{C_{ver}} \tag{1}$$

The numerator on the right hand side is a slight overestimate, since some of the good packets transmitted in the loss window may get resent in the next transmission cycle.

V. Analysis of the Models

A. *Detailed Analysis*

There are two ingredients to the analysis: the stationary distribution of the Markov chain $\{U_k\}$, and the mean durations of the various periods constituting a cycle. Between these two it is by far the more important to analyse $\{U_k\}$ exactly, as it is the distribution of the loss window that governs the probability of fast retransmission. We analyse this Markov chain using the exact transition probabilities, and do not assume the probabilistic congestion window increment model mentioned in Section III-B. This approximate model is used only in the computation of the mean cycle times (see Section V-A.2), for which, it turns out (see, [9]) that fairly crude approximations suffice.

A.1 Analysis of the Markov Chain $\{U_k\}$

We first obtain the transition probabilities for the Markov chain $\{U_k\}$. Let $U_k = M, M \in \{1, 2, \ldots, W_{max}\}$; we wish to determine $P\{U_{k+1} = j \mid U_k = M\}$ for $1 \leq j \leq W_{max}$. Define, for $M \in \{1, 2, \ldots, W_{max}\}$,

f_M = the probability that, when loss occurs in a cycle, at a congestion window of M, the next cycle directly starts in the congestion avoidance phase

Thus $1 - f_M$ is the probability that the next cycle begins in the slow-start phase. The various TCP versions are distinguished by the values of $\{f_M, 1 \leq M \leq W_{max}\}$.

We can easily obtain the following transition probability distribution for U_{k+1}, given that $U_k = M, 1 \leq M \leq W_{max}$. Define $\overline{f}_M := 1 - f_M$, and $\overline{p} := 1 - p$. Note that "w.p." denotes "with probability". Given that $U_k = M$ (and $m := \lceil \frac{M}{2} \rceil$), and recalling the window increment behaviour of TCP (starting with a window of 1, in slow start, $j - 1$ packets need to succeed for the window to reach j; $U_{k+1} = m$ if the first $m-1$ packets succeed, and not all of the next m packets succeed; $U_{k+1} = m+1$ if the first $(m-1)+m$ packets succeed, and not all of the next $m+1$ packets succeed, etc.), we have

$$U_{k+1} = \begin{cases} j & \text{w.p. } \overline{f}_M \overline{p}^{j-1} p \\ & \text{for } 1 \leq j \leq m-1 \\ m+k & \text{w.p. } \overline{f}_M \overline{p}^{d(k)}(1 - \overline{p}^{m+k}) + \\ & f_M \overline{p}^{h(k)}(1 - \overline{p}^{m+k}) \\ & \text{for } 0 \leq k \leq (W_{max} - 1 - m) \\ W_{max} & \text{w.p. } \overline{f}_M \overline{p}^{d(W_{max}-m)} + f_M \overline{p}^{h(W_{max}-m)} \end{cases} \tag{2}$$

where

$$\begin{aligned} d(k) &= (m-1) + m + (m+1) + \cdots + (m + (k-1)) \\ &= (k+1)((m-1) + \frac{k}{2}) \\ h(k) &= 0 + m + (m+1) + \cdots + (m + (k-1)) \\ &= d(k) - (m-1) \end{aligned}$$

Observe that the transition probabilities defined above are the "mixture" of two other transition probabilities $P = [p_{i,j}]$ and $Q = [q_{i,j}]$ on the finite state space $\{1, 2, \ldots, W_{max}\}$. P and Q are given as follows (we again denote $\lceil \frac{M}{2} \rceil$ by m, for $M \in \{1, 2, \ldots, W_{max}\}$).

$$p_{M,j} = \begin{cases} \overline{p}^{j-1} p & \text{for } 1 \leq j \leq m-1 \\ \overline{p}^{d(k)}(1 - \overline{p}^{m+k}) & \text{for } j = m+k, \\ & 0 \leq k \leq (W_{max} - 1 - m) \\ \overline{p}^{d(W_{max}-m)} & \text{for } j = W_{max} \end{cases}$$

$$q_{M,j} = \begin{cases} 0 & \text{for } 1 \leq j \leq m-1 \\ \overline{p}^{h(k)}(1 - \overline{p}^{m+k}) & \text{for } j = m+k, \\ & 0 \leq k \leq (W_{max} - 1 - m) \\ \overline{p}^{h(W_{max}-m)} & \text{for } j = W_{max} \end{cases}$$

Note that P corresponds to packet loss followed by slow start, and Q corresponds to packet loss followed by congestion avoidance. Finally, we define the vector

$$\underline{f} = (f_1, f_2, \ldots, f_{W_{max}}),$$

and then the transition probability matrix of $\{U_k\}$ is written as

$$(I - \text{diag}(\underline{f}))P + \text{diag}(\underline{f})Q,$$

where I is the $W_{max} \times W_{max}$ identity matrix.

The matrices P and Q are common to our models of all the TCP versions with the same values of W_{max} and packet error probability p. For the various TCP versions we calculate $\underline{f}$ as shown below.

TCP-OldTahoe and TCP-Tahoe: If packet loss occurs at a congestion window equal to M, then the slow-start threshold is set to $m = \lceil \frac{M}{2} \rceil$, and the congestion window is set to 1. Thus we have, for $1 \leq M \leq W_{max}$,

$$f_M = 0; \tag{3}$$

i.e., each cycle always begins in slow-start.

