Functioning of TCP Algorithms over a Wireless Link

by Farooq M. Anjum, Leandros Tassiulas

CSHCN T.R. 99-10
(ISR T.R. 99-20)

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **1999**	2. REPORT TYPE	3. DATES COVERED **-**

4. TITLE AND SUBTITLE
Functioning of TCP Algorithms over a Wirelesss Link

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research,875 North Randolph Street,Arlington,VA,22203-1768

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**40**	

Functioning of TCP Algorithms over a Wireless Link [*†]

Farooq Anjum and Leandros Tassiulas
ISR and Dept. Of Electrical Engg
Univ. Of Maryland, College Park.

March 28, 1999

Abstract

In this paper, we investigate the behavior of the various algorithms of TCP, the internet data transport protocol, 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 on account 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 1 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 whether coarse or fine. We have also investigated the effects of reducing the fast retransmit threshold.

1 Introduction

The transport protocol used by many internet based applications like http, ftp, telnet etc. is TCP (transmission control protocol). 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

[*] This research was supported in part by NSF under a CAREER award NCR-9502614 and by the AFOSR under grant 95-1-0061.

[†] A partial version of this paper appeared in Sigmetrics99 Atlanta.

TCP works is that it keeps increasing the sending rate of packets as long as no packets are lost. When packet losses occur, for e.g. due to the network becoming congested, TCP decreases the sending rate. Thus, basically 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 also, 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 the wireless link [16, 2, 3, 14, 4, 7]. Because of the difficulty of modelling 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 the TCP protocol 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 towards the design of a better transport protocol for the wireless networks has to be a better understanding of the way TCP works over the wireless links. This would reveal the reasons for the inefficiency of TCP over wireless links. In order to get a deeper understanding of the way that TCP works over wireless links there has been some effort recently on the analytical study of TCP over wireless links [10, 11, 17, 12]. Typically, the random losses on the wireless link are modeled using two different methods. In case of a simple model called the iid model, it can be assumed that each packet is lost with a certain constant probability independent of the other packets, which corresponds to a Bernoulli loss model. The second model namely the correlated packet loss model is more realistic and it does not model the packet losses as being independent of each other but does address the correlation between the losses of different packets.

[10] deals with the effects of iid packet losses on TCP performance. 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. But contrary to the actual situation, 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 ack causes the window to be incremented by 1 with

a certain probability. Note that both these assumptions are untrue but have to be resorted to in the approach taken by them 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] only OldTahoe is considered over a link with iid losses. [12] also considers Tahoe and Reno in a regime where the bandwidth-delay product of the network is high compared to the buffering in the network. The key result that they provide is the characterization of a condition under which the throughput over the lossy link degrades significantly. 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.

[11] considers the behavior of TCP Tahoe and OldTahoe in the presence of correlated packet losses. The results obtained though are applicable only in case of very low bandwidth wireless links. More emphasis is also placed on analyzing a link layer solution, of hiding the losses from the transport layer 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 [10] and hence suffers from the drawbacks noted earlier. [17] which deals with TCP OldTahoe assuming a correlated loss model, also fails to capture many interesting phenomena. 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 case of the analysis of links with large bandwidth delay products since in this case large number 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.

It should be pointed out that all the analytical studies and many of the simulation studies reported 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. Note that while considering single connections if one were to ignore the congestion control functions then the solution would be trivial. Some form of selective repeat ARQ with a window large enough to exceed the bandwidth delay product of the link would be the best solution. It is quite obvious that such a scheme would be a disaster when many connections are considered. Further, with the goal being to use the results of studies involving single flows to design improvements in TCP for use by multiple users, congestion control functions would also have to be considered and hence the above trivial solution is ruled out.

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 of future work. In this paper 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. Investigation into the effects of reducing the fast retransmit threshold as well as using finer timeout intervals is also carried out and we show that these incremental changes to the TCP versions are effective only in case of lossy conditions.

The plan of the paper is as follows. In section 2, we describe briefly the various TCP versions. In section 3, we specify the correlated loss model in detail. In section 4, 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 5 we characterize the throughput of the different TCP algorithms as a function of the wireless channel parameters. Section 6 deals with the analytical study of the performance of the different TCP versions under various conditions. Finally conclusions are presented in Section 7.

2 TCP Versions

We next look briefly at the way the receiver and transmitter function in case of a transport protocol like the TCP. The receiver sends back an ack for every packet received from the transmitter [*]. The

[*] For ease of description we ignore variations like acknowledging every other packet

acks convey information about the next expected packet sequence that the receiver expects. Note that if packet m is lost and packets after m get through, then the receiver sends back duplicate acks all conveying information that the receiver expects packet with sequence number m. The number of duplicate acks depends on the number of packets after m that the receiver gets. The acks are cumulative in that an ack carrying sequence number k, acknowledges all data packets upto k. The receiver presents packets in sequence to the user. Hence any packets received out of sequence are stored in a buffer maintained by the receiver with a maximum size corresponding to W_{rec} packets. The acks also carry information about the size of the buffer at the receiver. The transmitter window strategy guarantees that no more than W_{rec} packets will be in transit.

The transmitter operates on a window based transmission strategy. At any time t, the transmitter knows the packets that have been successfully acked by the receiver. Let $A(t)$ correspond to a window edge of packets such that acks for all packets to the left of $A(t)$ have been received at the sender while the ack for the packet to the immediate right of $A(t)$ has not been received by the sender at time t. Further, the transmitter also has a window $W(t)$ associated at time t such that only $A(t) + W(t)$ packets are allowed to be outstanding (without any acks received for those packets) at time t. On the receipt of acks, $A(t)$ advances while the window adaptation policy followed causes $W(t)$ to increase in a predetermined fashion. Loss of packets is detected by the transmitter either by a timeout or by the receipt of duplicate acks sent by the receiver. In the wireline network loss of packets typically signals congestion. Hence, on detection of a packet loss, the window $W(t)$ is decreased in a fashion determined by the window adaptation policy and transmission restarts by resending the lost packet. Note that $A(t)$ does not advance in such a situation. Note also that $W(t)$ cannot increase beyond a minimum of W_{rec} or W_m where W_m is the maximum possible window size allowable on account of the constraint on the sum of the bandwidth delay product and the buffer size. In the sequel, we assume that only W_m restricts the window size while W_{rec} does not, i.e. $W_{rec} > W_m$. The case where $W_{rec} \leq W_m$ can also be addressed using our approach but we do not do it here for lack of space.

