Comparative Study of Various TCP Versions Over a Wireless Link With Correlated Losses

Farooq Anjum, *Member, IEEE,* and Leandros Tassiulas

***Abstract*—In this paper, we investigate the behavior of the various Transmission Control Protocol (TCP) algorithms over wireless links with correlated packet losses. For such a scenario, we show that the performance of NewReno is worse than the performance of Tahoe in many situations and even OldTahoe in a few situations because of the inefficient fast recovery method of NewReno. We also show that random loss leads to significant throughput deterioration when either the product of the square of the bandwidth-delay ratio and the loss probability when in the good state exceeds one, or the product of the bandwidth-delay ratio and the packet success probability when in the bad state is less than two. The performance of Sack is always seen to be the best and the most robust, thereby arguing for the implementation of TCP Sack over the wireless channel. We also show that under certain conditions the performance depends not only on the bandwidth-delay product but also on the nature of timeout, coarse or fine. We have also investigated the effects of reducing the fast retransmit threshold.**

***Index Terms*—Correlated losses, packet train model, performance analysis, TCP algorithm, TCP over wireless.**

I. Introduction

TRANSMISSION Control Protocol (TCP) is the transport protocol used by many internet-based applications, including http, ftp, telnet, etc. TCP is a reliable end-to-end window-based transport protocol designed for the wireline networks characterized by negligible random packet losses. The way that TCP works is that it keeps increasing the sending rate of packets as long as no packets are lost. When packet losses occur, e.g., due to the network becoming congested, TCP decreases the sending rate. Thus, TCP infers that every packet loss is due to congestion and, hence, backs off in the form of reducing the send window. Extending TCP as used over the wireline links to the wireless links may not be an efficient solution due to the different characteristics of the wireline and the wireless links. This is because wireless networks are characterized by bursty and high channel error rates, unlike the wireline networks. Due to this, the throughput of a TCP connection over a wireless link suffers. In spite of this, the TCP protocol is still used to transfer data over the wireless link, though a lot of attention is currently being given to the design of a better protocol over wireless links [2]–[4], [7], [15], [19]. Because of the difficulty of modeling the TCP protocol analytically, many of these studies have been simulation based. On the other hand, it is not possible to obtain insight into the effects of particular parameters on the behavior of TCP using simulations of specific settings. Further, investigations to improve TCP or design a better transport protocol can become less cumbersome given a simple and accurate analytical model for TCP.

The first step toward the design of a better transport protocol for wireless networks has to be a better understanding of the way TCP works over wireless links. This would reveal the reasons for the inefficiency of TCP over wireless links. There have been several efforts recently on the analytical study of TCP over wireline [12] as well as over wireless links [9]–[11], [20]. Two classes of random losses have been considered, independent identically distributed (i.i.d.) and correlated. The effect of i.i.d. packet losses on TCP performance is studied in [9]. They address the TCP versions Tahoe, Reno, and NewReno, but only in the context of a local network scenario. They also evaluate the various protocol features such as fast retransmit and fast recovery. Unlike in this paper, the packet transmission times are assumed to be exponentially distributed, as is the transmission time of the packet on the lossy link. Further, they also model the congestion avoidance phase probabilistically in which each acknowledgment (ack) causes the window to be incremented by one with a certain probability. Note that both these assumptions have to be resorted to in this approach so as to carry out the mean cycle time analysis. In this study, the authors also consider the less frequent case where the size of the receiver window is the constraint on the sender's window increase and not the bandwidth delay product of the link. Thus, this study precludes the study of the basic TCP mechanism whereby the window size is increased until there is a loss due to congestion. This loss is caused on account of the bandwidth delay constraint. In [13], Mishra *et al.* consider only OldTahoe over a link with i.i.d. losses. Lakshman and Madhow [11] consider Tahoe and Reno in a regime where the bandwidth-delay product of the network is high compared to the buffering in the network. They show using approximate analysis that random independent packet loss leads to significant throughput deterioration when the product of the loss probability and the square of the bandwidth-delay product is larger than one.

In [10], Kumar and Holtzman consider the behavior of TCP Tahoe and OldTahoe in the presence of correlated packet losses. The results obtained are applicable only in case of very low bandwidth wireless links. More emphasis is placed on analyzing a link layer solution, of hiding the losses from the transport layer

Manuscript received June 3, 1999; revised April 3, 2002; approved by IEEE/ACM Transactions on Networking Editor T. V. Lakshman. This work was supported in part by the National Science Foundation under CAREER Award NCR-9502614 and by the Air Force Office of Scientific Research under Grant 95-1-0061. This paper was presented in part at the ACM SIGMETRICS, Atlanta, GA, 1999.

F. Anjum is with Telcordia Technologies, Morristown, NJ 07960 USA (e-mail: fanjum@telcordia.com).

L. Tassiulas is with the Department of Electrical Engineering, University of Maryland, College Park, MD 20742-3285 USA (e-mail: leandros@glue.umd.edu).

Digital Object Identifier 10.1109/TNET.2003.813033

by link layer retransmissions, to the problem faced by TCP over the wireless link. The approach used is also similar to the approach used in [9] and, hence, suffers from the drawbacks noted earlier. Zorzi and Rao [20] study TCP OldTahoe assuming a correlated loss model. The approach followed by them requires enumerating the transmission time, acknowledgment time, and timeout time of every packet in a cycle. This is computationally cumbersome in the case of analysis of links with large bandwidth delay products, since in this case large numbers of packets have to be accounted for. Further, this approach cannot be taken to model the fast retransmit and fast recovery mechanisms present in the newer TCP versions like Tahoe, Reno, NewReno, and Sack.

Simulation-based studies evaluating the performance of different TCP algorithms have also been reported. In [5], Chockalingam *et al.* deal with a simulation study of TCP NewReno under the effect of correlated errors on the wireless link. They consider only a 1.5-Mb/s wireless link characterized by a very low delay-bandwidth product.

In this paper, we analyze the behavior of the different TCP algorithms OldTahoe, Tahoe, NewReno, and Sack under conditions whereby the wireless channel is subjected to the more realistic case of correlated packet losses. Since the main aim of this study is to address the behavior of the different TCP versions under different conditions of the wireless channel, we do not consider the effects of multiple flows at all. This, in fact, could be a subject for future work. We show that in certain circumstances characterized by bursty loss conditions, the performance of NewReno can lag behind the performance of Tahoe. This performance gap can worsen against NewReno as the bandwidth-delay product increases. We also show that the performance of Sack is the best under all conditions, though at loss rates characterized by very small durations of good periods, the performance of the different versions is equally bad. We also derive conditions under which significant throughput deterioration occurs under correlated loss conditions. Investigations into the effects of reducing the fast retransmit threshold as well as using finer timeout intervals are also carried out and we show that these incremental changes to the TCP versions are effective only in case of lossy conditions.

This paper is organized as follows. In Section II, we specify the correlated loss model in detail. In Section III, we explain the approach that we take in order to study the behavior of the different TCP versions under different wireless channel conditions. In Section IV, we characterize the throughput of the different TCP algorithms as a function of the wireless channel parameters. Section V deals with the numerical study of the performance of the different TCP versions. Finally, conclusions are presented in Section VI. We would like to point out that due to lack of space, we do not describe the different TCP algorithms here. Readers requiring details can refer to [6], [9], and [17].

II. Loss Models

It is well known that mobile radio channels in an actual physical environment are subject to multipath fading. The multipath fading process in a mobile environment follows a Rayleigh distribution [8]. In order to model a correlated Rayleigh fading channel using a Markov chain, we consider a simple two-state Markov chain called the Gilbert–Eliot model as in [10], from which the description next is adapted. The two states that we consider are the "Good" state and the "Bad" state. We assume that the packet succeeds with probability 1 while in the good state and is lost with probability 1 while in the bad state. The transition probability matrix C_m of the simple two-state Markov chain is then given by

$$C_m = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}. \tag{1}$$

The chain is assumed to be embedded at the beginnings of packet transmissions. Thus, if a number of packets are being transmitted back to back, and if the channel is in a Good state when a packet is about to be transmitted, then this packet will be successful and the next packet will be successful or unsuccessful with probabilities $1-\alpha$ and α, respectively. This model which is commonly adopted in wireless fading channels [5], [10], [20] tracks fading but excludes impairments such as path loss and shadowing which vary on a much longer time scale. These phenomena are assumed to be taken care of by power control mechanisms and hence are not considered. Thus, given α and β, the channel properties are completely characterized. Further, the steady-state properties of the chain are then given by π_G and π_B which symbolize the steady-state probability that the channel is in the good state and bad state, respectively. These are given by

$$\pi_G = \frac{\beta}{\alpha+\beta}, \quad \pi_B = \frac{\alpha}{\alpha+\beta}. \tag{2}$$

Note that the average duration of the bad state is given by $1/\beta$ while the average duration of the good state is given by $1/\alpha$ slots or packets since we assume that each slot corresponds to the transmission time of a packet.

III. Approach

The behavior of the different TCP algorithms is analyzed based on the concept of packet trains which we introduce next. For ease of explanation, we consider TCP NewReno. A NewReno sender when sending new data is either in the slow start phase or the congestion avoidance phase. The loss of a packet in either of these phases is subsequently followed by mechanisms to recover the lost packet(s). A NewReno sender uses the fast retransmit mechanism whereby the arrival of a certain number of duplicate acks signals a packet loss. If not enough duplicate acks (as required by the fast retransmit mechanism) arrive back at the sender, then a timeout, which is the nonarrival of the ack of a packet within a certain time interval, is taken as an indication of packet loss. Following the detection of a packet loss through the fast retransmit mechanism the NewReno transmitter resorts to the fast recovery mechanism whereby the window size is reduced by half and the sender uses additional incoming acks to clock outgoing packets. The fast recovery mechanism concludes successfully when all the lost packets have been recovered and is followed by the congestion avoidance mechanism. If a retransmit timeout has to be resorted to, then the sender always starts in the slow start phase after the timeout interval.