TCP-Reno: In our model of TCP-Reno, suppose that the first packet loss in a cycle occurs at the congestion window M. Following this packet loss, the TCP transmitter does a fast-retransmit only if it receives K duplicate acks. Clearly, this is impossible if $M - 1 < K$, since the K acks must come out of the $M - 1$ packets that are transmitted following the lost packet. Hence, for $1 \leq M \leq K$,

$$f_M = 0 \tag{4}$$

Next, we suppose that $M - 1 \geq K$, and K of these packets do manage to get through to the receiver, thus generating K duplicate acks. The TCP-Reno transmitter retransmits, however, just the first packet that the receiver is missing, and sets $W_{th} = \lceil \frac{M}{2} \rceil =: m$, and $W = W_{th} + K$. If this is the only lost packet in the window, then upon receipt of this packet the receiver will ack all the packets sent so far by the transmitter, and fast recovery would succeed. If there are multiple losses, however, then the receiver will only send an ack bearing the number of the next lost packet in the window; in this case, coarse timeout will be avoided only if at least K duplicate acks arrive for *the second* lost packet too. Since we have assumed a relatively fast TCP transmitter, after the first loss occurs and K duplicate acks have been received, but before the lost packet has been retransmitted, with a probability close to 1, the remaining $M - 1$ packets in the window would already have been transmitted by the TCP transmitter. Thus when the lost packet is retransmitted, duplicate acks for the next lost packet in the original window can only be generated if at least K more packet transmissions follow the retransmission of the first lost packet.

Recall that after the K duplicate acks have been received at the TCP transmitter, the congestion window $W = m + K$. Now for each additional duplicate ack received (after these K) W is further incremented by 1. Let the number of good packets transmitted (in the orginal window) after the first lost packet be denoted by n. Then, after the TCP transmitter retransmits the first lost packet, it can send at least K more packets if $m+n \geq M+K$. This is because until the first lost packet is correctly received, and acked, the transmitter already has M packets outstanding; the congestion window will have to exceed this window by at least K to enable the transmitter to send K more packets right after retransmitting the first lost packet. Observe that this condition is not necessary, since receipt of the first lost packet would cause the "left edge" of the congestion window to move up to the next lost packet. We, however, assume that, when there are multiple losses, fast recovery succeeds only if the above inequality holds. We also assume that fast recovery succeeds only if at most 2 packets are lost in the loss window.

With these approximations we now continue with the calculation of f_M, for $M > K$. Since $n \leq M-1$, the above inequality cannot be satisfied if $m+(M-1) < M+K$, and will be satisfied only if there are no multiple losses if $m+(M-1) = M+K$. Observe that $m+(M-1) \leq M+K$ is the same as $M \leq 2(K+1)$. Hence, for $K+1 \leq M \leq 2(K+1)$, all packets following the first loss must be good, and the retransmission must be good; i.e,

$$f_M = (1-p)^{M-1} \cdot (1-p) \tag{5}$$

Finally, we are left with the range $2(K+1)+1 \leq M \leq W_{max}$, and for this we get

$$\begin{aligned} f_M &= (1-p)^{M-1} \cdot (1-p) + \\ &\quad Pr\{\text{exactly 1 more loss in } M-1 \text{ packets}\} \cdot \\ &\quad Pr\{\text{all the } (M-2)-(M-m) \text{ packets} \\ &\quad \text{subsequently sent are good}\} \cdot (1-p)^2 \\ &= (1-p)^{M-1} \cdot (1-p) + \\ &\quad (M-1)p(1-p)^{M-2} \cdot (1-p)^{m-2} \cdot (1-p)^2 \quad (6) \end{aligned}$$

The above analysis accounts for the probability of fast recovery from upto 2 losses. Observe that, if the loss probability p is large then the window M when loss occurs is small, and even if multiple losses occur the probability of being able to generate enough duplicate acks is small; on the other hand if p is small then it is unlikely that there are more than 2 losses in the window. Hence the above approximation is adequate.

Note that our model of TCP-Reno assumes that, following fast recovery, congestion avoidance starts with $W = W_{th} = \lceil \frac{M}{2} \rceil$, where M is the congestion window when the first loss in the previous cycle occurred.

TCP-NewReno: The transmitter does a fast-retransmit, and fast recovery succeeds if at least K duplicate acks are received following the first packet loss in a window. Hence, for $1 \leq M \leq K$,

$$f_M = 0 \tag{7}$$

and, for $K+1 \leq M \leq W_{max}$,

$$f_M = \sum_{n=K}^{M-1} \binom{M-1}{n} (1-p)^n p^{M-1-n} \tag{8}$$

This completes all the ingredients for the generation of the transition probability matrix of the Markov chain $\{U_k\}$. The stationary distribution can be obtained by any of the many standard techniques. Denote the stationary distribution by

