Performance of TCP over Lossy Upstream and Downstream Links with Link- level Retransmissions

Farooq Anjum and Ravi Jain
Applied Research
Telcordia Technologies, Inc. (formerly Bellcore)
Morristown, NJ.

Abstract— **We study the efficacy of using link-layer retransmissions to improve TCP performance over lossy wireless links. The scenario we consider is where TCP packets traverse a wired network to a base station, and thence over a single wireless hop to a stationary receiver. Unlike many previous studies, which rely on simulation, we develop an analytical model for calculating TCP throughput; unlike all previous analytical studies, we do not ignore the possibility of acknowledgement packets (ACKs) being lost on the reverse link from the wireless receiver to the base station. The analytical model captures the performance of four common TCP algorithms: OldTahoe, Tahoe, NewReno and Sack.**

We find that, for the scenarios studied, a moderate number of link-level retransmissions can significantly improve the throughput of TCP and its resilience to packet losses on the forward and reverse links; increasing the number of retransmissions further has relatively little benefit. We also show that ACK losses cannot be ignored unless the number of retransmissions permitted is high or errors on the reverse link are infrequent. Of the TCP algorithms studied, we find that Sack has the best performance, closely followed by NewReno; the performance of OldTahoe is very poor. We end with suggestions for further work.

I. Introduction

Wireless access to information systems and the Internet is becoming increasingly widespread, with numerous commercial products and services becoming available in recent years. Most applications, whether designed for use over wired or wireless networks, rely on the TCP protocol for reliable transport. However, TCP is designed for operation in wired networks where random packet losses due to transmission errors are negligible. It is well-known that TCP performs poorly over wireless links which suffer from packet losses due to the error- prone and idiosyncratic nature of the wireless medium [2], [3], [4]. TCP does not adapt appropriately to losses on wireless links, as it interprets the cause as being network congestion.

In this paper we study how to adapt TCP to wireless links, particularly for the important scenario where the last hop of a communication path is wireless. Like previous studies, we focus on the case of a single TCP flow because modeling the interaction between multiple TCP flows is notoriously difficult, and we wish to concentrate, for the moment, on developing insight into the interaction between random packet losses and the TCP dynamic window adjusting mechanism. A variety of strategies have previously been proposed for adapting TCP to wireless links in the scenario where the last hop of a communication path is wireless. These strategies include, for example, Indirect-TCP, which modifies the end-to-end semantics of TCP, and snoop, which relies on network- layer retransmissions over the wireless link to localize recovery; see [2] for a discussion. In this paper we consider the effect of relying simply upon link-level retransmissions over the wireless hop to compensate for wireless link errors. This strategy does not require any modifications to TCP and does not affect the end-to-end semantics of TCP.

Unlike many previous studies of TCP over wireless which are based on simulation, we develop an analytical model that models TCP as well as the effect of link- layer retransmissions. Recently there has been increased interest in developing analytical models for TCP [8], [9], [10]. The effect of link layer protocols on the TCP behavior has also been studied earlier. Many of these studies [2] have been mainly simulation based studies. On the other hand, [7] provides an analytical framework for quantifying the performance of TCP operating over a wireless link using a suitable link level error recovery mechanism. They consider TCP Tahoe and Reno in such a scenario. The main concern of this paper is the additional buffer requirement at the wireline to wireless interface so as to handle the wireless channel losses assuming infinite number of retransmissions at the link layer. [5] concludes that having independent retransmission protocols at the data-link and transport layer levels leads to degraded performance. However, this conclusion was reached based on analysis that did not take into account the generous timeout granularity which is assumed in practical TCP implementations.

Like many of these previous analytical studies we assume a simple loss model over the wireless link where each packet is assumed to be lost with a certain constant probability independent of other packets. However, we observe that these previous studies assume that while data packets can be lost on the wireless link, acknowledgement packets (ACKs) are not lost. This assumption is typically made in order to simplify the analysis or the presentation of results. We do not make this simplifying assumption, and show later that in several scenarios this assumption is not valid. Further, we also make less simplistic assumptions and also look at four different TCP algorithms.