In the model described, we ignore the fact that on successive timeouts the actual value of the timeout is increased exponentially. Further, for the purpose of simplifying the analysis, we also assume that packets retransmitted during fast recovery are not dropped. As a result, it is to be kept in mind that our analysis is less conservative and, hence, gives optimistic values compared to the actual implementations during regimes of high loss probability. We would also like to remark that it is very much possible using our approach to give up these assumptions at the cost of higher complexity. Since these assumptions affect the performance only in the region of high loss probability where the efficiency is already very low, we choose to reduce complexity by making these assumptions. Note also that these approximations done to simplify the analysis in no way affect the results that we obtain in this paper. As done in case of the other studies, we assume that acks are not lost. This is justified because of the fact that ack packets have a very small loss probability due to the small size of the packets. Further because of the cumulative nature of acks, the only consequence of ack losses is increased burstiness on the forward path [18].

We consider the operation of a NewReno transmitter in terms of cycles. A cycle begins with either the slow start phase or the congestion avoidance phase following the detection of a packet loss. The cycle ends either with the successful conclusion of the fast recovery mechanism or on the basis of the retransmit timeout mechanism. This timeout can occur due to the absence of the fast recovery mechanism. The fast recovery mechanism may be absent because of the lack of adequate duplicate acks to infer packet loss, thereby causing the fast retransmit mechanism to fail. A typical cycle in case of NewReno can start in the congestion avoidance phase, have the fast recovery phase, and end on the conclusion of the fast recovery phase, because all the packets transmitted during the fast recovery stage have been successful in reaching the receiver.

Thus, every cycle can be considered to have three stages. In the first stage, the sender is either in the slow start or the congestion avoidance phase. The fast recovery mechanism which constitutes the second stage follows the detection of a packet loss. During this stage, the window size is reduced and the packets inferred to be lost are retransmitted. The third stage consists of the retransmit timeout interval. While the first stage is present in every cycle, the second and the third stages may or may not be present depending on the packet loss(es).

In order to explain the working of these algorithms we define the kth minicycle of the first stage of ith cycle to be the time taken to transfer the kth window of packets during the first stage of the ith cycle. Hence, the first minicycle corresponds to the first window of packets on the start of a new cycle. Thus, every cycle of NewReno begins with one packet being transmitted in the first minicycle if it is in the slow start phase. If it is in the congestion avoidance phase, then the number of packets transmitted in the first minicycle depends on the window size when the packet drop was detected in the previous cycle. In every successive minicycle, the number of packets transferred is double the number of packets transferred during the present minicycle as long as the sender is in the slow start phase. During the congestion avoidance phase, the number of packets transferred in each minicycle is one more than the number of packets transferred during the previous minicycle.[1] This goes on until there are one or more packet drops in a particular cycle, causing the ack cycle to either dry up or generate enough duplicate acks resulting in the end of the first stage of the cycle. The end of the first stage can lead to the second stage. Since a NewReno sender recovers only one lost packet per round-trip time (RTT) during the fast recovery stage, we consider that each minicycle during the second stage carries a single packet which is being retransmitted. In practice, there may also be some new packets transmitted during this phase. We do not take these new packets into consideration. On the other hand, practical implementations have a bound on the number of packets which can be transmitted once the fast recovery stage is finished. We do not consider such bounds and, hence, both these effects should cancel out each other in our analysis, thereby having the analytical model follow the practical TCP implementations closely.

We assume that all the packets in a minicycle travel in a *train*. Thus, there is a packet train in every minicycle and the size of the packet train in the kth minicycle $k > 1$ depends on the size of the packet train in the previous minicycle and the phase of the NewReno sender. A new packet train starts once the ack for the first successful packet of the previous packet train comes back. This train ends when the packets corresponding to the last ack of the previous train have been transmitted by the sender or a certain number of duplicate acks reach the sender, thus giving some length to the train. Start of a successful timeout at any point also terminates a train. The length of the packet train which is the distance between the first packet of the train and the last packet of the train keeps on increasing since the number of packets in successive trains is an increasing function. This concept of a packet train is illustrated in Fig. 1. A great convenience offered by the packet train concept is that it helps to differentiate packets on the basis of the minicycle that they belong to. This, as we see later, greatly helps in calculating the throughput of a flow. This is because once we know the number of trains in a cycle as well as the window size σ when the packet drop was detected in the last cycle, the expected number of packets in the cycle as well as the mean cycle duration can be easily calculated. As we see later, this is the approach that we take to characterize the throughput of a flow. Of course, the expected number of trains in a cycle as well as the mean loss window depend on the TCP algorithm followed as well as on the packet loss probability characteristics of the wireless channel.

We would like to remark here that the concept of packet trains is not new and has also been considered elsewhere [14]. However, the details are sufficiently different especially in terms of the granularity and the assumptions made.

Example: Before proceeding further, we illustrate the above concepts using an example of a TCP NewReno sender. Consider the evolution of a TCP NewReno sender as shown in Table I. We start observation when the ack for packet 14 reaches the sender. We assume that packets 15–18 are lost. Thus, the first stage of this cycle ends on account of the fast retransmit feature when the third duplicate ack reaches the sender since the receiver has received packet 21. The second stage starts with the transmis-

[1]For ease of explanation, we are ignoring some constraints like delayed acks, which though can be easily incorporated into the description.

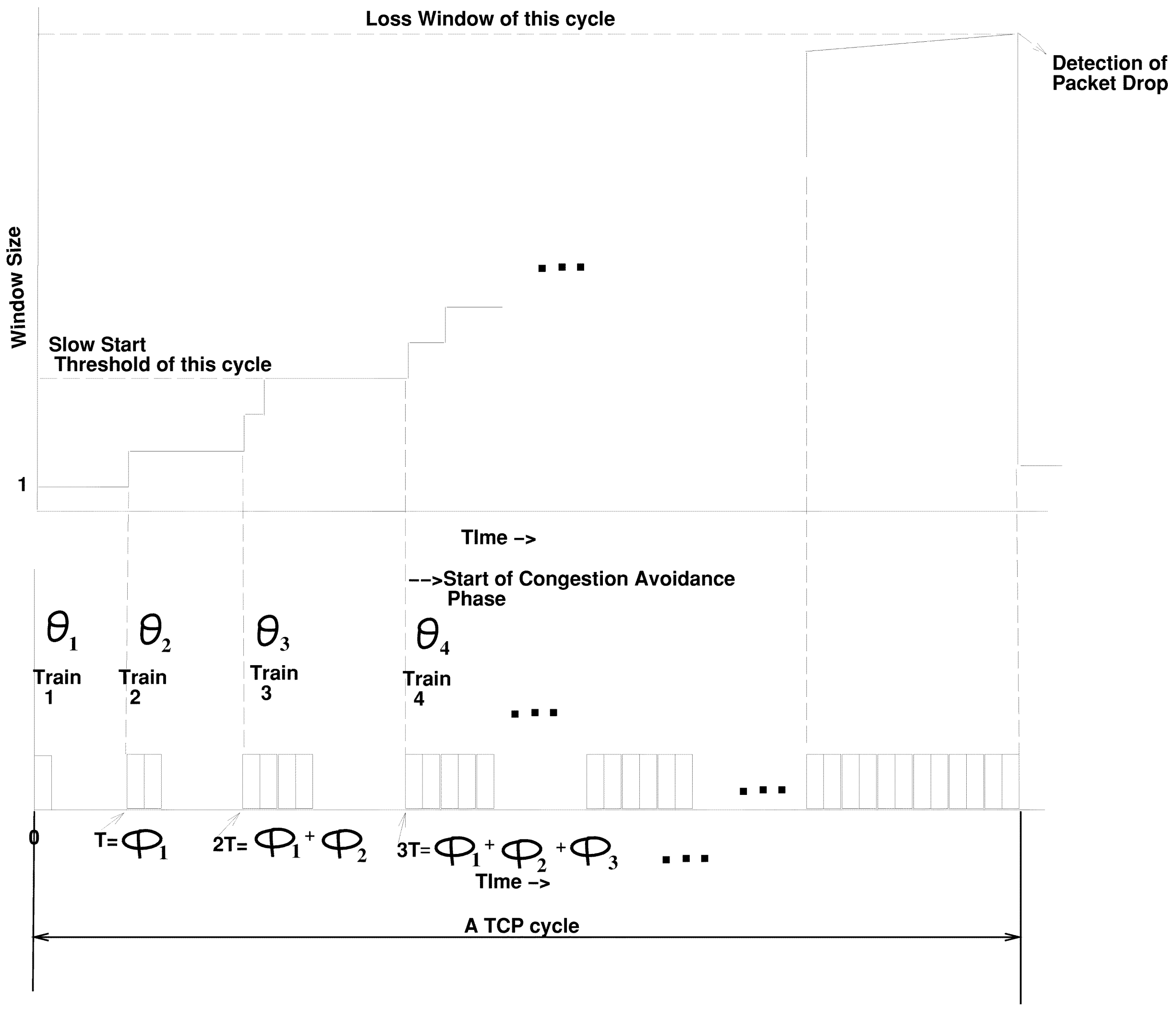

Fig. 1. Illustration of packet trains.

sion of packet 15. On the receipt of the ack for packet 15, the next packet, 16, is retransmitted. This repeats until packet 18 is retransmitted. Thus, all the lost packets are recovered by retransmitting only one lost packet per RTT. This cycle ends once the ack for packet 18 comes back, following which a new cycle begins.

Since the previous cycle did not have the third stage, the sender starts in the congestion avoidance phase in the first stage of the new cycle. Packets 23–26 are transmitted on the start of the new cycle and these packets constitute the first train. A new train (second train) starts once the ack for packet 23 reaches the sender and ends when the packets corresponding to the last ack of the previous train (for packet 26) have been transmitted. Thus, the second train starts with packet 27 and ends with packet 31. We assume that the first packet dropped in this cycle is packet 29, which is in the second train. Packets 30 and 31 are also assumed to be lost.