$$u_M, 1 \leq M \leq W_{max}$$

A.2 Calculation of the Mean Cycle Times

As discussed earlier, the kth cycle $(t_{k-1}, t_k]$ comprises two parts; see Figures 3 and 4. The first part is the interval $(t_{k-1}, l_k]$ during which all transmissions on the lossy link are good; the second part is the interval $(l_k, t_k]$, which is the *recovery phase*, during which the transmitter learns about the loss, either by receiving duplicate acks or by a timeout. This recovery period will have an initial part (that will depend on the number of packets outstanding at the time of loss and the number of them that are lost), and then a coarse timeout part if the transmitter is unable to execute fast recovery. At t_k the next cycle starts with a congestion window and transmission phase (slow-start or congestion avoidance) that is determined as described earlier. In the case of TCP-Tahoe, Reno and NewReno, if K duplicate acks are generated, the recovery phase can be quite complex as the transmitter will attempt to resend the lost packets, and will get partial acks that will change the congestion window in a complicated way. We have attempted to capture some of this complexity in the analysis of the Markov chain $\{U_k\}$ above. Exact analysis of the initial period of the recovery phase, however, does not seem to be that important, so long as we capture the portion of the recovery time due to a coarse timeout. When the packet loss probability is small then the preloss phase of the transmission cycle is large; also if only one loss occurs then recovery does start at the exact timeout epoch, as assumed. If the packet loss probability is large, then the window does not grow to large values, and, since the initial part of the recovery period depends on how large the window at the time of first loss is, again this part does not have a large impact on the final throughput calculation. Also with a large loss probability the probability of fast recovery is small, and most likely recovery takes place only after a timeout.

We define, for $1 \leq j \leq W_{max}$, and $0 \leq i \leq j$,

$\gamma_{i,j}^{(m)}$ = mean cycle time when the cycle starts with a congestion window j, the slow-start threshold is m, and i packets are queued at the transmitter of the lossy link

The calculation of $\gamma_{i,j}^{(m)}$ = is different for each of OldTahoe, Tahoe, Reno and NewReno; the details are provided in Appendix I. Recall that $(u_M, 1 \leq M \leq W_{max})$ denotes the stationary distribution of the Markov chain $\{U_k\}$.

TCP-OldTahoe and Tahoe: The TCP transmitter always starts the each cycle with the window 1 and in the slow-start phase. It follows that the mean cycle time is given by

$$C_{OldTahoe \text{ or } Tahoe} = \sum_{M=1}^{W_{max}} u_M \gamma_{0,1}^{(\lceil \frac{M}{2} \rceil)}$$

TCP-Reno and NewReno: If the first loss occurs in a cycle at a window of M, and $m := \lceil \frac{M}{2} \rceil$, then the next cycle will start with a window of 1 or of m, and a slow-start threshold m, the former case occuring w.p. $1 - f_M$, and the latter w.p. f_M. It follows that

$$C_{Reno \text{ or } NewReno} = \sum_{M=1}^{W_{max}} u_M \left(f_M \gamma_{0,\lceil \frac{M}{2} \rceil}^{(\lceil \frac{M}{2} \rceil)} + (1 - f_M) \gamma_{0,1}^{(\lceil \frac{M}{2} \rceil)} \right)$$

VI. Numerical Results and Discussion

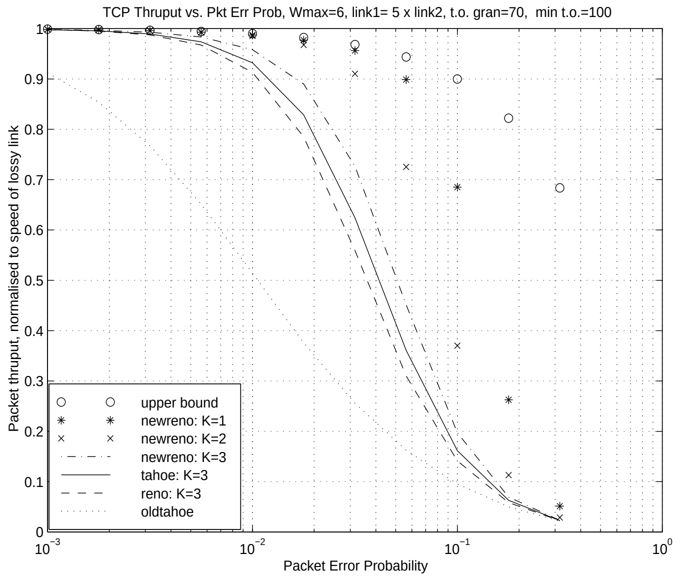

Fig. 5. Throughput of versions of TCP vs. packet loss probability; ($W_{max} = 6$, $\lambda = 5\mu$, K is the fast-retransmit threshold).

Figures 5, 6, and 7 show calculations from Equation 1, using the detailed analyses developed above. Recall that all times are normalised to the mean packet transmission times at the lossy link. In these figures timeout granularity is 70, minimum timeout is 100, and the fast retransmit threshold $K \in \{1, 2, 3\}$; Figure 5 has $W_{max} = 6, \lambda = 5$, Figure 6 has $W_{max} = 24, \lambda = 5$, and Figure 7 has $W_{max} = 48, \lambda = 5$. Let us first put these numbers in perspective. With reference to the scenario in Figure 1, consider a wireless link of effective bit-rate 1.5Mbps; with 1400bytes TCP segments, this implies a packet transmission time of 7.5ms on the wireless link. Suppose that the LAN host, which is on a 10Mbps Ethernet, can source data at the rate of 7.5Mbps (5 times the wireless link rate); the reduction in rate being due to TCP processing. Assuming the BSD implementation of TCP, the timeout granularity is 500ms; i.e, about 70 times the mean packet service time. The minimum timeout is 2