Our contributions in this paper can be summarized as follows. We develop an analytical model of the performance of TCP over wireless links, for the scenario where the last hop of a communication path is wireless and link-level retransmissions are used to recover from wireless link losses. The performance metric we study is TCP packet throughput. This analytical model is based on the *packet trains* model of TCP operation [1]; in this paper we show how the model can be used to compactly capture and present performance results for four different TCP algorithms: OldTahoe, Tahoe, NewReno and Sack. In addition,

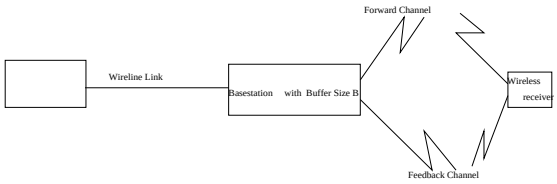

Fig. 1. *Scenario*

the analytical model explicitly considers packet losses for both data packets (i.e., on the forward link) as well as ACKs (i.e., on the reverse link), and we show that assuming an error-free reverse link is not alway valid. We use the model to characterize the conditions under which such an assumption can be made for simplicity in analysis without significantly affecting the results. We apply the model to quantitatively estimate the impact of modifying the link- level retransmission parameters on the performance of TCP.

The paper is organized as follows. The system model that we consider is described in Section 2. In section 3, we analyze the performance of the link layer scheme considered on the different TCP algorithms. We use the concept of packet trains in order to evaluate the performance of the different TCP algorithms. Section 4 is devoted to applying the analytical model to the evaluation of the behavior of different TCP versions under different scenarios for the link layer. Finally we conclude in section 5.

II. Scenario

The system we study in this paper consists of a fixed wired backbone network supporting a wireless access network. We consider a scenario where a fixed host computer, called the *TCP source*, is transmitting packets for a single TCP connection over a communication path consisting of many contiguous wireline links and ending in a single wireless link. The wireless link emanates from a base station to a stationary receiver, called the *wireless receiver*; see Fig. 1. The base station implements a reliable link layer (RLL) protocol stack.

The three commonly-used ARQ protocols at the link layer are Stop and Wait, Go Back N and Selective Repeat [11]. In this paper we consider Stop and Wait protocol operating at the base station. We assume that if the packet cannot be transmitted successfully in N tries, the sender discards the packet. We will show that, provided the link-layer retransmission parameters are tuned correctly, the simple Stop and Wait protocol can result in significant TCP perfomance improvements.

We assume that new packets are always available to be sent out by the TCP source. The base station is assumed to have two buffers: one for holding TCP packets as they arrive from the TCP source, and one for holding link-layer packets that are in the process of being transmitted or retransmitted. We assume that the size of the link layer buffer is greater than the network layer buffer by at least the product of the bandwidth and the roundtrip time over the wireless link. This ensures that packets are not lost due to overflow at the link layer buffer, thus allowing us to focus on the link-layer retransmission parameter N. We assume that the TCP timeout interval at the TCP source is large enough that if a timeout occurs, the link-layer buffer at the base station empties before the next packet arrives from the TCP source. The value of the TCP timeout interval is also assumed to be larger than the total round-trip time taken at the base station link layer to transmit or retransmit a packet, even with the maximum number of retransmissions allowed. This imposes a constraint on the maximum value that N can have. This constraint is justified considering both the fact that many current TCP implementations use a coarse timer as also the fact that the retransmissions are done only over a single hop while the TCP timeout interval takes into account all the links in the end-to-end communication path. Also, in any case N is limited by practical considerations. In fact, as we see later, increasing N beyond a certain value does not improve the system performance significantly.

III. Performance Analysis

In this section we characterize the throughput of the different TCP algorithms operating under the scenario given in the earlier section. The approach that we take is to convert the bit error rate (BER) at the link level on both the forward and reverse channels to a packet error rate as seen at the level of the transport layer. We also consider the effects of the retransmissions at the link layer on the actual round trip time as seen by the TCP layer. Using these, we then use the approach of packet trains [1] in order to characterize the throughput of the different TCP versions.