Since packet 27 is successful, the third train starts (on the receipt of the ack for packet 27) with the transmission of packet 32. Note that the TCP sender has not yet detected the loss of packet 29. Packet 33 is sent on the receipt of the ack for packet 28. Since the next three packets (29–31) are lost, no acks arrive for those packets. This signifies the end of the first stage of this cycle. At the same time, the timeout clock starts ticking after the transmission of packet 33. The acks for packets 32 and 33

TABLE I
TCP NEWRENO CYCLE

Pkt Recd info in ack	Ack	ω	W	Pkt sent
14	14	5(say)	8	22
14 I D.A.	19	5	8	(pkt 15-18 lost)
14 II D.A.	20	5	8	-
14 III D.A.(II stage)	21	4	7	15
14 IV D.A	22	4	8	
15	15	4	4	16
16	16	4	4	17
17	17	4	4	18
22(New cycle, I train)	18	4	4	23,24,25,26
23(II train)	23	4	4..	27
24	24	4	4..	28
25	25	4	4..	29(lost)
26	26	4	5	30 (lost),31(lost)
27(III train)	27	4	5..	32
28	28	4	5..	33
Timeout(III Stage)				
28(I D.A)	32	4	5...	-
28(II D.A.)	33	4	5..	-
New cycle	-	2	1	29

are duplicate acks and, hence, do not result in the transmission of any packets. Note that if not for the packet losses, then a new train would have started with the receipt of the ack for packet 32. Because of the lack of enough duplicate acks, this cycle ends with the third stage of a timeout. In this cycle, we have two stages: the first stage and the third stage. The next cycle starts with the retransmission of packet 29 and the process continues. Note that since the last cycle ended with a timeout, the sender in the new cycle begins in the slow start phase while in the first stage.

Now consider the other TCP algorithms. Any TCP sender, whether OldTahoe, Tahoe, or Sack when sending new data is generally either in the slow start phase or the congestion avoidance phase. An OldTahoe sender uses timeout as the mechanism to infer packet losses. A Tahoe sender, on the other hand, can also use the fast retransmit mechanism. Both of these senders resort to slow start after the detection of a packet loss by reducing the window size to one. The Sack sender, on the other hand, uses the fast retransmit mechanism to infer packet losses. But following the detection of a packet loss, the Sack sender, like the NewReno sender, resorts to the fast recovery mechanism whereby the window size is reduced to half and the sender uses additional incoming acks to clock outgoing packets. The fast recovery mechanism concludes successfully when all the lost packets have been recovered, and this is followed by the congestion avoidance mechanism. Of course, if there are not enough duplicate acks as required by the fast retransmit mechanism, then a retransmit timeout has to be resorted to, following which the sender always starts in the slow start phase.

Thus, a cycle of a Tahoe or OldTahoe sender starts with the slow start phase and ends following the detection of a packet loss either on the basis of the ack-based fast retransmit mechanism or the timer-based retransmit timeout mechanism. On the other hand, a cycle of the Sack sender begins with either the slow start phase or the congestion avoidance phase following the detection of a packet loss. The cycle ends either with the successful conclusion of the fast recovery mechanism or on the basis of the retransmit timeout mechanism. Thus, the OldTahoe, Tahoe, and Sack senders always have the first stage in a cycle. An OldTahoe sender does not have the second stage of fast recovery and always has the third stage of retransmit timeout. The Tahoe sender will not have the second stage and may or may not have the third stage of a timeout depending on whether enough packets are successful after the lost packet for the fast retransmit mechanism to succeed. In contrast, a Sack sender, like the NewReno sender, can have either the second stage of fast recovery or the third stage of a retransmit timeout if adequate acks to activate the fast retransmit mechanism are not present.

We now go back to the analysis. Let the number of trains in a cycle be denoted by n. On the basis of our explanation above, it is clear that there are two types of trains: type-1 trains during stage 1 and type-2 trains during stage 2. The first type of trains are the normal trains constituting the first stage of a cycle. Let n_1 denote the number of such trains. The type-2 trains are those constituting fast recovery. In the case of NewReno, these trains have only one packet that has been detected to have been lost previously. Let n_2 denote the number of such trains. Note also that $n_2 = 0$ for Tahoe and OldTahoe. $n_2 = 1$ for Sack if the cycle does not end in a timeout. This is because Tahoe and OldTahoe do not have fast recovery, while Sack has fast recovery, but it does not have many of these trains since it is not constrained to retransmit at most one dropped packet per RTT as in the case of Reno or NewReno.

As is clear from the above explanation, every cycle of OldTahoe and Tahoe begins with a slow start. In the case of NewReno and Sack, a cycle begins in the slow start phase only after a timeout has occurred. In the sequel, the size of the kth type-1 train in case of such cycles is denoted by θ_k. Thus, θ_k denotes the number of packets in the kth train. Further, $\theta_k = f(\theta_{k-1})$ and, thus, depends on the TCP algorithm followed. We also let $\theta_0 = 0$. For the cycles of NewReno and Sack starting in the congestion avoidance phase, the size of the kth type-1 train is denoted by $\bar{\theta}_k$.

The size of the loss window, i.e., the window size at which the packet loss is detected by the sender, is denoted by σ. W_m denotes the maximum size that the window can grow to, which by our earlier assumptions is given by $W_m = \mu T + 1$, where μ denotes the bandwidth of the wireless link, while the RTT of the link is denoted by T. Further, P_τ denotes the steady state probability of a timeout in a cycle. We will introduce further notation as we go along when necessary.

With this, let us consider the packet trains. The metric that we consider to evaluate the different TCP algorithms is the *throughput*, which we define next. Let the number of packets transferred during a cycle be denoted by $\hat{Q}$ and the duration of the cycle be denoted by ζ. Note that both $\hat{Q}$ and ζ are random variables. The steady state throughput Δ obtained by the TCP connection is given as

$$\Delta = \frac{E(\hat{Q})}{E(\zeta)}. \tag{3}$$

To determine the number of packets sent in a cycle, we need to know not only the number of trains in the cycle but also the window size in the previous cycle at which a packet loss was detected. Also, let Q denote the number of packets sent before the first dropped packet. $\tilde{Q}$ denotes the sum of Q and the number of packets in the second stage, if present. ϵ denotes the number of packets successfully sent by the source after the packet loss but before the packet loss is detected by the sender while still in the first stage. Hence, we have

$$\begin{aligned} E(\hat{Q}) &= E[E(\tilde{Q}/n_1, n_2, \sigma)] + E(\epsilon) \\ &= \sum_{\sigma}\sum_{n_1}\sum_{n_2} P(n_1, n_2, \sigma) E(\tilde{Q}/n_1, n_2, \sigma) + E(\sigma) \\ &= \sum_{\sigma} P(\sigma) \sum_{n_1} P(n_1/\sigma) \sum_{n_2} P(n_2/n_1, \sigma) \\ &\quad \times E(\tilde{Q}/n_1, n_2, \sigma) + E(\sigma) \\ &= \sum_{\sigma} P(\sigma) \sum_{n_1} P(n_1/\sigma) \sum_{n_2} P(n_2/n_1, \sigma) \\ &\quad \times [n_2 + E(Q/n_1, \sigma)] + E(\sigma) \\ &= E[E(n_2/n_1, \sigma)] + E[E(Q/n_1, \sigma)] + E(\sigma) \end{aligned} \tag{4}$$

and

$$E(Q/n_1, \sigma) = \sum_{k=0}^{n_1} \theta_k. \tag{5}$$

Equation (4) follows from (3) since $E(\tilde{Q}/n_1, n_2, \sigma) = n_2 + E(Q/n_1, \sigma)$. For the derivation above, for simplicity we assume that $E(\epsilon) = E(\sigma)$.

Remark: In the above, $P(\sigma)$ denotes the steady state probability of the loss window, i.e., the window size in the cycle at which the first stage ends due to a packet loss which is also the same as the window size at which the packet loss is detected by the sender. $P(n_1/\sigma)$ denotes the steady state conditional probability on the number of type-1 trains given the loss window, while $P(n_2/n_1, \sigma)$ is the steady state conditional probability on the number of type-2 trains given the loss window and the number of type-1 trains. $E(Q/n_1, \sigma)$ denotes the expected number of packets in stage 1 of a cycle given the loss window and the number of trains in the first stage.

We next look at the expected cycle time. Let ϕ_k denote the RTT taken by a packet of the kth type-1 train. Then we have

$$\zeta_{\mathrm{OT}} = \sum_{k=1}^{n_1+1} \phi_k + \tau_t \tag{6}$$

$$\zeta_{\mathrm{T}} = \sum_{k=1}^{n_1+1} \phi_k + I_\tau \tau_t \tag{7}$$

$$\zeta_{\mathrm{NR}} = \sum_{k=1}^{n_1+1} \phi_k + (1 - I_\tau) n_2 T + I_\tau \tau_t \tag{8}$$

$$\zeta_{\mathrm{S}} = \sum_{k=1}^{n_1+1} \phi_k + (1 - I_\tau) T + I_\tau \tau_t \tag{9}$$

where I_τ denotes the indicator function of the timeout event and τ_t denotes the value of the timeout interval, which is typically a multiple of 500 ms. OT refers to OldTahoe, T refers to Tahoe, NR to NewReno, and S to Sack. We would also like to point out that the summation in the above equations is over $n_1 + 1$ type-1 trains, since a packet lost in the n_1th type-1 train can be detected by the sender only in the $n_1 + 1$th type-1 train. Further, the effect of the exponential backoff in case of multiple timeouts can be incorporated by substituting a random variable denoting the size of the timeout interval instead of $I_\tau \tau_t$ in the equations above. Taking expectations, assuming $E(\phi_k) = T$, we have

$$\begin{aligned} E(\zeta_{\mathrm{OT}}) &= (E(n_1) + 1)T + \tau_t \\ E(\zeta_T) &= (E(n_1) + 1)T + \tau_t P_\tau \\ E(\zeta_{\mathrm{NR}}) &= (E(n_1) + 1)T + E(n_2)T + \tau_t P_\tau \\ E(\zeta_S) &= (E(n_1) + 1)T + (1 - P_\tau)T + \tau_t P_\tau. \end{aligned} \tag{10}$$

At this point, we can use (4) and (10) in (3) in order to calculate the throughput of a TCP flow for any of the TCP senders, namely, OldTahoe, Tahoe, NewReno, or Sack. But in order to use (4) and (10) for any TCP versions, we need to specify expressions for $P(\sigma), P(n_1/\sigma)$ and $P(n_2/n_1, \sigma), E(Q/n_1, \sigma)$ and P_τ. Hence, we next try to evaluate the expressions $P(\sigma), P(n_1/\sigma)$ and $P(n_2/n_1, \sigma), E(Q/n_1, \sigma)$, and P_τ, calling them as the loss window probability calculation, train probability calculation, packet count calculation, and timeout probability calculation, respectively, for the different TCP algorithms. Note that due to the different mechanics of the various TCP algorithms, the required expressions above would highly depend on the TCP algorithm being considered. This entire approach to characterize the throughput of a window-based protocol is what we call the *packet train model*.