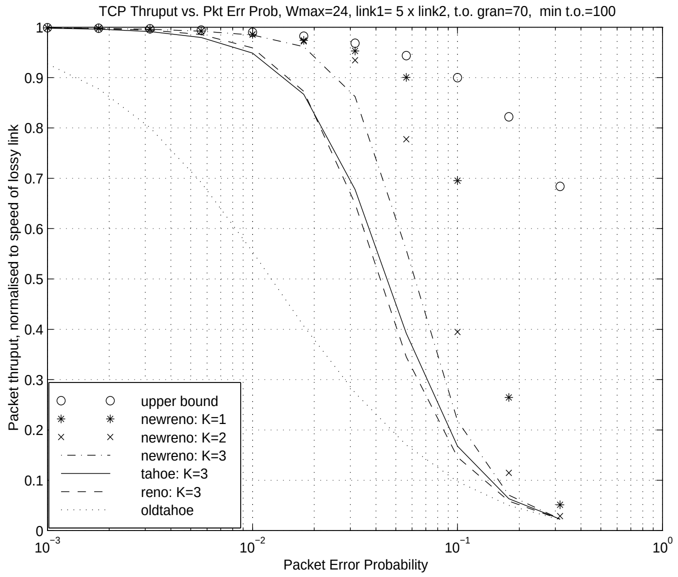

Fig. 6. Throughput of versions of TCP vs. packet loss probability; ($W_{max} = 24$, $\lambda = 5\mu$, K is the fast-retransmit threshold).

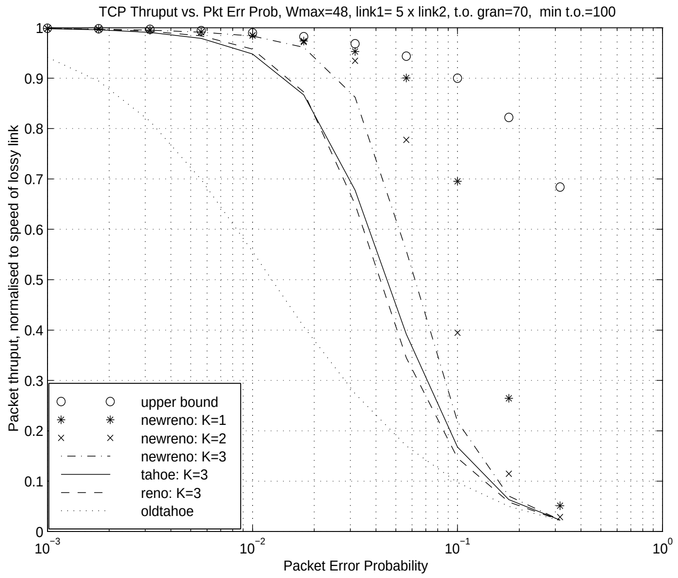

Fig. 7. Throughput of versions of TCP vs. packet loss probability; ($W_{max} = 48$, $\lambda = 5\mu$, K is the fast-retransmit threshold).

"ticks", which implies an average of 750ms, or 100 times the mean packet transmission time on the lossy link. With 1400byte packets, W_{max} of 6, 24, and 48 are equivalent to receiver buffer sizes of about 8kB, 32kB, and 64kB.

In these figures, throughput is plotted against packet loss probability. Since time is normalised to the service time on the lossy link, a throughput of 1 means that the lossy link is always occupied. In each figure we plot an upper bound that is an ideal throughput, not practically achievable by a distributed protocol. This is just the throughput of the closed queuing network with W_{max} customers, and two exponential servers, at the rates λ and μ, multiplied by the probability that a packet is good; i.e., $r(\lambda, W_{max})(1-p)$ ($r(\lambda, w)$ is the lossless throughput of the network with w packets circulating in it, and is given by $r(\lambda, w) = \frac{(\lambda^{(w+1)}-\lambda)}{(\lambda^{(w+1)}-1)}$, if $\lambda \neq 1$, and $r(1, w) = \frac{w}{1+w}$). The bound represents the situation in which packets circulate in the network at the maximum possible rate; the only overhead being that, on an average, $\frac{1}{1-p}$ packets are transmitted to send one good packet.

The numerical results in these figures are qualitatively similar. Throughput degrades with increasing loss probability, but different versions are affected to different extents by the increasing loss probability. TCP-OldTahoe is the worst performing protocol as it always suffers a coarse timeout in each transmission cycle. The fast retransmit feature of all the other versions reduces the probability of coarse timeout to different extents.

There is a precipitous drop in TCP Tahoe, Reno and NewReno performance for loss probabilities exceeding 10^{-2}; for these probabilities fast-retransmit does not succeed very often, and when it fails the entire coarse timeout granularity has to be waited for. Observe that, as expected, for large p, all TCP versions have identically poor performance.

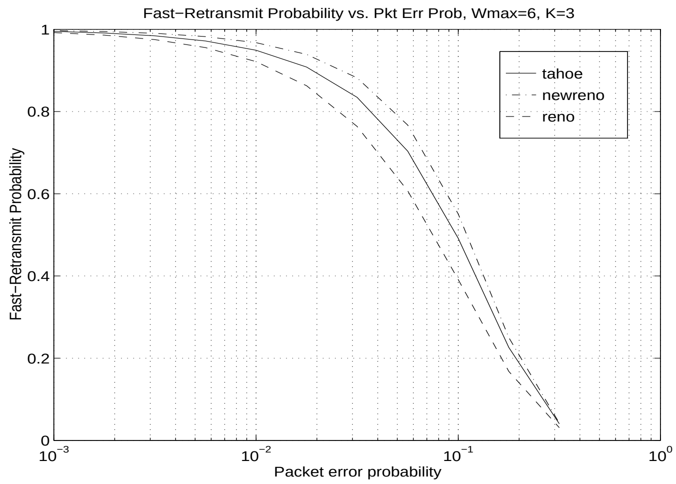

Fig. 8. Fast retransmit probability (see text for definition) vs. packet loss probability; ($W_{max} = 6, K = 3$).