A. Modeling link level errors

Let ϵ_1 denote the BER on the forward channel and ϵ_2 denote the BER on the reverse channel. Let S_1 and S_2 denote the link layer packet size and the link layer ACK size. Let the probability of a link layer packet being lost on the forward channel be denoted by p while the probability of the link layer packet being lost on the reverse channel be denoted by q. We assume that the probability of a packet being lost is independent of the loss or success of the other link layer packets. Then we have

$$p = 1-(1-\epsilon_1)^{S_1} \qquad q = 1-(1-\epsilon_2)^{S_2}$$

We point out a subtle feature in the operation of TCP with link-layer retransmissions when errors on the reverse channel are considered. It is possible that a packet reaches the wireless receiver successfully but the ACK is lost, and in fact N consecutive ACKs are lost, so that the base station link level source incorrectly infers that the packet has not been delivered, and discards the packet. However, suppose the next packet sent by the TCP source is delivered correctly and the corresponding TCP ACK is also correctly received within N tries. Since TCP ACKs are cumulative, the TCP source is able to infer from the ACK that the previous packet was successfully received by the wireless receiver, and need not be retransmitted even though the link layer at the base station has deemed it to be lost.

Given this feature it is difficult to obtain a simple expression for the exact probability of a TCP packet being successful. For simplicity we obtain instead expressions for the upper and lower bounds on the loss probability of a packet at the TCP level. The upper bound can be obtained by assuming the packet is lost even when it reaches the receiver successfully but no link layer ACKs make it back to the link layer TCP source. Hence the upper limit

on the probability of a TCP packet being lost is given by

$$\bar{p} = \sum_{i=0}^{N} \binom{N}{i} (1-p)^i q^i p^{N-i} \quad (1)$$

The lower limit on the probability of a TCP packet being lost can be obtained by assuming that the TCP packet is successful whenever it reaches the receiver. This amounts to assuming that the acknowledgments are not lost thereby considering an error-free reverse channel. Note that if the TCP packet does not reach the receiver then irrespective of what happens to the following packets the TCP packet has to be retransmitted. Thus the lower limit on the probability of a TCP packet being lost is given by p^N. We can similarly also calculate the average time for transmitting a TCP packet successfully over the wireless link.

The behavior of the different TCP algorithms is analyzed based on the concept of packet trains, which we introduce next.

B. The packet trains model

In this section we define the packet trains model which is used to capture the dynamic behavior of TCP in general and the several versions of TCP in particular. We assume the reader has some familiarity with TCP as well as the four TCP algorithms we consider, namely OldTahoe, Tahoe, NewReno and Sack; see [6] for an introduction.

In the packet trains model the operation of a TCP source is described in terms of time intervals called *cycles*, *stages*, and *minicycles*. Each cycle divided into stages, and the first and second stages of a cycle are divided into minicycles. For the moment, consider a minicycle to be the time during which a complete window of packets is transmitted by the TCP source, and a *packet train* to be the sequence of such packets; a more precise definition will be given below. Since the window size of TCP varies as the protocol progresses, all minicycles (and packet trains) are not of the same length.

For our purposes, the behavior of a TCP source is characterized by three attributes: (1) how successive windows of packets are sent as long as there are no packet losses; (2) how a packet loss is detected by the TCP source; and (3) how the TCP source attempts to recover from packet losses. The four TCP algorithms differ mainly in terms of these three attributes. We now describe the various choices available for each attribute. We will then describe the cycles for each TCP algorithm and how they differ.

Successive window sizes without packet loss. Let l_i denote the length of the ith packet train in the first stage of a particular TCP cycle, for $i \geq 0$. For all the TCP algorithms, the first stage of a cycle consists of either the *slow start* algorithm or the *congestion avoidance* algorithm of TCP. By definition, in slow start $l_i = 2l_{i-1}$, for $i > 0$, and $l_0 = 1$. On the other hand, in congestion avoidance $l_i = l_{i-1} + 1$, for $i > 0$, and $l_0 = W'$, where W' will be defined shortly[1]. Thus for either slow start or congestion avoidance, the window size, packet train size and minicycle length increases monotonically in the first stage of a cycle. The first stage ends when a packet loss is detected by the TCP source.