Since we are considering the packet train model, the success or failure of a particular packet depends on the success or failure of the previous packet as long as both the packets are in the same train. In the sequel, we assume that the first packet of every train finds the wireless link in its steady state. This is realistic as long as the intertrain gap is large enough which happens when we consider large bandwidth links under scenarios whereby the window does not grow to the maximum possible values. In such a case, the duration between trains is quite large compared to the packet transmission time and, hence, our assumption does in fact reflect reality. This assumption is also justified due to the fact that we are interested in the comparison of different TCP algorithms operating over a wireless link and, hence, this assumption affects all the different algorithms to the same extent.

Scenarios whereby the window does grow to the maximum possible values such that the intertrain gap is low enough can also be very easily taken care of. This can be done by calculating the window size W until the point at which our assumption holds. This window size W is a function of the intertrain gap which depends on the bandwidth of the link and the packet size. For window sizes beyond W, we consider that the success or failure of the first packet of a train depends on the success or failure of the last packet of the previous train, while for window size less than W we use the same approach as above. For simplicity, we do not further pursue this approach here.

IV. Throughput Characterization

In this section, we proceed to specify expressions for the loss window probability calculation, train probability calculation, packet count calculation, and timeout probability calculation. Due to lack of space, we show these calculations only for Tahoe. Expressions for the other algorithms follow similarly as the expressions for Tahoe and are provided in detail in [1]. We next describe the approach that we take in order to calculate $P(\sigma)$. Let W_ρ denote the maximum window size reached in the ρth cycle. Thus, $\{W_\rho\}_\rho$ denotes the sequence of window sizes at which packets are dropped in successive cycles. It is obvious that the loss window sequence $\{W_\rho\}_\rho$ forms a Markov chain. Hence, in order to determine the steady state probability of the loss window $P(\sigma), \sigma = 1, 2, \ldots, W_m$, we seek to characterize the probability $P(W_{\rho+1} = \eta / W_\rho = \sigma)$. Given the transition probability matrix, we can generate the stationary distribution $P(\sigma)$ using any of the standard methods. Thus, during the window probability calculation we characterize the transition probability matrix $P(W_{\rho+1} = \eta / W_\rho = \sigma)$.

Proposition 1: Loss Window Probability Calculation—Transition Probability: Consider TCP OldTahoe or Tahoe. Let the wireless link be governed by the correlated loss model with parameters α and β. Then, with ω as the slow start threshold we have

$$P(W_\rho = \omega + k / W_{\rho-1} = \sigma) \tag{11}$$

$$= \begin{cases} \alpha(1-\alpha)^{\omega+k-1}\left(\pi_G + \frac{\pi_B\beta}{1-\alpha}\right)^{\gamma_1}, & -\omega < k < 0 \,\&\, A \\ (1-\alpha)^{\omega+k-1}\left(\pi_G + \frac{\pi_B\beta}{1-\alpha}\right)^{\gamma_1}, & -\omega < k < 0 \,\&\, B \\ (\pi_G\alpha + \pi_B(1-\beta)) & \\ (1-\alpha)^{\gamma_3}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\gamma_2}, & 0 \le k < W_m - \omega \\ (1-\psi_1^\omega - \psi_2^\omega) & \\ (1-\alpha)^{\gamma_3}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\gamma_2}, & k = W_m - \omega \end{cases} \tag{12}$$

where

$$\begin{aligned} \gamma_1 &= \lceil \log_2(\omega+k) \rceil \\ \gamma_2 &= \lceil \log_2(\omega) \rceil + k \\ \gamma_3 &= s(\omega-1,k) = \omega - 1 + \omega + \cdots + \omega - 1 + k \\ \psi_1^\omega &= \pi_G(1-\alpha)^{\omega+k} \\ \psi_2^\omega &= \pi_B\beta(1-\alpha)^{\omega+k-1} \end{aligned}$$

and A denotes $\log_2(\omega+k)$ not int while B denotes $\log_2(\omega+k)$ int.

Proof: Consider a TCP Tahoe cycle. Every cycle starts with a window of one and a slow start threshold of ω. Then during the slow start phase, the window advances by one for every successful packet. Hence, to achieve a window of η, $\eta < \omega$, $\eta - 1$ packets have to be successful. Let $S_s(\eta)$ denote the probability of packets successfully reaching the receiver so as to attain a window size of η such that η belongs to the slow start phase, and let $F_s(\eta)$ denote the probability of a packet loss so as to have a loss window size of η, such that η belongs to the slow start phase. Thus, during the slow start phase, the probability of having a loss window of η in the ρth cycle, given that the loss window during the previous cycle is σ, is given by $S_s(\eta)F_s(\eta)$. We need to show that

$$S_s(\eta) = (1-\alpha)^{\eta-1}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\lceil \log_2(\eta) \rceil}$$

$$F_s(\eta) = \begin{cases} \pi_G\alpha + \pi_B(1-\beta), & \log_2(\eta) \text{ is an integer} \\ \alpha, & \log_2(\eta) \text{ is not an integer.} \end{cases}$$

$\omega + k$ is the drop window size during the cycle of interest. Since we consider TCP Tahoe during the slow start phase, we have $k < 0$. The window size and the slow start phase together imply that $\omega+k-1$ packets have been successful while the $\omega+k$th packet has been dropped. The train number corresponding to packet $\omega+k-1$ is $\lceil \log_2(\omega+k) \rceil$. Let $\log_2(\omega+k)$ not be an integer which implies that the $\omega+k$th packet is not the first packet of any train. In such a case, the probability of dropping the $\omega+k$th packet is affected by what happened to packet $\omega+k-1$. But packet $\omega+k-1$ has been successful, hence

$$F_s(\omega+k) = \alpha, \quad \text{if } \log_2(\omega+k) \text{ is not an integer.} \tag{13}$$

Let $\log_2(\omega+k)$ be an integer. In such a case, the $\omega+k$th packet is the first packet of a train. Hence, it is not affected by what happened to the previous packet. The probability of dropping the $\omega+k$th packet is then given by

$$\begin{aligned} F_s(\omega+k) &= \sum_{i=\text{good,bad}} \text{Prob that channel is in } i\text{th state} \\ &\quad \text{probability of dropping packet} \\ &= \pi_G\alpha + \pi_B(1-\beta), \quad \text{if } \log_2(\omega+k) \text{ is an integer.} \end{aligned} \tag{14}$$

Now consider the probability of $\omega+k-1$ packets being successful. $\omega+k-1$ packets occupy $\lceil \log_2(\omega+k) \rceil$ trains. The channel is in a good state before the start of a new train with probability π_G. Hence, the first packet of the train in such a condition is successful with probability $1-\alpha$. If the channel is in a bad state before the start of a train, which happens with probability π_B, then the packet is successful with a probability β. Thus, we have $\lceil \log_2(\omega+k) \rceil$ places to fill with either π_G or π_B in all possible combinations. Hence

$$\begin{aligned} S_s(\omega+k) &= \pi_G^{\lceil \log_2(\omega+k) \rceil}(1-\alpha)^{\omega+k-1} \\ &\quad + \pi_G^{\lceil \log_2(\omega+k) \rceil - 1}(1-\alpha)^{\omega+k-2}\pi_B\beta + \cdots \\ &\quad + \pi_G^{\lceil \log_2(\omega+k) \rceil - j}(1-\alpha)^{\omega+k-1-j}\pi_B^j\beta^j + \cdots \\ &\quad + (1-\alpha)^{\omega+k-1-\lceil \log_2(\omega+k) \rceil}(\pi_B\beta)^{\lceil \log_2(\omega+k) \rceil} \\ &= (1-\alpha)^{\omega+k-1}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\lceil \log_2(\omega+k) \rceil}. \end{aligned} \tag{15}$$

Thus, the expression for the transition probability for the case $k < 0$, i.e., the slow start phase, follows from (13), (14), and (15).

Now consider the case $k \ge 0$, i.e., the congestion avoidance phase. Let $S_c^\sigma(\eta)$ denote the probability of packets successfully reaching the receiver so as to attain a window size of η such that η belongs to the congestion avoidance phase. Further, let $F_c(\eta)$ denote the probability of a packet loss so as to have a loss window size of η, such that η belongs to the congestion avoidance phase. Thus, during the congestion avoidance phase the probability of having a loss window of η in the ρth cycle given that the loss window during the previous cycle is σ is given by $S_c^\sigma(\eta)F_c(\eta)$. In this case, we need to show that

$$S_c^\sigma(\eta) = (1-\alpha)^{s(\omega-1,\eta-\omega)}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\lceil \log_2(\omega) \rceil + \eta - \omega}$$

$$\begin{aligned} F_c(\eta) &= 1 - \pi_G(1-\alpha)^\eta - \pi_B\beta(1-\alpha)^{\eta-1}, \qquad \eta < W_m \\ &= 1, \qquad \eta = W_m. \end{aligned}$$

During the congestion avoidance phase, the window advances by one packet for every successful window of packets delivered. For a drop window of $\omega+k$ during the present cycle, at least one of the $\omega+k$ packets has to be unsuccessful. The probability that not a single packet of a train of size $\omega+k$ is dropped is given by $\pi_G(1-\alpha)^{\omega+k} + \pi_B\beta(1-\alpha)^{\omega+k-1}$. Hence, we have

$$F_c(\omega+k) = 1 - \pi_G(1-\alpha)^{\omega+k} - \pi_B\beta(1-\alpha)^{\omega+k-1}, \qquad \omega+k < W_m \tag{16}$$

while if $\omega+k = W_m$, then $F_c(\omega+k) = 1$ since a packet is surely dropped.