Thus as can be seen, the window adaptation procedure depends on the policy followed. Based on the policy followed, we have different TCP algorithms. We next explain the different TCP algorithms that we study. Each differs on the response in the form of window size decrease to one or more packet drops and the fashion of window increases on no packet drops. The algorithms are of 5 types namely, TCP OldTahoe, TCP Tahoe, TCP Reno, TCP NewReno and TCP Sack.

There are also other TCP algorithms like TCP Vegas but we do not concern ourselves with them in the rest of the paper and hence do not describe them here. The window adaptation procedure for the different TCP versions was originally proposed and developed by Van Jacobson [8]. The description next follows that of [6] and [12].

Data exchange using a transport protocol involves a session set up phase, a data transfer phase and a session tear down phase. Since, in this paper we are interested only in the bulk throughput performance, we describe and model only the data transfer phase of the protocol which dominates the resulting performance. We first describe the basic window adaptation procedure common to all TCP versions and then look at the specific congestion control and data loss recovery protocols implemented in the different TCP versions. At each time t, we denote the transmitter's congestion window by $W(t)$ and a threshold called the slow start threshold by $\omega(t)$. These are defined for all the protocol versions at all times. Hence, we have

- if $W(t) < \omega(t)$, each new ack from the receiver results in $W(t)$ being incremented by 1. This constitutes the **slow start phase**. Thus this results in the window size doubling every round trip time during this phase.

- if $W(t) \geq \omega(t)$, each ack for a new packet results in $W(t)$ being increased by $1/\lfloor W(t) \rfloor$ as long as the congestion window $W(t)$ is less than the receiver window. If $W(t)$ equals the receiver window size, then it remains unchanged on receipt of a new ack. This constitutes the **congestion avoidance phase.** Thus, this results in the window size increasing by one every round trip time during this phase provided the congestion window size is less than the receiver window size .

- if timeout occurs at the transmitter, then set $W(t^+) = 1$ and $\omega(t^+) = \lfloor W(t)/2 \rfloor$ and retransmission starts from the first lost packet detected

If a packet is lost or damaged while in transit from the sender to the receiver then no ack corresponding to this packet is issued and this packet will have to be retransmitted. But in order to determine when to retransmit a timer is associated with every packet when it is sent. If the timer expires before the packet is acknowledged, then the sender must retransmit. But a problem that arises is how to determine the value of the timer. Using a very small value as well as using a very large value for the retransmission timer both create problems. The approach normally taken is to have an adaptive scheme whereby an exponential smoothing technique is used to estimate the round trip time of the connection. The variability in the estimate in the form of mean deviation is then used along with the estimated round trip time to arrive at a dynamic estimate of the value of the retransmit timer [8]. Further, many implementations then convert the calculated round trip time in terms of a coarse timer granularity of 0.5 seconds. Hence, it is expected that whenever a packet is not acked within this interval the probability that it is lost is far greater than the probability that it is delayed for so long and in the rest of this work we do neglect the latter possibility. Thus, timeouts always imply packet loss. Further, we also take the approach that only one retransmission timer is maintained for an entire window of packets. If an ack is received then the appropriate packets are removed from the active window and the timer is reset. If the timer goes off without the expected ack being received, then retransmit the packet at the front of the window and reset the timer. Of course, it has to be remarked that the TCP standard does allow other possible implementations but we do not consider those either because the other possibilities are more complex to implement or because the other possibilities are less efficient than the option that we consider. We next provide version specific details of the different TCP versions.

TCP Old Tahoe:[8] The window adaptation is as given above. A lost packet is always detected by timeout. Hence, after a packet loss the transmitter has to wait for the timeout which is generally coarse (timer interval in increments of 500ms).

TCP Tahoe: [6] In contrast to the case of Old Tahoe where every packet loss is detected by a timeout, in Tahoe if the number of duplicate acks equalling Ω are received by the sender, the sender behaves as if timeout has occurred and begins retransmission of the packet perceived to be lost. It is to be remarked here that a duplicate ack signifies either that the packet following the correctly acked packet was delayed so that it ultimately arrived out of order or that the packet was lost. In the former case the packet does ultimately reach the receiver and the sender should not retransmit this packet. In the latter case the arrival of a duplicate ack serves as an early warning that a packet is lost. In order to make sure that we have a case of a lost packet rather than a reordered packet Jacobson [8] recommends that a TCP sender wait until it receives three duplicate acks to the same packet. Thus in practical implementations the value of Ω is taken as 3. This is because it is seen that for such a value of Ω, it is highly likely that the following packet is lost and not reordered.

The sender window and slow start threshold are as given above. This method of detecting packet loss on the basis of the duplicate acks is called the fast retransmit procedure. As is obvious, this can lead to higher channel utilization and better throughput which can be especially significant when dealing with coarse timers. Note that the strategy followed to recover from lost data in case of TCP Tahoe is to retransmit packets that might already have been successfully delivered.

TCP Reno: [6] This version modified the fast retransmit operation to include fast recovery. Once the packet loss is detected by the fast retransmit procedure, the sender retransmits the packet deemed to be lost and reduces its congestion window by half. Now, instead of entering the slow-start phase, the sender uses additional incoming acks to clock subsequent outgoing packets. Thus, if the packet loss was at a window of σ, then $\omega(t^+) = \lfloor \sigma/2 \rfloor$ and $W(t^+) = \omega(t^+) + \Omega$, the addition of Ω being done since Ω packets have successfully left the network leading to the Ω duplicate acks.

Now after retransmitting only the first lost packet, the sender waits for the ack of the retransmitted (lost) packet. If the sender receives any more duplicate acks while waiting for the ack of the retransmitted packet, $W(t)$ is increased by 1 for each duplicate ack received. Thus the sender "inflates" its window by the number of duplicate acks received. After half a window of duplicate acks are received, the sender transmits a new packet for each additional duplicate ack received. In other words, until $\lfloor \sigma/2 \rfloor$ + number of duplicate acks received exceeds σ, the sender cannot transmit any more new packets. Upon receipt of an ack for new data (recovery ack) the sender exits fast recovery and starts transmission with a window of $W(t^+) = \omega(t^+) = \lfloor \sigma/2 \rfloor$.

When a single packet is dropped from a window of data then the ack for the first retransmission will complete the recovery as is obvious from above. But, if there were multiple packet losses from a window of data then the ack for the first retransmitted packet leads to a partial ack. A partial ack takes out the sender from fast recovery and the usable window is deflated back to the size of the congestion window. If the resulting window size is sufficient to permit the sender to transmit additional packets that can elicit further duplicate acks then the sender may enter fast recovery again and repeat the process for the next lost packet. But if the number of duplicate acks are insufficient, then the recovery stalls and a timeout has to be waited for. After a timeout the basic algorithm is followed.