[1]Note that for simplicity we ignore some common variations like delayed ACKs but these can be incorporated in the analysis if desired.

Packet loss detection. There are two means used by TCP for detecting a packet loss: either by a *timeout* or by receiving a specified number of duplicate acknowledgements (*DACKs*), or both. A timeout occurs when an ACK for a particular packet is not received by the TCP source within a pre-specified amount of time. The DACK mechanism operates as follows. Each ACK received by the TCP source contains information about the sequence number of the last packet which was received correctly and in sequence by the receiver (we call this the Last ACK). If the TCP source receives $D + 1$ consecutive ACKs where Last ACK is the same (say, $j - 1$) it infers that the packet with sequence number j has been lost. Thus the TCP source infers a packet loss when it receives D DACKs. Typically, $D = 3$. The value of D for any given implementation is called *retransmit threshold*.

Packet recovery. Once a packet loss has been detected, a TCP source recovers either by reverting to slow start with $l_0 = 1$, or by following a process called *fast recovery* described as follows. Let W' be the size of the source window when the packet loss was detected by the TCP source on the basis of the receipt of D DACKs. In fast recovery, the source reduces its window size to $(W'/2) + D$. Then the first packet that was lost is retransmitted, and the TCP source waits until an ACK is received for this packet before retransmitting the next packet deemed to be lost. If the TCP source receives any more DACKs while waiting for the ACK of the retransmitted packet, the window size is increased by 1 for each DACK received. After $(W'/2)$ DACKs are received, the TCP source can transmit a new packet for each additional DACK received; this process is continued until all lost packets have been recovered.

Based on the behaviour of the different TCP algorithms, we now define a *packet train* at the TCP source as follows. A packet train starts once the TCP source receives the ACK for the first successful packet of the previous train. A packet train ends either when (1) the TCP source has transmitted the number of packets justified by the ACKs received for the previous train, or (2)the TCP source receives D DACKs or (3)a successful timeout interval is started. A *minicycle* is the length of the time corresponding to a given packet train.

The packet trains model makes two simplifying assumptions, which tend to compensate each other. The first is that during fast recovery for NewReno each packet train only contains retransmitted packets, and has length $m_i = 1$, for $i \geq 0$. The model ignores the fact that new packets may also be transmitted during this stage. The second assumption is that there is no bound on the number of packets that may be sent once fast recovery is completed. In fact most practical implementations have such a bound. The compensating nature of these assumptions makes the packet trains model very accurate, as we will show by means of simulations in Section IV.

In the analysis though, we ignore the fact that on successive timeouts the actual value of the timeout is increased exponentially by Karn's algorithm. Further for the purpose of simplifying the analysis, we also assume that packets retransmitted during fast recovery are not dropped. This latter assumption affects only the NewReno and Sack TCP sources. As a result our analysis gives somewhat 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 of TCP is already very low, we choose to reduce complexity by making these assumptions. We also show later by comparing our model with simulation results that in spite of these approximations our model matches the simulation results very well.

The strength of the packet trains model lies in the fact that the throughput of TCP can be calculated as the ratio of the expected number of packets in a cycle to the mean cycle duration. Further, these two quantities can in turn be easily calculated once two parameters are calculated: (1) the number of trains in a cycle; and (2) the window size at the start of the cycle, which is determined by the window size when the packet drop in the previous cycle was detected. Details of these calculations are not shown here for lack of space and can be obtained from a detailed version of this paper.

IV. Performance evaluation

In the previous section we have characterized the behavior of different TCP algorithms. 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 the TCP versions work over a wireless link. We consider the effects of errors on both the reverse channel and the forward channel. We consider two scenarios. A *low bandwidth scenario* consists of a 0.8Mbps wireless link which is the bottleneck while a *high bandwidth scenario* consists of a 4Mbps wireless link. The total one way delay over wireline links from the TCP source to the basestation is 5ms while the one way delay over the wireless link is 2.5ms. Further we assume a TCP timer granularity of 500 ms. The size of the buffer at the gateway is assumed to be equivalent to the bandwidth delay product of the link in the absence of any link layer retransmissions. The size of each TCP packet is 500 bytes. The other parameters namely D, ϵ_1, ϵ_2, N, S_1 and S_2 vary over the different cases and are specified where necessary.