The congestion avoidance phase starts after $\omega - 1$ packets are successful. Hence, to reach a window of $\omega + k\ k \geq 0$, we require $s(\omega-1,k) = \omega - 1 + \omega + \omega + 1 + \cdots + \omega + k - 1$ successful packets. These $s(\omega-1,k)$ packets occupy $\lceil \log_2(\omega) \rceil + k$ trains. Hence, using an approach similar to (15), we have

$$S_c^\sigma(\omega+k) = (1-\alpha)^{s(\omega-1,k)} \left[\pi_G + \frac{\pi_B \beta}{1-\alpha} \right]^{\lceil \log_2(\omega) \rceil + k}. \tag{17}$$

Thus, the expression for the transition probability for the case $k \geq 0$ follows from (16) and (17). $\diamond$

Remark: Note that by letting $\beta = 1 - \alpha$ we obtain an i.i.d. packet loss model. The transition probability for such a case with $p = \alpha$ is given as

$$P(W_\rho = \omega + k / W_{\rho-1} = \sigma) = \begin{cases} q^{\omega+k-1}p, & -\omega < k < 0 \\ q^{s(\omega-1,k)}A & \\ q^{s(\omega-1,k)}, & k = W_m - \omega \end{cases}$$

where $A = [1-(1-p)^{\omega+k}]\ 0 \leq k < W_m - \omega$ and $q = 1 - p$.

We next consider the timeout probability calculation. Given a loss window of σ', we have

$$P_\tau = \sum_{\sigma'=1}^{W_m} P(\sigma')\tau_{\sigma'} \tag{18}$$

where $\tau_{\sigma'}$ is the probability of timeout in a cycle. Hence, we next proceed to specify the expression for $\tau_{\sigma'}$.

We would like to remark here that the starting point for the channel is the bad state. But we calculate the timeout probability assuming that all the packets in the loss window belong to the same train. Note that this is an approximation in the case whereby a packet is lost from the middle of a train in which case the entire loss window consists of packets from two different trains. For such a case, we also have to consider the channel state that the first packet of the second train encounters. This could be taken care of at the cost of extra complexity but we choose not to do so. Further, since this assumption is done for all the TCP algorithms, it does not affect the results.

Proposition 2: Timeout Probability Calculation: Consider TCP Tahoe during the ρth cycle. Let $\Omega > 0$ denote the number of duplicate acks on the receipt of which the sender enters the fast retransmit phase. Then $\tau_{\sigma'}$, the probability of timeout given that the loss window is σ', is given by

$$\tau_{\sigma'} = \sum_{i=\sigma'-\Omega}^{\sigma'-2} \sum_{m=1}^{\min(\sigma'-1-i,i+1)} \sum_{l=m-1}^{\min(m,i)} \binom{\sigma'-i-2}{\sigma'-1-i-m} \times \binom{i}{i-l} (1-\beta)^{i-l}\alpha^l\beta^m(1-\alpha)^{\sigma'-1-i-m} + (1-\beta)^{\sigma'-1}. \tag{19}$$

Proof: A timeout results if and only if less than Ω duplicate acks arrive at the sender. In a loss window of σ', one packet is already lost. Hence, a timeout occurs if and only if the number of packets lost after the first packet loss is greater than or equal to $\sigma' - \Omega$. Note that the maximum number of packets which can be lost at this point is $\sigma = \sigma' - 1$ since a packet is already lost in a window of σ'.

Let us consider the case of i drops. We remark here that a timeout results if and only if $i \geq \sigma' - \Omega$. Note that if $i = \sigma' - 1$, then probability of timeout is given by $(1-\beta)^\sigma$. Hence, we next consider $\sigma' - 1 > i$. These i drops are possible either by having the channel in a bad state at the beginning of a transmission and transitioning to the bad state itself, which happens with probability $1 - \beta$, or by having the channel in a good state at the beginning of a transmission and transitioning to a bad state, which happens with probability α. Let l indicate the number of times that the channel starts in a good state. Hence, we have the probability of i drops as

$$(1-\beta)^{i-l}\alpha^l. \tag{20}$$

i packet drops also imply that the rest of the packets are not dropped and result in duplicate acks. Again, two scenarios result here with the channel beginning in a good state or a bad state. Let m denote the number of times that the channel starts in the bad state. Hence, the probability of i drops implies that $\sigma - i$ packets get through which is given by the probability

$$\beta^m(1-\alpha)^{\sigma-i-m}. \tag{21}$$

The probability of one such way of a timeout occurring is $(1-\beta)^{i-l}\alpha^l\beta^m(1-\alpha)^{\sigma-i-m}$. $++\beta{*}{*}\alpha++\ \beta{*}{*}\alpha\cdots{*}{*}\alpha$ denotes the general form of a trace of packet successes or losses leading to a timeout with $1-\beta$ raised to appropriate power occupying the $++$ spaces and $1-\alpha$ raised to appropriate power occupying the $**$ spaces. Hence, we have $l = m$ or $l = m - 1$, i.e.

$$m \geq l \geq m - 1.$$

Further

$$\sigma - i - m \geq 0 \rightarrow \sigma - i \geq m \tag{22}$$

$$i - l \geq 0 \rightarrow i \geq l \tag{23}$$

$$\rightarrow i + 1 \geq m \geq l. \tag{24}$$

Thus, given i, l, and m, we next look at the number of possible combinations of a trace leading to a timeout. Consider the pair β and $1-\alpha$. Note that since we are starting with a lost packet, the pattern $(1-\alpha)^f f = 0, 1, 2, \ldots$ can occur only after β has occurred. Thus, β occurs m times, while the pattern $1 - \alpha$ can occur $\sigma - i - m$ times such that β precedes the trace. Hence, $m - 1$ β's each raised to power 1, can appear in any order with $\sigma - i - m$ repetitions of $1-\alpha$ with no restrictions on consecutive appearances of $1 - \alpha$. Thus, we are asking for the number of subpopulations of size $\sigma - i - m$ in a population of size $\sigma - i - m + m - 1$, which equals

$$\binom{\sigma - i - 1}{\sigma - i - m}. \tag{25}$$

Similarly, considering the pattern of α and $1-\beta$, we are asking for the number of subpopulations of size $i - l$ in a population of size $i - l + l$, which equals

$$\binom{i}{i-l} \tag{26}$$

since the first occurrence of α can be after $1-\beta$ has occurred. Thus, the total number of combinations is given by the product of (25) and (26). Hence, the probability of a timeout given i losses $\tau^i_{\sigma'}$, such that $\sigma > i \geq \sigma' - \Omega$, is given by

$$\tau^i_{\sigma'} = \sum_{m=1}^{\min(\sigma-i,i+1)} \sum_{l=m-1}^{\min(m,i)} \binom{\sigma-i-1}{\sigma-i-m} \times \binom{i}{i-l} (1-\beta)^{i-l}\alpha^l\beta^m(1-\alpha)^{\sigma-i-m}.$$

Thus, the probability of a timeout given a loss window size of σ' then follows as

$$\tau_{\sigma'} = \sum_{i=\sigma'-\Omega}^{\sigma-1} \sum_{m=1}^{\min(\sigma-i,i+1)} \sum_{l=m-1}^{\min(m,i)} \binom{\sigma-i-1}{\sigma-i-m} \times \binom{i}{i-l} (1-\beta)^{i-l}\alpha^l\beta^m(1-\alpha)^{\sigma-i-m} + (1-\beta)^{\sigma}.$$

$\diamond$

It is to be remarked that in case of OldTahoe, every packet loss is accompanied by a timeout and, hence, $\tau_{\sigma'} = 1\, \forall\sigma'$.

Proposition 3: Train Probability Calculation: Consider TCP Tahoe or OldTahoe. Then the probabilities of the different trains is given as

$$\begin{aligned} &P(n_1 = k/\sigma) \\ &\quad = (1-\alpha)^{\theta_1+\theta_2+\cdots+\theta_{k-1}} \left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{k-1} \\ &\quad\quad \times [1 - \psi_{1k} - \psi_{2k}] \quad 0 < k < t_m \\ &P(n_1 = t_m/\sigma) \\ &\quad = (1-\alpha)^{\theta_1+\theta_2+\cdots+\theta_{t_m-1}} \left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{t_m-1} \end{aligned}$$

where

$$\begin{aligned} \psi_{1j} &= \pi_G(1-\alpha)^{\theta_j} \\ \psi_{2j} &= \pi_B\beta(1-\alpha)^{\theta_j-1} \\ t_m &= W_m - \omega + \lceil\log_2(\omega)\rceil + 1. \end{aligned} \tag{27}$$

Proof: For n_1 to be k, we require that $k-1$ trains experience no packet loss while there is at least one packet lost in the kth train. Since the number of packets in the jth train is given to be θ_j, we have following the remarks made in the proof of the loss window probability calculation, that

$$\begin{aligned} &P(n_1 = k/\sigma) \\ &\quad = (1-\alpha)^{\sum_{i=1}^{k-1}\theta_i} \\ &\quad\quad \times \left[\sum_{i=1}^{k-1} \binom{k-1}{i} \pi_G^{k-i-1} \left(\pi_B\beta(1-\alpha)^{-1}\right)^i\right] \\ &\quad\quad \times \left\{1 - \pi_G(1-\alpha)^{\theta_k} - \pi_B\beta(1-\alpha)^{\theta_k-1}\right\}, \quad 0 < k < t_m \\ &\quad = (1-\alpha)^{\sum_{i=1}^{k-1}\theta_i} \left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{k-1} \\ &\quad\quad \times \{1 - \pi_G(1-\alpha)^{\theta_k} - \pi_B\beta(1-\alpha)^{\theta_k-1}\}. \end{aligned} \tag{28}$$

When $k = t_m$, a packet is lost from the t_mth train with probability 1 and, hence

$$\begin{aligned} &P(n_1 = t_m/\sigma) \\ &\quad = (1-\alpha)^{\theta_1+\theta_2+\cdots+\theta_{t_m-1}} \left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{t_m-1}. \end{aligned} \tag{29}$$

$\diamond$

Calculating $E(n_1)$ given the above probability distribution can then be done as

$$E(n_1) = E[E(n_1/\sigma)] \tag{30}$$

$$= \sum_{\sigma} P(\sigma) \sum_j jP(n_1 = j/\sigma). \tag{31}$$

Remark: t_m in this case denotes the maximum number of trains possible in a cycle.

Remark: Note that $n_2 = 0$ w.p. 1 in case of Tahoe and OldTahoe.