The strategy followed by the Reno sender to recover from losses is to retransmit *at most* one dropped packet per round trip time. For a single packet lost from a data window, Reno significantly improves upon the behavior of Tahoe but with multiple packet losses from a data window, Reno suffers from serious performance problems [6, 12].

TCP New-Reno: [6] This version addresses the problem faced by Reno when multiple packets are lost from a window of data. In Reno receipt of partial acks takes the sender out of fast recovery. In contrast, the NewReno sender does not get out of fast recovery on the receipt of partial acks. Instead, partial acks are treated as an indication that the packet immediately following the acknowledged packet has been lost and should be retransmitted. Thus, when multiple packets are lost from a single window of data, NewReno can recover without a retransmission timeout, retransmitting *one lost packet per round-trip time until all lost packets from that window have been retransmitted* [6]. Of course, if the number of packets lost is such that the sender cannot enter the fast retransmit phase, then a timeout has to be resorted to. The sender remains in fast recovery until all of the data outstanding when fast recovery was initiated has been acknowledged.

Since NewReno addresses the shortcomings of Reno, it is obvious that Reno can never better the performance shown by NewReno. Hence, in the rest of the paper, we do not concern ourselves with Reno. Another reason for this is also that it has been shown for multiple losses, Reno can perform worse than Tahoe [6, 12].

TCP Sack:[6] The Sack (selective ack) version of TCP that we consider is a conservative extension of Reno in that they use the same policies for the window adaptation. Each ack received by the sender contains information about any non-contiguous set of data that has been received and queued at the receiver. From this the sender is able to infer the different packets lost. The Sack sender enters fast recovery on receiving Ω duplicate acks. The actual implementation during fast recovery differs from the Reno implementation but the overall idea is the same and hence we do not describe this here for lack of space. Details about the actual implementation can be found in [6]. On receiving partial acks, the sender does not exit fast recovery just like in NewReno. But a difference with NewReno is that since the sender has more information about the different packets lost, it *is not constrained to retransmit at most one dropped packet per round trip time as NewReno is.* The sender exits fast recovery when a recovery ack is received. Even here, when the sender is unable to enter the fast retransmit phase due to lack of adequate duplicate acks, timeout has to be resorted to.

In the analysis though, 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 the 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 [15]. Our description above of the different TCP versions has been very brief. Readers requiring more details can refer to [6].

3 Loss Models

It is well known that mobile radio channels in an actual physical environment are subject to multipath fading. This multipath fading process can be slowly varying for typical user speeds and carrier frequencies in a mobile environment. Due to this the assumption of iid packet losses as is generally done is not entirely true. Hence, we consider a model incorporating correlated losses as we specify next.

The multipath fading process in a mobile environment follows a Rayleigh distribution [9]. The issue of modelling a correlated Rayleigh fading channel by means of a simple two state Markov chain has been addressed in [11] from where 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 is 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 [11, 17, 5] 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

$$\pi_G = \frac{\beta}{\alpha + \beta} \qquad \pi_B = \frac{\alpha}{\alpha + \beta} \tag{2}$$

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.

4 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 generally 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. Of course if not enough duplicate acks as required by the fast retransmit mechanism arrive back at the sender, then a timeout, which is the non-arrival of the ack of a packet within a certain time interval, is taken as an indication of the 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 this is followed by the congestion avoidance mechanism. Of course if a retransmit timeout has to be resorted to, then the sender always starts in the slow start phase after the timeout interval.

Hence, the operation of a NewReno transmitter can be considered 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. Thus a typical cycle in case of New Reno 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.

Thus, every cycle can be considered to have three stages, the first stage in which 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). Note that with our assumptions only one of either the second or the third stages is present in a NewReno sender.

In order to explain the working of these algorithms we define 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 if in slow start phase begins with one packet being transmitted in the first mini-cycle and if it is in the congestion avoidance phase, then the number of packets transmitted in the first mini-cycle depends on the window size when the packet drop was detected in the previous cycle. In every successive mini-cycle the number of packets transferred is double the number of packets transferred during the present mini-cycle as long as the sender is in the slow start phase. During the congestion avoidance phase the number of packets transferred in each mini-cycle is one more than the number of packets transferred during the previous mini-cycle.[†] 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. End of the first stage can lead to the second stage. Since a NewReno sender recovers only one lost packet per 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

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

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 mini-cycle travel in what we call a train. Thus, there is a packet train in every mini-cycle and the size of the packet train in the kth mini-cycle $k > 1$, depends on the size of the packet train in the previous mini-cycle 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. A great convenience offered by the packet train concept is that it helps to differentiate packets on the basis of the mini-cycle 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.

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 4a. We start observation when the ack for packet 14 reaches the sender. We assume that packets 15-18 is 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 transmission 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 in the second train which is packet 29. Packets 30 and 31 are also assumed to be lost.

Pkt Recd info in ack	Ack for pkt	ω	W	Pkt sent
14	14	5(say)	8	22
14 I dup ack	19	5	8	(pkt 15-18 lost)
14 II dup ack	20	5	8	-
14 III dup ack(II stage)	21	4	7	15
14 IV dup ack	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 dup ack)	32	4	5...	-
28(II dup ack)	33	4	5..	-
New cycle	-	2	1	29

Table 4a: A TCP NewReno cycle

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 packet 32 and packet 33 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 starts 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, hence 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 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 case of NewReno these trains have only one packet such that the particular packet has been detected to be 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 round trip time as in case of Reno or NewReno.