A. Effects of errors on the forward channel

We first consider the effects of errors on the forward channel on the different TCP algorithms. The BER over the reverse channel is assumed to be fixed at $\epsilon_2 = 10^{-4}$. For each of the different TCP algorithms we calculate the lower limit and upper limit on the performance. The value of retransmit threshold has been set to $D = 3$ in these evaluations. The resulting figures are shown in figures 2 and 3 for the high bandwidth scenario. In these figures we plot the BER on the forward channel on the x-axis. The throughput normalized to the bottleneck link bandwidth is shown on the y- axis. Two curves are shown in each figure corresponding to each of the TCP algorithms. The upper curve shows the upper limit of the normalized throughput. This is calculated by assuming the lower limit on the packet loss probability. The lower curve shows the lower limit of the normalized throughput. This is calculated by assuming the maximum value of the packet loss probability. (In both figures, the curves for NewReno and Sack are almost coincident.)

In figure 2 we assume that the link layer retransmits every packet at most $N = 5$ times. The difference between the upper and lower performance limits in this case is very little. Note that the upper performance limit is the same as assuming that the reverse channel is error free. Thus, in this regime the assumption of an error free reverse channel can be made without much difference in the results provided the BER on the reverse channel is not overly large.

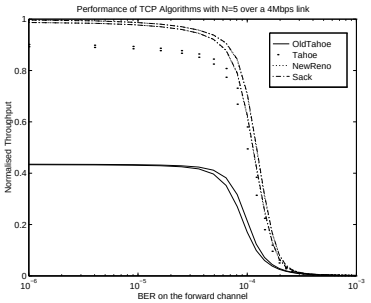

Fig. 2. *Effects of errors on the forward channel and link-level retransmissions*

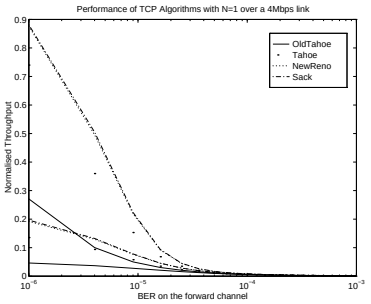

Fig. 3. *Effects of errors on the forward channel and no link-level retransmissions*

Figure 3 considers the case whereby the link layer scheme does not retransmit any packet. The BER on the reverse channel is assumed to be 10^{-4}. In this case the difference between the upper and lower limits is quite significant. Hence ACK losses cannot be ignored in the analysis when the link layer scheme does not perform packet retransmissions.

B. Effects of errors on the reverse channel

We next consider the effects of varying BER on the reverse channel. In figure 4 we show results for the different TCP algorithms namely OldTahoe, Tahoe, NewReno and Sack. On the x- axis we plot the BER on the reverse channel. Throughput normalized to the bandwidth of the bottleneck link is plotted on the y-axis. The BER on the forward channel is assumed to be $\epsilon_1 = 10^{-6}$. For each of the TCP algorithms we consider four different cases, where N equals 1, 2, 5 and 10. (The curve for NewReno is almost coincident with that for Sack.)

We see that the performance of the different TCP versions with $N = 1$ is very sensitive to the BER on the reverse channel and becomes equally poor at high BER. As N increases, the throughput for all TCP versions improves. In addition, all TCP versions become more robust to the BER on the reverse channel: the normalized throughput remains constant in spite of the increase in BER on the reverse channel. In all cases we see that the performance of Sack is the best while the performance of

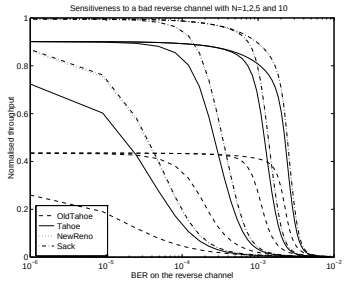

Fig. 4. *Effects of errors on the reverse channel and varying number of link-level retransmissions*