Proposition 4: Packet Count Calculation: Consider TCP Tahoe or OldTahoe. The number of packets during a cycle given the number of trains and the loss window during the previous cycle is given by

$$E(Q/n_1 = k, \sigma) = \theta_1 + \cdots + \theta_k, \quad 0 < k \leq t_m; \quad 0 < \sigma \leq W_m$$

where

$$\begin{aligned} \theta_1 &= 1 \\ \theta_{j-1} &= 2^{j-2}\theta_1 < \omega \leq 2^{j-1}\theta_1 \\ \theta_j &= \omega = \lfloor\sigma/2\rfloor \quad \text{where } j = \lceil\log_2(\omega)\rceil + 1 \\ \theta_{j+l} &= \theta_j + l \;\; 0 < l \leq t_m - j. \end{aligned}$$

Proof: Note that every cycle of Tahoe starts with one packet, i.e., $\theta_1 = 1$. The packets in a train keep on doubling as long as the window size is less than the slow start threshold. Following this, the number of packets in a train increases by one in each successive train. Let the slow start threshold be reached in the jth train. Hence, $j = \lceil\log_2(\omega)\rceil + 1$ and the maximum size of a train is W_m. With this, the expressions for the terms given above as well as for $E(Q/n_1, \sigma)$ are obvious.$\diamond$

V. Performance Study

Now that the loss window probability, train probability, packet count, and timeout probability are specified, we use the expressions in (4) and (10) to calculate the throughput using (3). Since it is difficult to obtain a closed-form solution for the throughput, we graph the different expressions given in order to obtain an understanding of the way TCP versions work over a wireless link with correlated losses. We consider all the TCP algorithms, namely, OldTahoe, Tahoe, NewReno, and Sack, in this section. The different parameters that we consider while studying the behavior of the TCP versions under correlated losses are α, β, packet size S (bytes), link bandwidth μ (Mb/s), timeout value(s) and RTT time(s). In order to validate our conclusions, we also use simulations. We use the widely used simulation package ns2 [16]. When using simulations, we obtain the throughput as an average over two trials, each

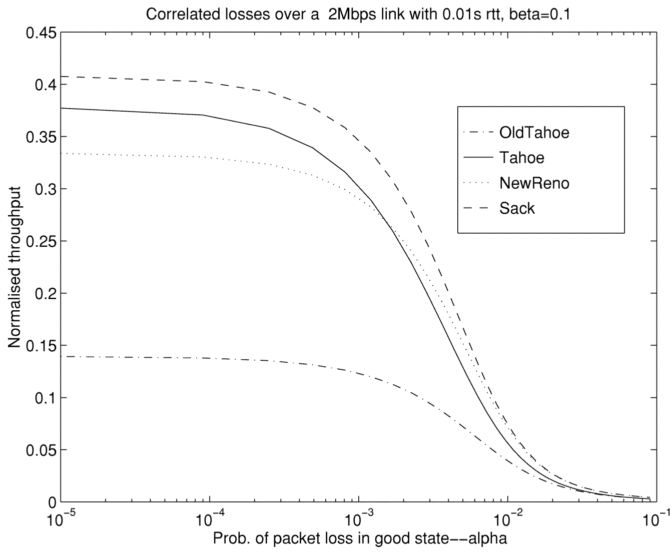

Fig. 2. Behavior of different TCP algorithms with correlated packet loss model under bursty loss conditions over a link with a bandwidth delay product of 20 packets.

trial lasting for a duration of 50 s. The default values of the parameters are assumed to be $S = 125$ bytes or $S = 500$ bytes, timer granularity of 0.5 s for coarse timeout and 0.05 s for fine timeout [9] while $\Omega = 3$.

We next investigate the performance of the different TCP algorithms over wireless links using both the packet train model as well as simulations. Consider Fig. 2 which illustrates the performance of a wireless link with a bandwidth of 2 Mb/s and an RTT of 0.01 s using the packet train model. We assumed a packet size of 125 bytes here resulting in a bandwidth delay product of 20 packets. β is assumed to be 0.1 while α is varied and shown on the x axis. The throughput normalized to the link bandwidth is shown on the y axis.

An important observation is that for low values of β, NewReno performs worse than Tahoe. This is because of the nature of NewReno of taking one RTT to recover each lost packet which leads to a smaller throughput than achievable by Tahoe. For example, in a window of 10, assuming five bursty packet losses (that is, five consecutive packets lost) in a loss window of 9 or more a timeout does not occur since there are enough successful packets to inform the sender of packet losses through the duplicate ack mechanism. In this scenario, NewReno takes five RTTs to recover from these five packet losses by sending one lost and at most two packets in each RTT. In contrast, Tahoe, by resorting to slow start, recovers these packets more efficiently by sending more number of packets on the average. We show an example of one such possibility in Tables II and III. As shown, when five consecutive packets 15–19 are lost, Tahoe recovers these packets within three RTTs while NewReno takes five RTTs to recover these packets, thereby causing a higher inefficiency in the link utilization. Of course, the drawback may be that Tahoe had had to retransmit some packets unneccessarily, yet in terms of throughput the Tahoe sender is more efficient than the NewReno sender. Thus, Tahoe is able to send 14 packets not present at the receiver during these five RTTs while NewReno is able to send only five packets not present at the receiver during these five RTTs.

TABLE II
EVOLUTION OF TAHOE WHEN PACKETS 15–19 ARE LOST TAKING THREE RTTs TO RECOVER THE LOST PACKETS

Pkt info in ack	Ack for pkt	ω	W	Pkt sent
14	14	-	8	22
14 I dup ack	20	-	8	-
14 II dup ack	21	-	8	-
14 III dup ack	22	4	1	15
15	15	4	2	16,17
16	16	4	3	18,19
17	17	4	4	20,21
18	18	4	4.25	22
22	19	4	4.5	23,24,25,26

TABLE III
FAST RECOVERY FOR NEW RENO WHEN PACKETS 15–19 (INCLUSIVE) ARE LOST TAKING FIVE RTTs TO RECOVER THE LOST PACKETS

Pkt info in ack	Ack for pkt	ω	W	Pkt sent
14	14	-	8	22
14 I dup ack	20	-	8	-
14 II dup ack	21	-	8	-
14 III dup ack	22	4	7	15
15 (Partial ack)	15	4	4	16
16 (Par ack)	16	4	4	17
17 (Par ack)	17	4	4	18
18 (Par ack)	18	4	4	19
22 (Par ack)	19	4	4	23,24,25,26

Of course, Sack, by virtue of the selective ack option, recovers most efficiently and, hence, exhibits the best performance. Note also that as the maximum possible window size W_m grows larger, the performance of NewReno lags behind the performance of Tahoe by a larger degree. This is because the size of the bursts may not be enough to cause a timeout due to the large loss window leading to lost packet recovery through the inefficient fast recovery method of NewReno. As a corollary, this implies that at higher values of α, the performance of NewReno can be better compared to the performance of Tahoe since at high values of α the maximum possible value to which the window can grow to is limited, thereby masking the weakness of NewReno's fast recovery mechanism. We can observe this from Fig. 2.

It is to be remarked here that if packet losses were more staggered like in the i.i.d. model, NewReno, while being constrained to send at most one lost packet per RTT, could also send more than two packets (the other packets carrying new data) per RTT. This plus the nature of Tahoe of retransmitting even the packets successfully received in such a scenario ensures that the performance of NewReno is better than the performance of Tahoe in an i.i.d. loss regime (causing staggered packet losses).

In order to verify this, we simulate a scenario with a 2-Mb/s link with an RTT of 0.02 s. β is assumed to be 0.2 and the packet size is 500 bytes. The resulting performance of Tahoe, NewReno, and Sack is shown in Fig. 3. We again see the better performance of Tahoe as compared to New Reno. Note also the better performance of all the algorithms for low values of α as compared to the previous figures. This is due to the higher value of β, low value of the bandwidth-delay product as well as due to performance of the packet train model being less optimistic for low values of α. We have also verified this behavior in other

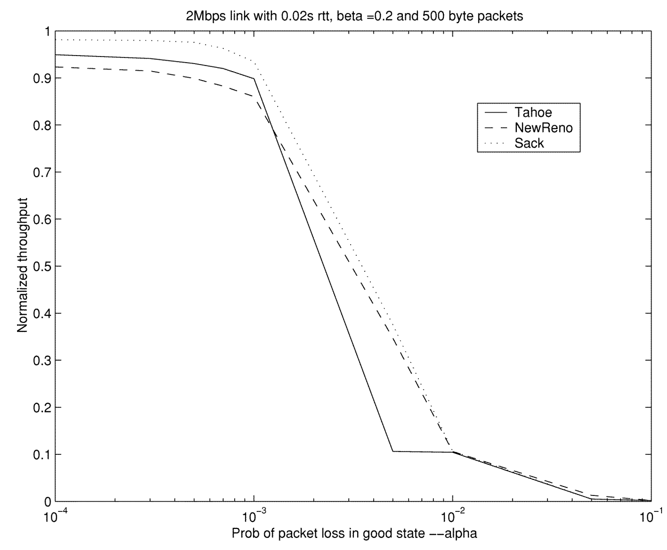

Fig. 3. Illustrating the performance of various TCP algorithms using simulations.

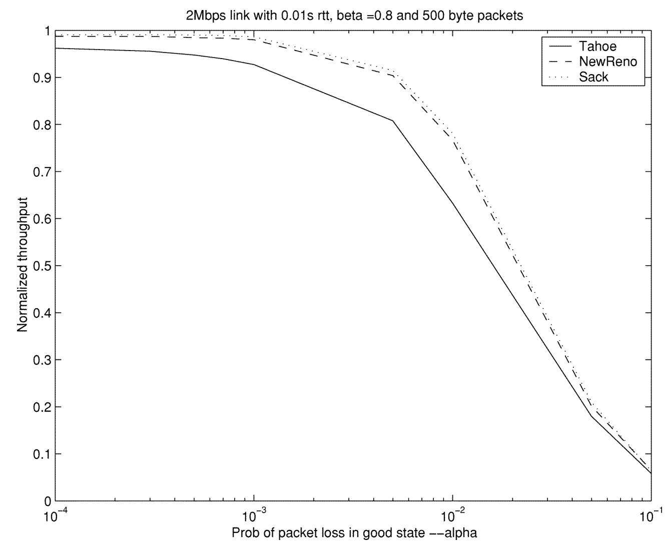

Fig. 4. Illustrating the performance of various TCP algorithms using simulations for less deleterious bursty conditions.