As is clear from the above explanation every cycle of OldTahoe and Tahoe starts with a slow start. In 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 round trip time (rtt) of the link is denoted by T. Further P_τ denotes the steady state probability of a timeout in a cycle. We would like to remark that 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 $\hat{Q}$ while 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

$$\begin{aligned}
&E(\hat{Q}) = E[E(\tilde{Q}/n_1, n_2, \sigma)] + E(\epsilon) \\
&= \sum_{\sigma}\sum_{n_1}\sum_{n_2} P(n_1, n_2, \sigma)E(\tilde{Q}/n_1, n_2, \sigma) + E(\sigma) \\
&= \sum_{\sigma} P(\sigma) \sum_{n_1} P(n_1/\sigma) \sum_{n_2} P(n_2/n_1, \sigma)E(\tilde{Q}/n_1, n_2, \sigma) \\
&+E(\sigma) \\
&= \sum_{\sigma} P(\sigma) \sum_{n_1} P(n_1/\sigma) \sum_{n_2} P(n_2/n_1, \sigma)\,[n_2 + E(Q/n_1, \sigma)] \\
&+E(\sigma) \\
&= E[E(n_2/n_1, \sigma)] + E[E(Q/n_1, \sigma)] + E(\sigma) \qquad (4) \\
&E(Q/n_1, \sigma) = \sum_{k=0}^{n_1} \theta_k \qquad (5)
\end{aligned}$$

The fourth equation above follows from the third equation 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 round trip time taken by a packet of the kth type 1 train. Then we have

$$\zeta_{OT} = \sum_{k=1}^{n_1+1} \phi_k + \tau_t \qquad (6)$$

$$\zeta_T = \sum_{k=1}^{n_1+1} \phi_k + I_\tau \tau_t \qquad (7)$$

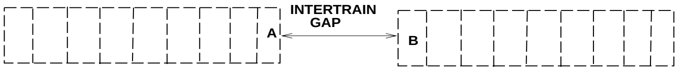

Figure 1: *Gap between different trains*

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

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

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 500ms. 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 as also seen in case of the example given earlier. Further, 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

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

At this point, we can use equations 4 and 10 in equation 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 equations 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. We would like to point out that due to the different mechanics of the various TCP algorithms, the required expressions above would highly depend on the TCP algorithm being considered. We remark here that 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. As far as the first packet of a train is concerned, in the sequel we assume that the first packet of every train finds the wireless link in its steady state. In other words, the first packet of every train finds the wireless link in the good state with probability π_G and in the bad state with a probability π_B. This implies that, as shown in figure 1, effect of the success or failure of packet A on packet B is negligible. 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. Another reason for making this assumption is also that the mean intertrain gap in case of the different TCP algorithms is also large enough. Finally 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 in figure 1 is low enough can also be very easily taken care of. This can be done by calculating the window size W until 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. We do not further pursue this approach for simplicity.

5 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. We do this only for TCP Tahoe for lack of space. Expressions for the other algorithms follow similarly as the expressions for Tahoe and are provided in detail in [1].

5.1 Loss Window Probability Calculation

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 $P(\sigma), \quad \sigma = 1, 2, \cdots, W_m$ 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 Transition Probability— *Consider TCP OldTahoe or TCP 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,*

$$P(W_\rho = \omega + k / W_{\rho-1} = \sigma) = \begin{cases} \alpha(1-\alpha)^{\omega+k-1}\left(\pi_G + \frac{\pi_B\beta}{1-\alpha}\right)^{\gamma_1} & -\omega < k < 0\textit{\& } log_2(\omega+k) \textit{ not int} \\ (1-\alpha)^{\omega+k-1}\left(\pi_G + \frac{\pi_B\beta}{1-\alpha}\right)^{\gamma_1} \cdot & -\omega < k < 0\textit{\& } log_2(\omega+k) \textit{ int} \\ (\pi_G\alpha + \pi_B(1-\beta)) & \\ (1-\alpha)^{\gamma_3}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\gamma_2} \cdot & 0 \le k < W_m - \omega \\ (1-\psi_1^\omega - \psi_2^\omega) & \\ (1-\alpha)^{\gamma_3}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\gamma_2} & k = W_m - \omega \end{cases} \tag{11}$$

where

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

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 $F_s(\eta)$ 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

$$\begin{aligned} S_s(\eta) &= (1-\alpha)^{\eta-1}\left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{\lceil log_2(\eta)\rceil} \\ F_s(\eta) &= \begin{cases} \pi_G\alpha + \pi_B(1-\beta) & log_2(\eta) \text{is an integer} \\ \alpha & log_2(\eta) \text{is not an integer} \end{cases} \end{aligned}$$

$\omega + k$ is the drop window size during the cycle of interest. Since we consider TCP Tahoe during the slow start phase, we have $k < 0$. Now, 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

$$F_s(\omega + k) = \alpha \quad \text{if } \ log_2(\omega + k) \text{ is not an integer} \tag{12}$$

Now consider that $log_2(\omega + k)$ is 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

$$\begin{aligned} F_s(\omega + k) &= \sum_{i=\text{good,bad}} \text{Prob that channel is in ith state . probability of dropping packet} \\ &= \pi_G\alpha + \pi_B(1-\beta) \ \text{ if } \ log_2(\omega + k) \text{ is an integer} \end{aligned} \tag{13}$$

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 $1 - \alpha$. While 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, note that we have $\lceil log_2(\omega + k)\rceil$ places to fill with either π_G or π_B in all possible combinations. Hence, we have

$$\begin{aligned} S_s(\eta) &= \pi_G^{\lceil log_2(\omega+k)\rceil}(1-\alpha)^{\omega+k-1} + \pi_G^{\lceil log_2(\omega+k)\rceil-1}(1-\alpha)^{\omega+k-2}\pi_B\beta + \cdots + \\ & \quad \pi_G^{\lceil log_2(\omega+k)\rceil-j}(1-\alpha)^{\omega+k-1-j}\pi_B^j\beta^j + \cdots + (1-\alpha)^{\omega+k-1-\lceil log_2(\omega+k)\rceil}(\pi_B\beta)^{\lceil log_2(\omega+k)\rceil} \\ &= (1-\alpha)^{\omega+k-1}\sum_{i=0}^{\lceil log_2(\omega+k)\rceil}\left(\begin{array}{c}\lceil log_2(\omega+k)\rceil \\ i\end{array}\right)\pi_G^{\lceil log_2(\omega+k)\rceil-i}\left[\pi_B\beta(1-\alpha)^{-1}\right]^i \\ &= (1-\alpha)^{\omega+k-1}\left[\pi_G+\frac{\pi_B\beta}{1-\alpha}\right]^{\lceil log_2(\omega+k)\rceil} \qquad (14) \end{aligned}$$

Thus the expression for the transition probability for the case $k < 0$ *i.e.* the slow start phase, follows from equations 12, 13 and 14.