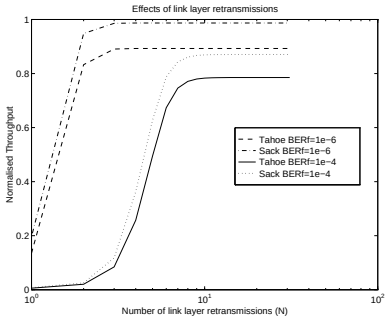

Fig. 5. *Effects of errors on the forward channel and varying the number of retransmissions*

OldTahoe is the worst.

Note that all the curves shown in this figure correspond to the lower limit on the bandwidth. The upper limit is obtained when the value of the BER on the reverse channel is zero. Thus for the purposes of this discussion the throughput for $BER = 10^{-6}$ can be regarded as a rough indication of the upper limit on throughput performance. As can be seen from the figure a significant difference between the upper limits and lower limits arises for all the different TCP versions at all values of BER on the reverse channel with a value of $N = 1$. On the other hand as the number of packet retransmissions at the link level is increased i.e. N is increased, the performance difference between the upper and lower limits for a given value of the BER on the reverse channel decreases. Of course, when the BER on the reverse channel is very high, i.e greater than 10^{-3}, there is a significant difference between the upper and lower limits which implies that in this regime the effects of ACK losses cannot be neglected in the analysis. We observe similar results for the low bandwidth scenario also.

C. Effects of increasing the link layer retransmissions

We next consider the effect of increasing the link layer retransmissions. We consider only TCP Sack and TCP Tahoe in figure 5. In this figure the BER on the reverse channel is taken as 10^{-4} while we consider two different cases of BER on the forward channel namely 10^{-4} and 10^{-6}. Number of link layer retransmissions N, is plotted on the x axis while as in earlier figures the normalized throughput is plotted on the y axis.

We see that when the BER on the forward channel is lower, fewer retransmissions are necessary to achieve the maximum possible throughput. Note that we are plotting only the lower limits of the different curves for the purpose of clarity in this figure. As we know from earlier figures, the difference between lower and upper throughput limits is significant only when N is about 1. As N is increased beyond a certain point there is no significant improvement in the throughput. This certain point though depends on the quality of errors on the forward and reverse channel as can be seen from figure 5.

V. Conclusion

In this paper we have developed an analytical model to study the efficacy of a Stop and Wait link layer retransmission scheme on the performance of four different TCP algorithms. Unlike previous analytical studies we have considered the effect of errors over both the forward and reverse channels. For the scenarios studied, we show that:

- Link-level retransmissions can significantly improve the throughput of TCP and its resilience to packet losses on the forward and reverse links, up to a maximum value of about 5 transmissions per TCP packet, after which increasing the number of retransmissions has relatively little benefit. (Of course, the maximum value beyond which the throughput improvement is negligible is a function of the quality of the forward and reverse channels.)
- Without link layer retransmissions the effect of ACK losses due to errors on the reverse channel cannot be ignored, unless the bit-error rate on the reverse channel is of the order of 10^{-6} or less.

We are continuing to investigate the impact of link-layer retransmissions on TCP behavior for the scenario where the last hop is wireless. Further work includes: (1) considering the impact of alternative link-layer retransmission strategies such as Go Back N and Selective Repeat; (2) investigating the effect of link-level packet fragmentation; and (3) taking into account bursty losses over the wireless link.

References

[1] Farooq M. Anjum and Leandros Tassiulas. On the behavior of different tcp algorithms over a wireless channel with correlated packet losses. pages 155–165, Atlanta, May 1999. ACM Sigmetrics.

[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. DeSimone, M.C. Chuah, and O.C. Yue. Throughput performance of transport-layer protocols over wireless lans. Proc. Globecom, 1993.

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

[7] Chaskar H., T. V. Lakshman, and Madhow U. On the design of interfaces for tcp/ip over wireless. In *IEEE Milcom*, 1996.

[8] 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.

[9] 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.

[10] 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.

[11] S.Lin and D. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1983.