Observe that for $W_{max} = 6$, Reno throughput is worse than that of Tahoe for all values of p. When W_{max} is increased from 6 to 24 (see Figures 5 and 6) the throughput of the protocols that attempt fast-retransmit increases, but the throughput of NewReno increases the most. The larger window causes Reno throughput to exceed that of Tahoe for small p, but still be below Tahoe for large p. With a small window, if a multiple loss occurs then there may not be enough good packets for 2 fast-retransmits to succeed; this is why Reno is poorer than Tahoe for small W_{max} and small p. With a larger window, and large loss probabilities, there is a greater chance of multiple losses in the loss window; this is why Reno throughput does not increase so much,

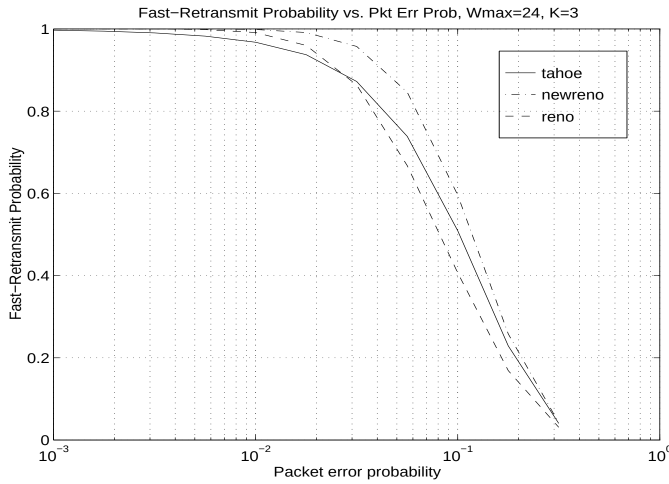

Fig. 9. Fast retransmit probability (see text for definition) vs. packet loss probability; ($W_{max} = 24, K = 3$).

except for very small loss probabilities for which it exceeds that of Tahoe. NewReno derives the most advantage from the larger window. In each cycle Tahoe begins from slow-start, and hence is not able to take as much advantage of the larger window as does NewReno. These statements are clearly supported by the plots of fast-retransmit probability (the expectation, w.r.t. the stationary distribution of $\{U_k\}$, of the probability that coarse timeout is avoided) shown in Figures 8 and 9. Further increase of W_{max} to 48 (Figure 7) does not make any significant difference. Note that these observations corroborate the observations from simulation sample paths reported in [6]. In [6], fragments of simulation sample paths are provided to show how the recovery processes of the various protocols differ when there are one or more packet losses; it is pointed out that Reno's fast recovery could stall if multiple packets are lost in the loss window. In order to obtain throughputs, however, one needs to quantify the probabilities of the various possibilities; our analysis provides such a quantification.

It is useful to obtain a *rule of thumb* for the loss probability up to which the throughput is acceptable, say, more than 90% of the maximum throughput $(1-p)r$, where $r = r(\lambda, W_{max})$ (see above). Suppose we want to find the p up to which the throughput is more than α (say, .9) times $(1-p)r$. Then, using approximations developed in [9], we can show that $p \leq \sqrt{\frac{1-\alpha}{K.r.rto\_min}}$ where we have assumed that α is close to 1. The r in the denominator just scales the rto_min to the network throughput; a smaller value of r for a fixed rto_min will make the rto_min relatively less of a waste as compared to the transmission time in a cycle. For $K = 3$, $rto\_min = 100$, $r = 1$ and $\alpha = .9$, this yields $p \leq .018$, which matches well with the Tahoe curve in Figure 6.

In Figures 5, 6, and 7 we also show for NewReno the effect of reducing K to 2 or to 1. The improvement in throughput is significant, specially for $K = 1$. This is related to the recent TCP-Vegas proposals ([3]), in which the transmitter records the transmission epoch of every packet, and upon the receipt of the first duplicate ack checks to see if the time since the transmission of the missing packet exceeds the current rto value; if it does then that packet is retransmitted. Setting $K = 1$ in our model corresponds to the situation in which there is always a retransmission upon receipt of the first duplicate ack, and hence yields a best case performance with this modification in TCP-Vegas.

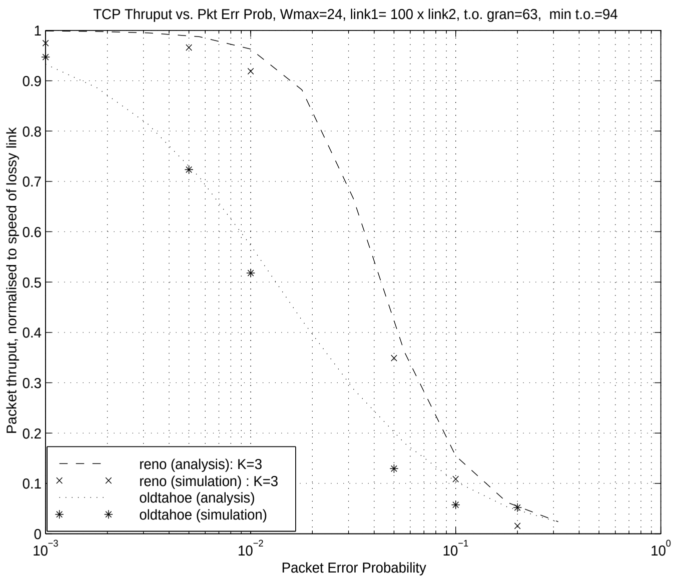

Fig. 10. Throughput of TCP OldTahoe and TCP Reno vs. packet loss probability; comparison of simulation and analysis; ($W_{max} = 24$, $\lambda = 100\mu$).