Now consider the case $k \geq 0$ *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

$$\begin{aligned} S_c^\sigma(\eta) &= (1-\alpha)^{s(\omega-1,\eta-\omega)}\left[\pi_G+\frac{\pi_B\beta}{1-\alpha}\right]^{\lceil log_2(\omega)\rceil+\eta-\omega} \\ F_c(\eta) &= 1-\pi_G(1-\alpha)^\eta-\pi_B\beta(1-\alpha)^{\eta-1} \quad \eta < W_m \\ &= 1 \quad \eta = W_m \end{aligned}$$

During the congestion avoidance phase, the window advances by one packet for every successful window of packets delivered. Further, for a drop window of $\omega+k$ during the present cycle at least one of the $\omega+k$ packets has to be unsuccessful. Probability that not a single packet of a train with 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

$$F_c(\omega+k) = 1-\pi_G(1-\alpha)^{\omega+k}-\pi_B\beta(1-\alpha)^{\omega+k-1} \quad \omega+k < W_m \qquad (15)$$

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

Further, 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, we have using an approach similar to equation 14

$$S_c^{\sigma}(\omega + k) \;\; = \;\; (1-\alpha)^{s(\omega-1,k)} \left[\pi_G + \frac{\pi_B \beta}{1-\alpha}\right]^{\lceil log_2(\omega)\rceil + k} \tag{16}$$

Thus the expression for the transition probability for the case $k \geq 0$ *i.e.* the congestion avoidance phase, follows from equations 15 and 16.

◇

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

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

5.2 Timeout Probability Calculation

For the timeout probability calculation we have

$$P_\tau \;\; = \;\; \sum_{\sigma'=1}^{W_m} P(\sigma')\tau_{\sigma'} \tag{17}$$

where $\tau_{\sigma'}$ is the probability of timeout in a cycle given that the loss window is σ'. 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: *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*

$$\begin{aligned} \tau_{\sigma'} \;\; &= \;\; \sum_{i=\sigma'-\Omega}^{\sigma'-2} \;\; \sum_{m=1}^{min(\sigma'-1-i,i+1)} \;\; \sum_{l=m-1}^{min(m,i)} \begin{pmatrix} \sigma' - i - 2 \\ \sigma' - 1 - i - m \end{pmatrix} \begin{pmatrix} i \\ i - l \end{pmatrix} (1-\beta)^{i-l}\alpha^l \beta^m (1-\alpha)^{\sigma'-1-i-m} \\ &+ \;\; (1-\beta)^{\sigma'-1} \end{aligned} \tag{18}$$

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'.

Hence, with this 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

$$(1-\beta)^{i-l}\alpha^{l} \tag{19}$$

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

$$\beta^{m}(1-\alpha)^{\sigma-i-m} \tag{20}$$

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. Note that a β cannot occur after a β unless an α has occurred between them and vice versa. Hence, we have $l = m$ or $l = m-1$ *i.e.*

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

Further,

$$\begin{aligned} \sigma - i - m \geq 0 \quad &\rightarrow \quad \sigma - i \geq m && (21)\\ i - l \geq 0 \quad &\rightarrow \quad i \geq l && (22)\\ &\rightarrow \quad i+1 \geq m \geq l && (23) \end{aligned}$$

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 $(1-\alpha)^f \; f = 0, 1, 2, \cdots$ 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

$$\begin{pmatrix} \sigma - i - 1 \\ \sigma - i - m \end{pmatrix} \tag{24}$$

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

$$\begin{pmatrix} i \\ i - l \end{pmatrix} \tag{25}$$

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 equations 24 and 25. Hence, probability of a timeout given i losses $\tau^i_{\sigma'}$, such that $\sigma > i \geq \sigma' - \Omega$ is given by

$$\tau^i_{\sigma'} = \sum_{m=1}^{min(\sigma-i,i+1)} \sum_{l=m-1}^{min(m,i)} \begin{pmatrix} \sigma - i - 1 \\ \sigma - i - m \end{pmatrix} \begin{pmatrix} i \\ i - l \end{pmatrix} (1-\beta)^{i-l} \alpha^l \beta^m (1-\alpha)^{\sigma-i-m}$$

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

$$\begin{aligned} \tau_{\sigma'} = & \sum_{i=\sigma'-\Omega}^{\sigma-1} \sum_{m=1}^{min(\sigma-i,i+1)} \sum_{l=m-1}^{min(m,i)} \begin{pmatrix} \sigma - i - 1 \\ \sigma - i - m \end{pmatrix} \begin{pmatrix} i \\ i - l \end{pmatrix} (1-\beta)^{i-l} \alpha^l \beta^m (1-\alpha)^{\sigma-i-m} \\ & +(1-\beta)^{\sigma} \end{aligned}$$

◇

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

5.3 Train Probability Calculation

Proposition 3 *Consider TCP Tahoe or TCP OldTahoe. Then the probabilities of the different* *trains is given as*

$$\begin{aligned} P(n_1 = k/\sigma) &= (1-\alpha)^{\theta_1+\theta_2+\cdots+\theta_{k-1}} \left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{k-1} [1-\psi_{1k}-\psi_{2k}] \quad 0 < k < t_m \\ P(n_1 = t_m/\sigma) &= (1-\alpha)^{\theta_1+\theta_2+\cdots+\theta_{t_m-1}} \left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{t_m-1} \qquad (26) \end{aligned}$$

where

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

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

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

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

$$P(n_1 = t_m/\sigma) = (1-\alpha)^{\theta_1+\theta_2+\cdots+\theta_{t_m-1}} \left[\pi_G + \frac{\pi_B\beta}{1-\alpha}\right]^{t_m-1} \qquad (29)$$

◇

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

$$\begin{aligned} E(n_1) &= E[E(n_1/\sigma)] \qquad (30) \\ &= \sum_\sigma P(\sigma) \sum_j jP(n_1 = j/\sigma) \qquad (31) \end{aligned}$$

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/OldTahoe.

5.4 Packet Count Calculation

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

$$E(Q/n_1 = k, \sigma) \quad = \quad \theta_1 + \theta_2 + \cdots + \theta_k \quad 0 < k \leq t_m \; 0 < \sigma \leq W_m \tag{32}$$

where

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

Proof: Note that every cycle of Tahoe starts with 1 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.

◇

6 Analytical Study

Now that the loss window probability, train probability, packet count and the timeout probability are specified, we use these expressions in equations 4 and 10 to calculate the throughput using equation 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. While in the earlier section, expressions only in case of Tahoe are specified, we do 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 (bits), link bandwidth μ (Mbps), timeout value (s), rtt T (s) and the fast retransmit threshold Ω (packets). The default values of the parameters are $S = 125$bytes, timeout granularity of 0.5 seconds for coarse timeout and 0.05 seconds for fine timeout [10] while $\Omega = 3$ unless otherwise specified.