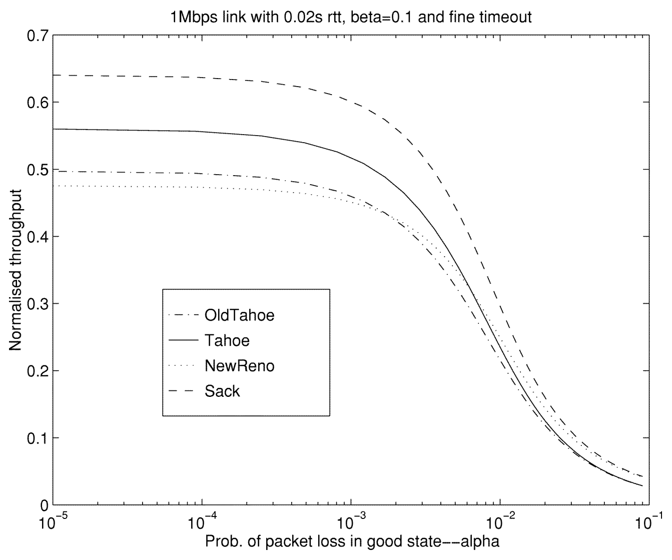

Fig. 5. Illustrating that at low values of β the performance depends only on the bandwidth-delay product when using a fine timeout interval.

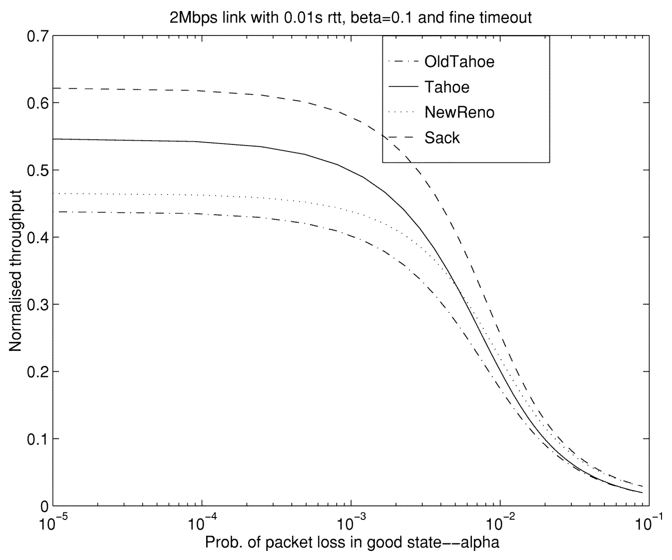

Fig. 6. Illustrating that at low values of β the performance depends only on the bandwidth-delay product when using a fine timeout interval.

scenarios. A similar scenario with $\beta = 0.8$ is shown in Fig. 4. It can be seen from these figures that as the value of β increases, the performance of the different TCP algorithms improves.

We also see from these figures that as α increases, the performance of all the algorithms decreases. Note that the reciprocal of β gives the average number of packets lost while the reciprocal of α gives the average number of good packets. Hence, as β increases, the probability of timeout decreases and the performance is hence better. At moderate values of β, there are more chances of multiple packet drops in a window while at high values of β around one, generally only single packet drops occur.

We have also observed that the normalized throughput becomes better for low values of β as the RTT increases. This is because of the large bandwidth-delay product link which implies that even with $\beta = 0.1$, the chances of an entire window of packets being dropped for window sizes high enough are quite low. Further, the timeout duration also becomes a smaller multiple of the RTT as the RTT increases and, hence, the use of coarse timeout becomes less significant. Thus, it can be inferred that at low values of β the performance does not depend on just the bandwidth-delay product but also on the value of the RTT. This is not the case for high values of β. This is because at high β values, Tahoe, NewReno, and Sack experience timeouts very infrequently, leading to insensitivity to the granularity of the timeout interval. Thus, it can be expected and it has also been verified that at relatively high values of β, fine timeout does not make much of a difference compared with the coarse timeout. (In this case, each tick of the TCP clock is much smaller than the coarse timeout granularity of 500 ms.)

To show that the performance difference is indeed due to the coarse timeout granularity for low values of β, in Figs. 5 and 6, we consider link bandwidths of 1 and 2 Mb/s, respectively, while ensuring that the bandwidth delay products remain the same at 20 packets by properly choosing the RTT values. For these two figures, we use the packet train model assuming a packet size of 125 bytes. Comparing these two figures, it can be seen that the performance of the different TCP versions is nearly similar, with the performance becoming identical as the timeout granularity is decreased further. Another point to be noted is that as the granularity of the timeout interval becomes finer and less significant with respect to the RTT, the behavior gap of Tahoe and OldTahoe decreases and, hence, NewReno exhibits the worst performance of all the TCP versions, as expected in this regime.

We next obtain approximate conditions under which random packet losses lead to significant throughput deterioration. There are two factors that have to be taken into consideration. The first is α, which governs the maximum window size possible in each cycle. The second factor is β, which governs the probability of a timeout.

Concentrating on the first factor, every cycle can have at most t_m trains, which is a function of W_m and σ. The duration of each train is T seconds, which is the RTT. Hence, for good performance, noting that the duration of each slot equals the transmission time of a packet, we require

$$\begin{aligned}\frac{1}{\alpha} &\geq t_m \cdot T \cdot \text{ (number of packets transmitted per unit time)}\\ &\geq t_m T \mu. \end{aligned} \tag{32}$$

This follows since the reciprocal of α denotes the mean number of successful packets in a cycle, and we desire that the mean duration of the good period exceed the duration of an entire cycle. It is quite obvious that by ensuring this, not only does the window grow to the maximum possible size W_m, but also the next cycle begins with a high slow start threshold which is desirable. Now, since t_m is proportional to W_m, approximating t_m by W_m, we have a necessary condition for good behavior that

$$\frac{1}{\alpha} \geq (\mu T)^2.$$

Looking at the second factor next, for good performance a necessary condition is that

$$\frac{1}{\beta} < \mu T/2.$$

This follows since the reciprocal of β denotes the mean number of packets lost in the loss window. If this is comparable to the size of the loss window, then it will result in a timeout which has to be avoided for good performance. Hence, we require that this quantity be smaller than the average window size assuming that the window can grow to its maximum possible values. It has to be remarked here that this condition would not be necessary if the connection were using fine timeout values since in that case the effect of a timeout is not severe at all.

We verify the above conditions in Fig. 7. This simulation result considers a 1-Mb/s link with a one-way delay of 0.05 s, $\beta = 0.2$, and 500-byte packets. Thus, this results in a link bandwidth delay of 25 packets. Hence, the two necessary conditions for good performance translate into $\alpha \leq 0.0016$ and $\beta > 0.08$. From the figure, we see that the link utilization is greater than 60% in the regime where these conditions are satisified for all

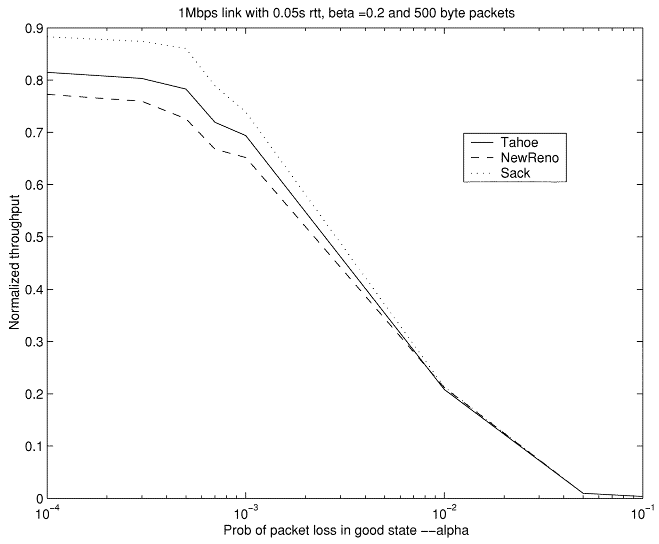

Fig. 7. Illustrating the applicability of conditions which lead to significant throughput deterioration.

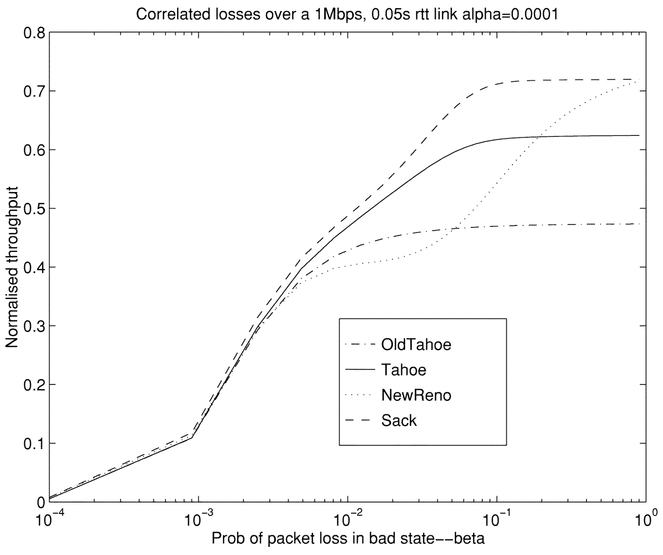

Fig. 8. Illustrating the effects of varying β for a link with a bandwidth-delay product of 50 packets.

the TCP algorithms. We also see the performance lag in case of NewReno as compared with Tahoe.