Finally, in Figure 10 we compare our analytical results with the results from a hybrid TCP simulator. The simulator uses the actual TCP stack on a PC with the Linux operating system; the network model (Figure 2) is implemented in the loopback "driver" in the kernel. Owing to the 1 msec timer granularity in this simulator, we have taken the packet transmission time on the lossy link to be 8 msec; also we could only simulate a very fast source host. The TCP implementation in Linux is a version of Reno. To simulate TCP OldTahoe, the fast retransmit feature in the TCP code is suppressed. Figure 10 shows that in spite of the many assumptions and approximations in the analysis, the analysis and simulation results are quite close. Recall from Equation 1 that the $(1 - p)(EU_{ver} - 1)$ term overestimates the number of successful packets; if this term is dropped, then the analytical results for Reno fall below the simulation results.

VII. Conclusion

In this paper we have studied models for TCP in a local networking scenario, i.e., propagation delays are negligible. One of the key issues that our analysis captures is the impact of coarse timeout values, and we are able to quantify the ability of the fast-retransmit feature to avoid coarse timeouts. We summarise here the main observations from our analysis and the numerical results (the actual numbers are for the parameters we have used in Section VI).

With a minimum timeout of 100 times the mean service time at the lossy link, OldTahoe performs relatively poorly even with loss probabilities as low as .001, its throughput dropping to 0.5 of the lossy link service rate when the loss probability increases to .01. In [11], the authors also report severe degradation of OldTahoe performance with increasing random loss probability. The performance reported by them is much worse since they model a large round trip propagation delay (100 times the bottleneck service time in their numerical results); hence, each time there is a loss, the window has to be built up over several multiples of the round-trip time, until the next loss occurs. At a loss probability of .01, this yields an OldTahoe throughput of about 10% of bottleneck throughput, as compared to 0.5 for our local network scenario.

The fast-retransmit feature in Tahoe, Reno and NewReno improves their performance significantly up to a loss probability of .01, but there is a precipitous drop beyond this point. Coarse timers, and a large minimum timeout, is the main reason for this drop in throughput. Yet the throughput is more than twice that of OldTahoe, up to an error probability of about .15, beyond which all versions provide worse than 10% throughput. Since Reno is pessimistic in doing fast retransmission, and, in case of multiple losses, needs additional duplicate acks for continuing fast recovery, its throughput suffers for large packet loss probabilities; for $p > .02$ it is no better than Tahoe (with fast-retransmit). These results of ours corroborate the observations made by Fall and Floyd [6].

Setting the duplicate ack threshold for fast retransmission to 1 can be expected to double the throughput of the basic fast-retransmit protocols for the error probability range (.05, .2). This is related to one of the proposals in TCP-Vegas (see [3]).

In work reported in detail in [9], we have found that, for the purpose of computing the throughput of the protocols with the fast-retransmit feature, it is important to capture the fast-retransmit probability well, and it is sufficient to use crude approximations for the mean cycle times. For the fast retransmission probability, we found it necessary to use the complete distribution of the congestion window at loss instants, whereas for the mean cycle times an approximation based on a simple fixed-point idea (also reported in [11]) worked well.

The analysis reported in this paper assumes a Bernoulli packet loss model (packet losses are independent with some given probability p). We have extended this analysis to fading wireless channels by using a two state Markov model for packet errors; this work is reported in [10].

Appendix

I. Calculation of $\gamma_{i,j}^{(m)}$

Here we will need the probabilistic window increment model described in Section III-B. We define, for $1 \leq j \leq W_{max}$, and $0 \leq i \leq j$,

$\gamma_{i,j}^{(m)}$ = mean cycle time when the cycle starts with a congestion window j, the slow-start threshold is m, and i packets are queued at the transmitter of the lossy link (i.e., in the IS queue)

$\eta_j^{(m)}$ = probability that the TCP transmitter increments the congestion window when the congestion window is j, the slow-start threshold is m, and an acknowledgement is received

Define, for $0 \leq i, 0 \leq j, 0 \leq i+j \leq W_{max}$,

$r_{i,j}$ = mean time until the next cycle starts after a loss has occurred in a cycle, if, at the loss epoch, the transmitter can send j more packets before exhausting the current window, and i packets are queued at the transmitter of the lossy link

Now, with reference to Figure 2, the following equations are obtained. For $1 \leq j \leq W_{max}$

$$\gamma_{0,j}^{(m)} = \frac{1}{\lambda} + \gamma_{1,j}^{(m)}$$

and, for $1 \leq i \leq (j-1)$,

$$\begin{aligned}\gamma_{i,j}^{(m)} &= \frac{1}{1+\lambda} + \frac{\lambda}{1+\lambda}\gamma_{i+1,j}^{(m)} + \frac{(1-p)(1-\eta_j^{(m)})}{1+\lambda}\gamma_{i-1,j}^{(m)} + \\ & \frac{(1-p)\eta_j^{(m)}}{1+\lambda}\gamma_{i-1,j+1}^{(m)} + \frac{p}{1+\lambda} r_{i-1,j-i}\end{aligned}$$

and

$$\gamma_{j,j}^{(m)} = 1 + (1-p)(1-\eta_j^{(m)})\gamma_{j-1,j}^{(m)} + (1-p)\eta_j^{(m)}\gamma_{j-1,j+1}^{(m)} + p r_{j-1,0}$$

Observe that, given $r_{i,j}, 0 \leq i+j \leq W_{max}-1$, we can calculate $\gamma_{i,j}^{(m)}, 0 \leq j \leq W_{max}, 0 \leq i \leq j$ by first calculating $\gamma_{i,W_{max}}^{(m)}, 0 \leq i \leq W_{max}$, and then recursively calculating for $j = W_{max}-1, j = W_{max}-2, \ldots$, etc.