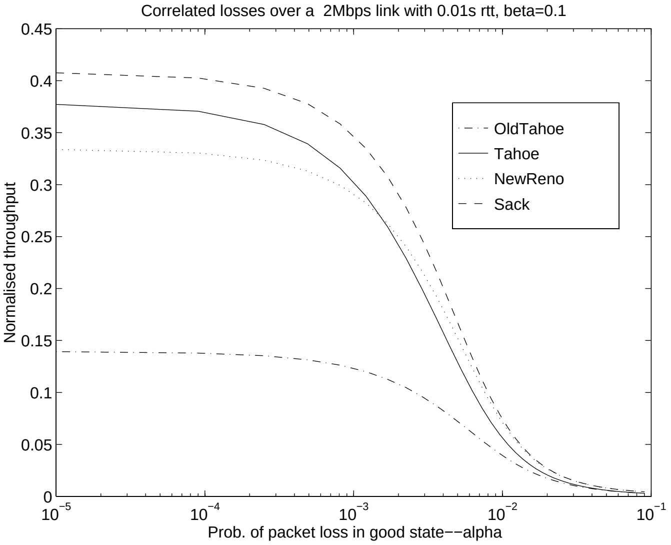

Figure 2: *Behavior of different TCP algorithms with correlated packet loss model under bursty loss conditions over a link with a bandwidth delay product of 20 packets*

Before proceeding, 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 atmost 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

$$\begin{aligned} \frac{1}{\alpha} &\geq t_m.T.\text{number of packets transmitted per unit time} \\ &\geq t_m T \mu \end{aligned} \tag{33}$$

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

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

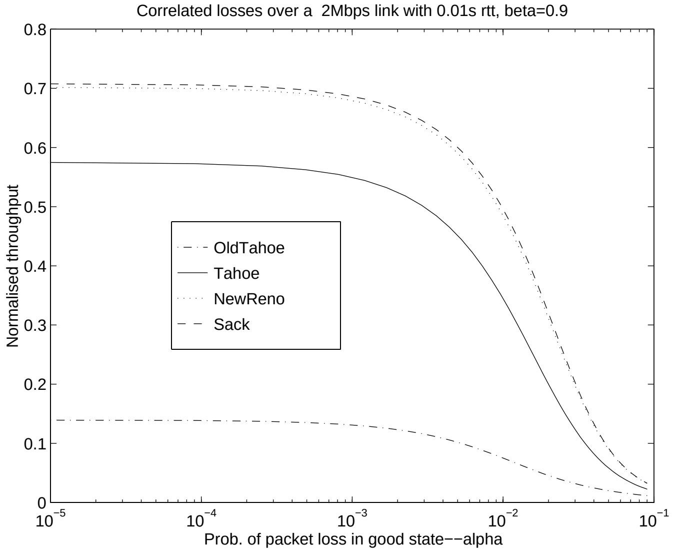

Figure 3: *Illustration of the effects of non-bursty loss conditions showing much performance improvement compared to the bursty loss conditions*

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 it's maximum possible values. It has to be remarked here that this condition would not be necessary if the connection is using fine timeout values since in that case the effect of a timeout is not severe at all. As we see later, for all the scenarios considered both the above conditions are necessary to ensure good link utilization.

Now, consider figure 2 which illustrates the performance of a wireless link with a bandwidth of 2Mbps and an rtt of 0.01s. 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. Similar scenarios with $\beta = 0.9$ is shown in figure 3. As can be seen from these figures as the value of β increases , the performance of the different TCP algorithms (except OldTahoe) also improves. Also 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. But since

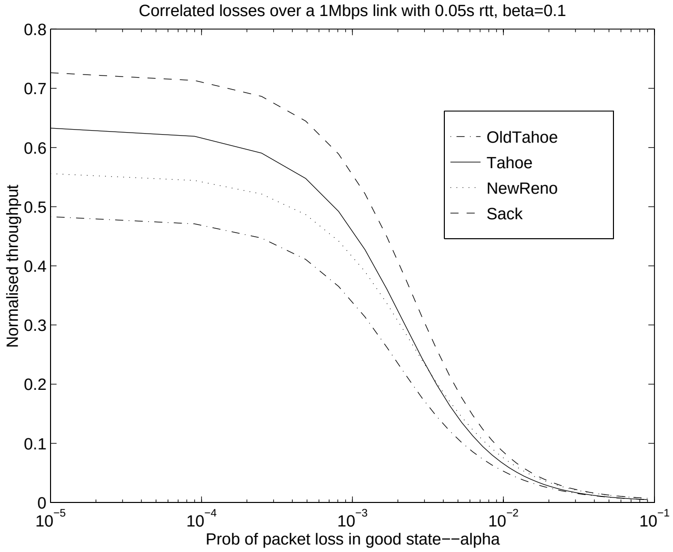

Figure 4: *Performance of TCP versions under very bursty conditions showing significant performance improvement at low α for a higher bandwidth-delay product of 50 packets with $\beta = 0.1$*

OldTahoe always depends on a timeout to infer packet loss, there is no performance improvement in contrast to the other TCP versions. At moderate values of β, there are more chances of multiple packet drops in a window while at high values of β around 1, generally only single packet drops occur.

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 6a: Evolution of Tahoe when packets 15-19 are lost taking 3 rtts to recover the lost packets

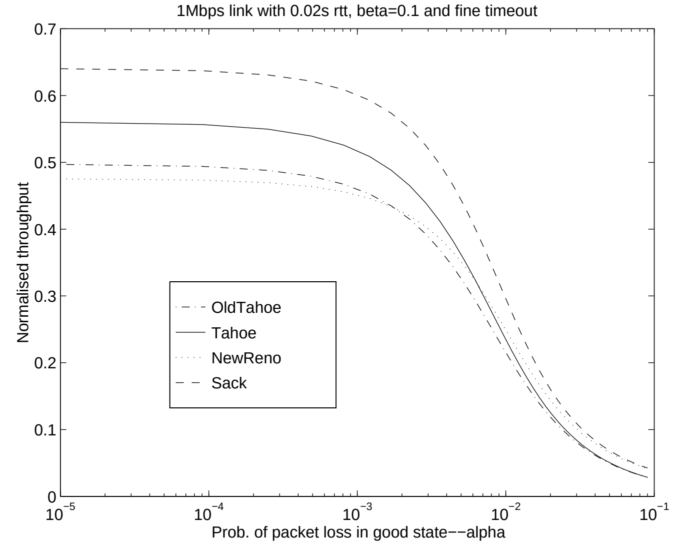

Figure 5: *Illustrating that at low values of β the performance depends only on the bandwidth-delay product when using a fine timeout interval*