We next investigate the effects of β while keeping α constant. These are shown in Figs. 8 and 9, both of which are based on the packet train model. For both these figures, we also assume a packet size of 125 bytes. In Fig. 8, we consider a scenario with a bandwidth-delay product of 50 packets and a wireless link bandwidth of 1 Mb/s. At very low values of β, timeouts are the norm and, hence, the performance of all TCP versions is similar. The robustness of Sack and Tahoe to values of β above a threshold, as remarked earlier, is also seen. This is because Sack and Tahoe recover the same way for single and multiple packet drops (with Sack resorting to fast recovery in both cases and Tahoe going through slow start for both scenarios), they exhibit similar behavior for both moderate and high values of β. In contrast, NewReno recovers from multiple packet drops by retransmitting one lost packet per RTT and, hence, its behavior improves continually as β increases, until at high values of β the

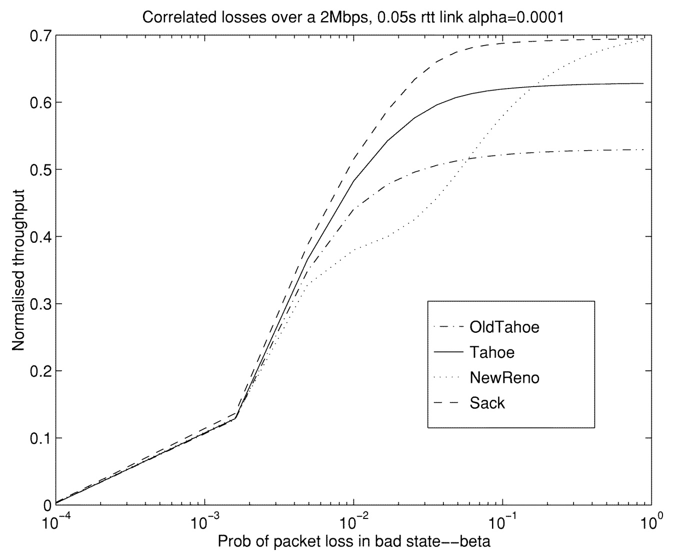

Fig. 9. Illustrating the effects of varying β for a link with a bandwidth-delay product of 100 packets.

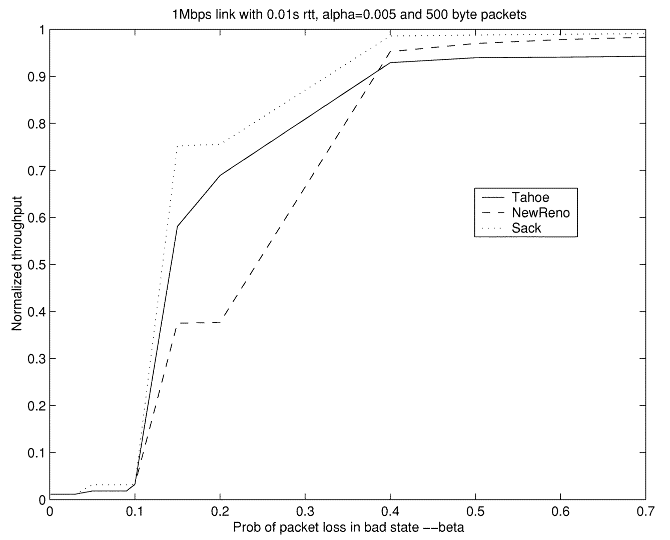

Fig. 10. Illustrating the effects of varying β using simulations for a low bandwidth-delay product link.

performance of NewReno and Sack becomes very similar. From Fig. 9, which considers a link with a bandwidth delay product of 100 packets and a link bandwidth of 2 Mb/s, it can be seen that the minimum robustness threshold decreases as the bandwidth-delay product increases. The performance of NewReno, however, is very sensitive to the value of β. It can also be seen that at high values of the RTT with a corresponding high value of the bandwidth delay product, the performance of NewReno is also the worst of all the TCP versions under bursty loss conditions, as mentioned earlier. The performance is also seen to become better when $\beta > 0.02$, since $\alpha \leq 0.0001$, which are the two conditions for a link with a bandwidth delay of 100.

In Fig. 10, we use simulations to show the effects of β while keeping α constant for a low bandwidth-delay link. The conditions for good link utilization translate to $\alpha \leq 0.04$ and $\beta > 0.4$. Thus, the minimum robustness threshold increases as the bandwidth-delay product decreases. This is also seen by comparing this figure with the earlier figures. We also see the performance deterioration of NewReno for low and moderate values of β.

VI. Conclusion

In this paper, we have looked at the behavior of the different TCP algorithms over a wireless channel with correlated packet losses. Like the earlier analytical studies and many of the simulation studies, we have been concerned with just a single TCP connection. In addition to modeling of the interaction between multiple TCP flows being notoriously difficult, the main reason for this is to develop insight into the interaction between random packet losses and the TCP dynamic window adjusting mechanism.

We first provide an analytical model for studying the performance of the different TCP algorithms, namely, OldTahoe, Tahoe, NewReno, and Sack, operating over a wireless link with correlated packet losses. We then provide conditions on the wireless channel satisfaction of which ensures that the throughput of the TCP algorithm tends to the best possible throughput. We see that the behavior of Sack is the best in all regimes. Another important result that we have shown is that for situations of even moderately bursty losses, the performance of NewReno is worse than Tahoe with the performance gap widening with higher values of W_m. This is a serious flaw in the performance of NewReno, which also argues for the widespread implementation of Sack. Also, at values of high α, the performance difference between the different versions decreases with the difference becoming insignificant as the value of β decreases. It is also seen that Sack and Tahoe are insensitive to the value of β as long as β is not low enough, while NewReno's performance improves continually as β increases. This implies that Sack and Tahoe are less sensitive to the bursty conditions above a certain threshold.

We have also shown that performance of the different TCP versions under correlated packet loss depends not only on the bandwidth delay product but also on the granularity of the timeout timer for low values of β. For high values of β, the performance depends just on the bandwidth delay product. Further, it is also seen that reducing the granularity of the timeout interval as also reducing the value of the fast retransmit threshold makes a difference only in case of very bursty loss conditions and in scenarios where the window cannot grow to large sizes for high values of β. We have also shown that at very high bursty loss conditions the performance of all the TCP versions is similar.

References

[1] F. M. Anjum, "Analysis and design of a reliable wireless transport protocol and fair network mechanisms for the Internet," Ph.D. dissertation, Univ. Maryland, College Park, May 1999.

[2] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, pp. 756–769, Dec. 1997.

[3] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance over wireless networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, Dec. 1995.

[4] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environment," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 850–857, June 1995.

[5] A. Chockalingam, M. Zorzi, and R. R. Rao, "Performance of TCP on wireless fading links with memory," in *Proc. IEEE Int. Conf. Communications*, June 1998, pp. 595–600.

[6] K. Fall and S. Floyd. (1996, Mar.) Comparisions of Tahoe, Reno and Sack TCP. [Online]. Available: ftp://ftp.ee.lbl.gov

[7] Z. J. Haas and P. Agrawal, "Mobile-TCP: An asymmetric transport protocol design for mobile systems," in *Proc. IEEE Int. Conf. Communications*, vol. 2, Montreal, QC, Canada, June 1997, pp. 1054–1058.

[8] W. C. Jakes Jr., *Microwave Mobile Communications*. New York: Wiley, 1974.

[9] A. Kumar, "Comparative performance analysis of versions of TCP in local network with a lossy link," Rutgers Univ., New Brunswick, NJ, Tech. Rep. WINLAB-TR 129, Oct. 1996.

[10] A. Kumar and J. Holtzmann, "Comparative performance analysis of versions of TCP in local network with a lossy link—Part II: Rayleigh Fading Mobile Radio Link," Rutgers Univ., New Brunswick, NJ, Tech. Rep. WINLAB-TR 133, Nov. 1996.

[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. Networking*, vol. 5, pp. 336–350, June 1997.

[12] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, July 1997.

[13] P. P. Mishra, D. Sanghi, and S. K. Tripathi, "TCP flow control in lossy networks: Analysis and enhancements," in *Proc. IFIP TC6 Working Conf. Computer Networks, Architecture and Applications (NETWORKS'92)*. New York: North-Holland, 1993, pp. 181–193.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM*, 1998, pp. 303–314.

[15] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. K. Tripathi, "Using channel state dependent scheduling to improve TCP throughput over wireless LANs," *ACM Wireless Networks*, vol. 3, no. 1, pp. 91–102, Mar. 1995.

[16] UCL/LBNL/VINT Network Simulator, ns version 2. [Online]. Available: http://www-mash.cs.berkeley.edu/ns/

[17] W. R. Stevens, *TCP/IP Illustrated*. Reading, MA: Addison-Wesley, 1994, vol. 1.

[18] T. V. Lakshman, U. Madhow, and B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: A study of TCP/IP performance," in *Proc. IEEE INFOCOM*, vol. 3, Apr. 1997, pp. 1199–1209.

[19] K. Wang and S. K. Tripathi, "Mobile-end transport protocol: An alternative to TCP/IP over wireless links," in *Proc. IEEE INFOCOM*, Apr. 1998, pp. 1046–1053.

[20] M. Zorzi and R. R. Rao, "Effect of correlated errors on TCP," in *Proc. Conf. Information Sciences and Systems*, Mar. 1997, pp. 666–671.

Farooq Anjum (M'01) received the Ph.D. degree in electrical and computer engineering from the University of Maryland, College Park, in 1999.

He is currently a Research Scientist with Telcordia Technologies, Morristown, NJ, where his research interests include network security, middleware technologies, and quality-of-service issues in wireless networks. He leads a Java Call Control Advanced (JCAT) group within the Java APIs for Advanced Integrated Networks (JAIN) consortium. He is also an Adjunct Professor at the University of Pennsylvania, Philadelphia, and the Stevens Institute of Technology, Hoboken, NJ.

Leandros Tassiulas received the Diploma in electrical engineering from the Aristotelian University of Thessaloniki, Thessaloniki, Greece, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, in 1989 and 1991, respectively.

From 1991 to 1995, he was an Assistant Professor in the Department of Electrical Engineering, Polytechnic University, Brooklyn, NY. In 1995, he joined the Department of Electrical Engineering, University of Maryland, where he is currently an Associate Professor. He holds a joint appointment with the Institute for Systems Research, University of Maryland, and is a member of the Center for Satellite and Hybrid Communication Networks, established by NASA. His research interests are in all aspects of communications with a focus on wireless networks.

Dr. Tassiulas received a National Science Foundation (NSF) Research Initiation Award in 1992, an NSF Faculty Early Career Development Award in 1995, and an Office of Naval Research Young Investigator Award in 1997. He coauthored a paper that received the IEEE INFOCOM'94 Best Paper Award. He is an Associate Editor for IEEE TRANSACTIONS ON INFORMATION THEORY and has served on the Editorial Board of the IEEE/ACM TRANSACTIONS ON NETWORKING.