For the standard TCP congestion control algorithm (adapted to our probabilistic window increment model), the window increment probabilities are given as follows

$$\eta_j^{(m)} = \begin{cases} 1 & \text{for } 1 \leq j \leq m-1 \\ \frac{1}{j} & \text{for } m \leq j \leq W_{max}-1 \\ 0 & \text{for } j = W_{max} \end{cases}$$

It remains to provide the values $r_{i,j}, 0 \leq i+j \leq W_{max}-1$; again these will depend on the particular version of TCP that is being analysed.

Define, for $0 \leq i, 0 \leq j, 0 \leq i+j \leq W_{max}$,

$\alpha_{i,j}$ = starting with i packets in the queue of the lossy link transmitter, and j packets to be transmitted from the TCP transmitter, the mean time until the system is empty, if the TCP transmitter does not send anything after exhausting the j packets

Note that, if the first loss in a cycle occurs at the window M, and if at that time there are $i(\leq M-1)$ packets remaining in the queue of the lossy link, then the "exact timeout" (see Section III-B) will occur $\alpha_{i,M-1-i}$ time units later. The values of $\alpha_{i,j}$ are obtained from the following equations. For $0 \leq i \leq W_{max}$

$$\alpha_{i,0} = \frac{i}{\mu}$$

For $1 \leq j \leq W_{max}$

$$\alpha_{0,j} = \frac{1}{\lambda} + \alpha_{1,j-1}$$

Finally, for $1 \leq j \leq W_{max}-1, 1 \leq i \leq W_{max}-j$,

$$\alpha_{i,j} = \frac{1}{1+\lambda} + \frac{\lambda}{1+\lambda}\alpha_{i+1,j-1} + \frac{1}{1+\lambda}\alpha_{i-1,j}$$

We are now in a position to show how we use the above quantities to obtain the mean cycle times for the various TCP versions. In the following, we denote the timeout granularity by rto_gran, and the minimum timeout by rto_min. Recall that $(u_M, 1 \leq M \leq W_{max})$ denotes the stationary distribution of the Markov chain $\{U_k\}$.

TCP-OldTahoe: In our model of TCP-OldTahoe, the TCP transmitter always times out after the first loss in a cycle, and then starts the next cycle with the window 1 and in the slow-start phase. It follows that, for OldTahoe, for $0 \leq i, 0 \leq j, 0 < i+j \leq W_{max}-1$,

$$r_{i,j} = \max\left\{(\alpha_{i,j} + \frac{rto\_gran}{2}), \frac{j}{\lambda} + rto\_min\right\}.$$

Half the timeout granularity is added to the exact timeout to account for the coarse timeout calculation. Note that $r_{0,0}$ is the recovery time if the very first packet transmitted in a cycle is lost. This would be the second of two consecutive timeouts. Thus, we take care of the exponential backoff behaviour of the timeout calculation mechanism ([15]) by approximating

$$r_{0,0} = 2rto\_min$$

The idea of this approximation is that in the LAN environment, and for a large value of minimum timeout, the first timeout is going to be just the minimum timeout. If the first packet sent, when the next cycle starts, is also lost then the timeout is doubled. It is unlikely that this will happen yet another time.

Recall that the $r_{i,j}$ values are used in the calculation of the $\gamma_{i,j}^{(m)}$ values.

TCP-Tahoe: In our model of TCP-Tahoe, the TCP transmitter does a fast-retransmit if K duplicate acks are received following the first loss in a cycle. Define $\phi_k, 0 \leq k \leq W_{max}-1$, as follows: for $0 \leq k \leq K-1$

$$\phi_k = 0 \tag{9}$$

and, for $K \leq k \leq W_{max}-1$,

$$\phi_k = \sum_{n=K}^{k} \binom{k}{n} (1-p)^n p^{k-n} \tag{10}$$

Note that ϕ_k is the probability that fast-retransmit succeeds if, at the time that the first loss occurs in a cycle, the number of packets in the lossy link queue plus the number that can be sent from the TCP transmitter is k. For our model of TCP-Tahoe, it follows that, for $0 \leq i, 0 \leq j, 0 < i+j \leq W_{max}-1$,

$$\begin{aligned} r_{i,j} &= \phi_{(i+j)}\alpha_{i,j} + (1-\phi_{(i+j)}) \cdot \\ & \max\left\{(\alpha_{i,j} + \frac{rto\_gran}{2}), \frac{j}{\lambda} + rto\_min\right\} \end{aligned}$$

The second term accounts for rto_gran and rto_min when fast-retransmit does not succeed. As before, we take care of the exponential backoff behaviour of the timeout calculation mechanism by approximating

$$r_{0,0} = 2rto\_min$$

These $r_{i,j}$ values are used in the calculation of the $\gamma_{i,j}^{(m)}$ values as shown earlier.