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

Table 6b: Fast Recovery for New Reno when packets 15-19 (inclusive) are lost taking 5 rtts to recover the lost packets

Another important observation from these figures is also 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 *e.g.* in a window of 10 as in these figures, assuming 5 bursty packet losses (read 5 consecutive packets lost)

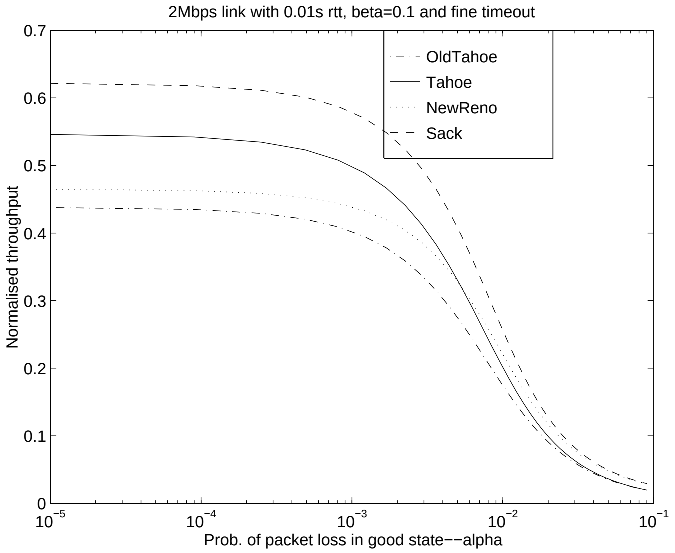

Figure 6: *Illustrating that at low values of β the performance depends only on the bandwidth-delay product when using a fine timeout interval*

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 5 rtts to recover from these 5 packet losses by sending one lost and utmost 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 Table 6a and 6b. As shown, when 5 consecutive packets 15-19 are lost, Tahoe recovers these packets within 3 rtts while NewReno takes 5 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 but 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 5 rtts while NewReno is able to send only 5 packets not present at the receiver during these 5 rtts. 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 the performance of Tahoe by a larger degree. We see this later in figure 4 and 10. 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, performance of NewReno can

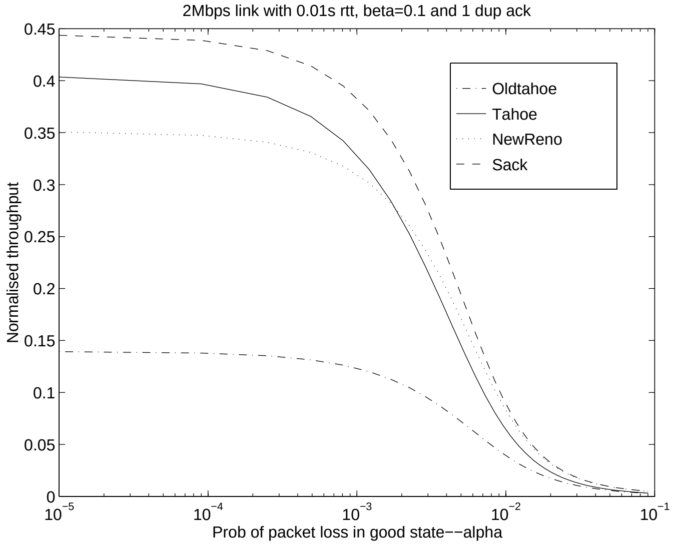

Figure 7: *Illustrating that reducing the fast retransmit threshold has a marginal effect in the regime of low β*

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 figure 2.

It is to be remarked here that if packet losses were more staggered like in the iid model, NewReno while being constrained to send at most one lost packet per rtt, can also send more than two packets 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 iid loss regime (causing staggered packet losses). We remark here that it can be seen from these figures that the link utilization achieves the maximum possible for the concerned TCP algorithm whenever the necessary conditions derived earlier are satisfied. Note that with a link bandwidth delay of 20 packets, the two necessary conditions for good performance translate into $\alpha \leq 0.0025$ and $\beta > 0.1$.

Figure 4 considers a wireless link with a bandwidth of 1Mbps and 0.05s rtt. Thus the two conditions for a bandwidth delay product of 50 packets translate into $\alpha \leq .0004$ and $\beta > 0.04$. The same observations made earlier hold in this case, the most notable being that the performance of NewReno is worst than that of Tahoe under very bursty loss conditions (read β around 0.1 and lower) at higher bandwidth delay products as expected. Further from these figures it can also be

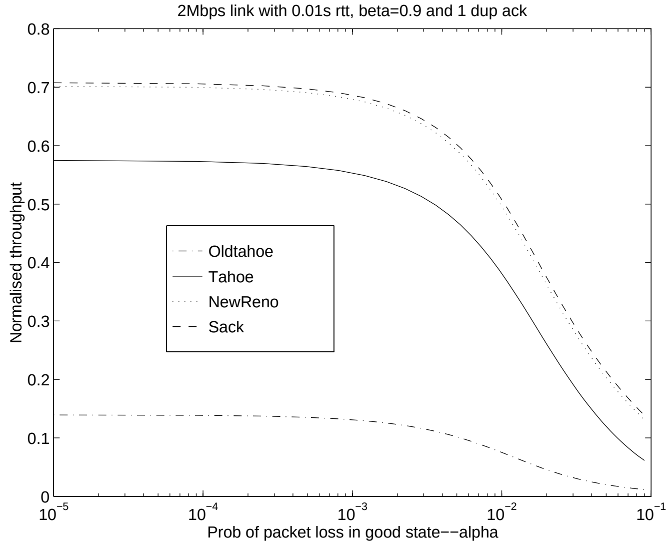

Figure 8: *Illustrating that reducing the fast retransmit threshold has a significant effect in the regime of high β only for high α values*

seen 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. Note that this effect is predominant only with low values of β as can be seen from figures 2 and 4. 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 also 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.