TCP-Reno: In our model of TCP-Reno, the transmitter does a fast retransmit on receiving K duplicate acks, but expects K more duplicate acks for each subsequent packet lost. We need the probabilities $f_M, 1 \leq M \leq W_{max}$, defined by Equations 4, 5, and 6 in Section V-A.1 for TCP-Reno. Using these probabilities, we get for Reno, for $0 \leq i, 0 \leq j, 0 < i+j \leq W_{max}-1$,

$$\begin{aligned} r_{i,j} &= f_{(i+j+1)}\alpha_{i,j} + (1-f_{(i+j+1)}) \cdot \\ & \max\left\{(\alpha_{i,j} + \frac{rto\_gran}{2}), \frac{j}{\lambda} + rto\_min\right\} \end{aligned}$$

Here we are asserting that, for TCP Reno, if a fast-retransmit does occur, the recovery time is given by the $\alpha_{i,j}$ values, since Reno recovers every packet in the loss window before starting the next transmission cycle. As before, to take care of exponential backoff for timeouts, we set

$$r_{0,0} = 2rto\_min$$

Again the $r_{i,j}$ values are used to obtain the $\gamma_{i,j}^{(m)}$ values.

TCP-NewReno: In our model of TCP-NewReno, fast recovery succeeds if, after a loss, K duplicate acks are received. In contrast with Tahoe, however, a fast-retransmit is followed by fast recovery. Thus our model for NewReno combines the $r_{i,j}$ formulas for Tahoe with the mean cycle time computation of Reno. Using the probabilities $f_M, 1 \leq M \leq W_{max}$, defined by Equations 7 and 8 in Section V-A.1, we get, for $0 \leq i, 0 \leq j, 0 < i+j \leq W_{max}-1$,

$$\begin{aligned} r_{i,j} &= f_{(i+j+1)}\alpha_{i,j} + (1-f_{(i+j+1)}) \cdot \\ & \max\left\{(\alpha_{i,j} + \frac{rto\_gran}{2}), \frac{j}{\lambda} + rto\_min\right\} \end{aligned}$$

and

$$r_{0,0} = 2rto\_min$$

Thus, the $\gamma_{i,j}^{(m)}$ values can be computed for NewReno.

ACKNOWLEDGEMENT

The referees' comments have helped greatly to improve the presentation of the paper. The author is grateful to S.G. Sanjay and Ajit Anvekar who developed the hybrid simulator for TCP.

References

[1] A. Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," *Proc. 15th International Conf. on Distributed Computing Systems (ICDCS)*, May 1995.
[2] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *Proc. ACM Sigcomm '96*, Stanford, CA, August 1996.
[3] L. Brakmo and L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communication*, Vol. 13, No. 8, pp. 1465-1480, Oct. 1995.
[4] R. Cáceres, and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 5, pp 850-857, June 1995.
[5] A. Desimone, M.C. Chuah, and O.C. Yue, "Throughput performance of transport layer protocols over wireless LANs." *Proc. IEEE Globecom'93*, December 1993.
[6] K. Fall, and S. Floyd, "Comparisons of Tahoe, Reno, and Sack TCP," manuscript, ftp://ftp.ee.lbl.gov, March 1996.
[7] Van Jacobson, "Congestion avoidance and control," *Proc. ACM Sigcomm '88*, August 1988.
[8] V.K. Joysula, R.B. Bunt, J.J. Harms, "Measurements of TCP performance over wireless connections," *Proc. WIRELESS'96, 8th Int. Conf. Wireless Communications*, Calgary, Alberta, Canada; pp. 289-299, July 8-10, 1996.
[9] Anurag Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," Technical Report WINLAB-TR-129, Rutgers University, October 1996.
[10] Anurag Kumar and Jack Holtzman, "Comparative Performance of Versions of TCP in a Local Network with a Lossy Link, Part II: Rayleigh Fading Mobile Radio Link," Technical Report WINLAB-TR-133, Rutgers University, November 1996. To appear in *Sadhana*, Proceedings of the Indian Academy of Science, 1998.
[11] T.V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss" *IEEE Transactions on Networking*, Vol. 3, No. 3, pp. 336-350, June 1997.
[12] P.P. Mishra, D. Sanghi, and S.K. Tripathi, "TCP Flow Control in Lossy Networks: Analysis and Enhancements," in IFIP Transactions C-13 *Computer Networks, Architecture and Applications*, S.V. Raghavan, G.v. Bochmann, and G. Pujolle (eds.), Elsevier North Holland, pp. 181-193, 1993.
[13] T.S. Rappaport, *Wireless Communications: Principles and Practice*, Prentice Hall, N.J., 1996.
[14] W.R. Stevens, *Unix Network Programming*, Prentice Hall.
[15] W.R. Stevens, *TCP/IP Illustrated, Volume 1*, Addison-Wesley, Reading, Mass., Nov. 1994.
[16] R.W. Wolff, *Stochastic Modelling and the Theory of Queues*, Prentice Hall, 1990.

Anurag Kumar: B. Tech. (E.E.), I.I.T. Kanpur, 1977; Ph.D., Cornell University, 1981. Member of Technical Staff at AT&T Bell Labs, Holmdel, N.J., for over 6 years. Since 1988 with the Indian Institute of Science (IISc), Bangalore, in the Dept. of Electrical Communication Engg., where he is now Professor. Also the Coordinator at IISc of the Education and Research Network (ERNET) Project, which has set up a country-wide computer network for academic and research institutions. Research and consultancy interests: modelling, analysis, control and optimisation problems arising in communication networks and distributed systems.