To show that the performance difference is indeed due to the coarse timeout granularity for low values of β, in figures 5 and 6 we consider link bandwidths of 1 Mbps and 2 Mbps respectively while ensuring that the bandwidth delay products remain the same at 20 packets by properly choosing the rtt values. 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. Comparing figure 6 with figure 2, the performance improvement for the different

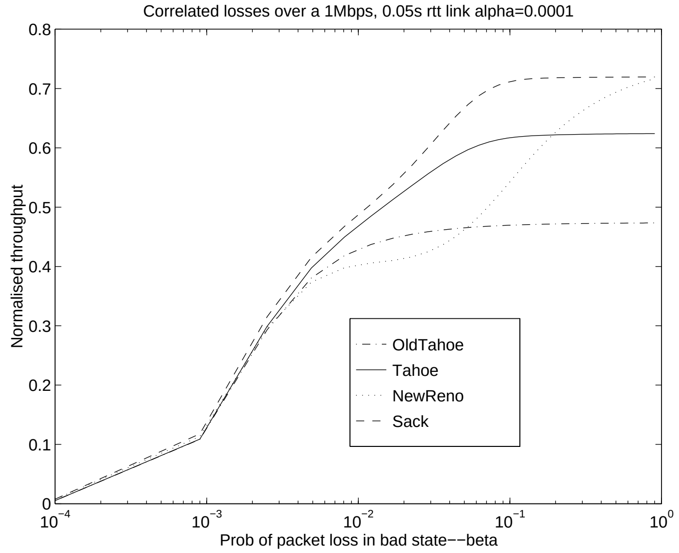

Figure 9: *Illustrating the effects of varying β for a link with a bandwidth-delay product of 50 packets*

TCP versions using a fine timeout is obvious. 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. This is also seen in figures 5 and 10. Validity of our earlier remark that while using fine timeouts the second condition is not important is also proven by looking at these figures.

Effects of changing the fast retransmit threshold from 3 to 1 are shown in figures 7 and 8 with values of β being 0.1 and 0.9 respectively. Comparing these with figures 2 and 3 it can be seen that at low values of β, the reduction in the fast retransmit threshold improves the performance marginally only at low α values. At high α and low β values the threshold reduction does not lead to a significant performance change since in this regime small windows and large bursts are the norm. From figures 3 and 8 it can be seen that the lower threshold makes a significant difference only at high values of α. Thus, the threshold reduction makes a significant difference only in the regime of small window values and small bursts (big α and big β). In the other regimes the effect is not that significant. Hence going for a reduction in threshold may not be that good an idea unless this is accompanied by some way to sense the state of the channel. Note also that a threshold of one is not robust to packet reordering by the network. So the option of reducing the fast retransmit threshold if chosen would have to be done with care.

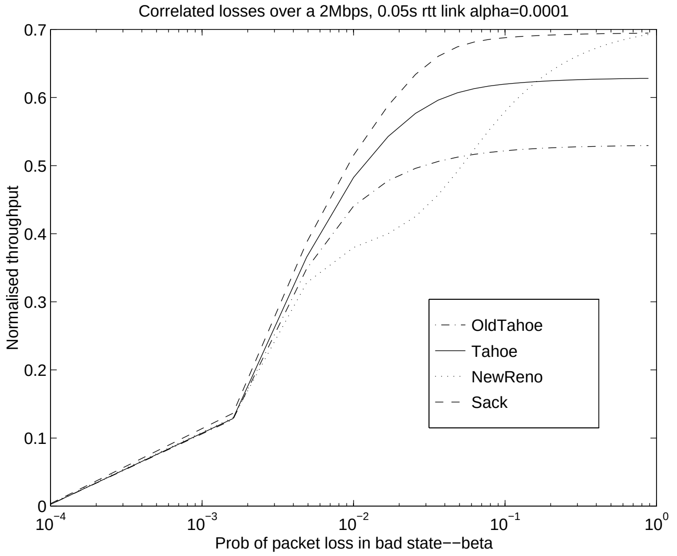

Figure 10: *Illustrating the effects of varying β for a link with a bandwidth-delay product of 100 packets*

We next investigate the effects of β while keeping α constant. Since, these results can only enhance some of the conclusions drawn earlier, we just look at two scenarios. These are shown in figures 9 and 10. In figure 9, we consider a scenario with a bandwidth-delay product of 50 packets and a wireless link bandwidth of 1Mbps. 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 it's behavior improves continually as β increases until at high values of β the performance of NewReno and Sack becomes very similar. From figure 10 which considers a link with a bandwidth delay product of 100 packets and a link bandwidth of 2Mbps, it can be seen that the minimum robustness threshold decreases as the bandwidth-delay product increases. The performance of NewReno though 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.

7 Conclusion

In this paper we have looked at the behavior of the different TCP algorithms over a wireless channel with correlated packet losses. 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 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] Farooq M. Anjum. Analysis and design of a reliable wireless transport protocol and fair network mechanisms for the internet. *Ph.D Thesis, Univ Of Maryland at 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 Transactions on Networking*, June 1997.

[3] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 14, Dec 1995.

[4] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environment. *IEEE JSAC*, 1994.

[5] A. Chockalingam, M. Zorzi, and R.R. Rao. Performance of tcp on wireless fading links with memory. *IEEE ICC*, June 1998.

[6] K. Fall and S. Floyd. Comparisions of tahoe, reno and sack tcp. *ftp://ftp.ee.lbl.gov*, Mar 1996.

[7] Z.J. Haas and P. Agrawal. Mobile-tcp: An asymmetric transport protocol design for mobile systems. In *ICC*, Montreal, Canada, June 1997.

[8] Van Jacobson. Congestion avoidance and control. *Proc. ACM Sigcomm*, pages 314–329, Aug 1988.

[9] Jakes W. C. Jr. *Microwave mobile communications,*. John Wiley & Sons, New York, 1974.

[10] A. Kumar. Comparative performance analysis of versions of tcp in local network with a lossy link. *Tech Rep WINLAB-TR 129,*, Oct 1996,also TON Aug1998 pp 485-498.

[11] 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. *Tech Rep WINLAB-TR 133*, Nov 1996.

[12] T.V. Lakshman and U. Madhow. The performance of tcp/ip for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. on Networking*, 5(3):336–350, June,1997 1997.

[13] P.P. Mishra, D. Sanghi, and S. K. Tripathi. Tcp flow control in lossy networks: Analysis and enhancements. *IFIP Trans. C-13 Computer Networks, Architecture and Applications*, pages 181–193, 1993. Elsevier North Holland.

[14] 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*, 3(1), Mar 1995.

[15] Lakshman T.V., U. Madhow, and B. Suter. Window based error recovery and flow control with a slow acknowledgement channel: a study of tcp/ip performance. In *Proc. INFOCOM*, April 1997.

[16] K. Wang and S. K. Tripathi. Mobile-end transport protocol: An alternative to tcp/ip over wireless links. *Infocom*, pages 1046–1053, April 1998.

[17] M. Zorzi and R.R. Rao. Effect of correlated errors on tcp. *Proc. CISS*, pages 666–671, March 1997.