IMPROVEMENT OF TCP PERFORMANCE OVER HETEROGENEOUS NETWORKS USING CONTINOUS ERROR DETECTION BASED LINK LEVEL ERROR RECOVERY

by

Raghavan Anand, Kannan Ramchandran
and Sanjay Shakkottai

Memorandum No. UCB/ERL M00/56

15 August 2000

IMPROVEMENT OF TCP PERFORMANCE OVER HETEROGENEOUS NETWORKS USING CONTINOUS ERROR DETECTION BASED LINK LEVEL ERROR RECOVERY

by

Raghavan Anand, Kannan Ramchandran
And Sanjay Shakkottai

Memorandum No. UCB/ERL M00/56

15 August 2000

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Improvement of TCP Performance over Heterogeneous Networks using Continuous Error Detection based Link Level Error Recovery

Raghavan Anand, Kannan Ramchandran
EECS Department
University of California Berkeley
(anand,kannanr)@eecs.berkeley.edu

Sanjay Shakkottai
ECE Department
University of Illinois at Urbana-Champaign
shakkott@comm.csl.uiuc.edu

Abstract

The performance of TCP over heterogeneous networks is severely degraded due to packet losses due to errors on the wireless channel, which are interpreted to be congestion related, by TCP, creating window cutbacks and inefficient channel utilization. Link level error recovery protocols have been proposed that seek to improve the throughput performance. In this work we present a method for using continuous error detection based link level error recovery for improving the throughput of TCP over heterogeneous networks. The ability to detect error before the end of a packet allows for an increased number of retransmissions over the wireless hop, before a TCP timeout, resulting in an increased probability of successful reception at the mobile host. This method is general enough to integrate with any existing state-of-the-art link level error recovery protocols, and its complexity is low enough to be supportable in low power hand-helds and other information appliances that might proliferate the personal communications market of the future.

1 Introduction

There has recently been widespread interest in the extension of data networks to the wireless domain. As Internet's Transmission Control Protocol (TCP) is currently the predominant data transport protocol on which applications run, many researchers are studying the issues in operating TCP over wireless and heterogeneous networks. It has been shown that the performance of TCP is very poor in the presence of lossy channels[10, 11, 16]. This is because TCP is designed to consider all packet losses as congestion-related, and throttle its rate whenever losses occur. Over a wireless link, packet losses occur due to both channel errors as well as congestion, and these non-congestion-related losses cause TCP to cut back its congestion window resulting in reduced throughput and low channel utilization.

Many schemes have been proposed to improve the throughput performance and a good comparison of the different schemes can be found in[1]. It was observed there that the best improvements are obtained using link layer error recovery with selective acknowledgments. This tries to hide the link-related losses from the TCP sender by using Forward Error Correction (FEC) and/or local retransmissions over the wireless link. The local retransmissions can be done such they are tuned to the characteristics of the wireless link and therefore can provide a significant improvement in throughput performance.

In this work, we present a new link level error recovery protocol using continuous error detection based on *arithmetic coding*. The ability to detect errors in a continuous fashion while transmitting packets over the wireless channel can result in reduced delays in detecting errors which can lead to more retransmission attempts before TCP timeout happens, resulting in improved throughput performance. This idea can easily integrate into existing state-of-the-art link layer recovery protocols like the Berkeley SNOOP Protocol, and improve their performance.

Before we present the network topology and the protocol that we propose, we briefly summarize how TCP works, and the problems that it faces over wireless channels.

2 TCP: A Brief Overview

In this section, we provide a brief overview of the Internet's transport protocol, the Transmission Control Protocol (TCP). We describe its objectives and its congestion control mechanism.

TCP is a connection oriented, adaptive window flow control protocol that uses a "bang-bang" technique for window adaptation. The window size hunts for the "optimum" window size but always overshoots; this results in packet loss, following which there is reduction in the window size. A typical sample path of the TCP congestion window is shown in Figure 1. TCP provides the following services to the higher layers:

(i) Reliable, end to end, in-sequence data transport

Any TCP connection has 2 entities associated with it, the TCP transmitter and the TCP receiver. The transmitter dynamically adjusts its window size as described in Section 2.1. The receiver advertises a maximum window beyond which the transmitter's window cannot increase. When any packet is lost or damaged, the receiver does not acknowledge that packet, causing the transmitter to eventually retransmit the packet.

(ii) Congestion control

TCP detects network congestion in one of two ways:

[a] Packet loss at some intermediate node due to buffer overflow.

[b] Packets or acknowledgments (ACKs) delayed due to congestion resulting in TCP timeout.

In either of these cases, the window drops to one and hence, reduces the load on the network. This is the original TCP algorithm proposed by Van Jacobson [8].

2.1 Window Adaptation in TCP

The transmitter window adapts using the following algorithm:

(i) **Slow Start Phase**: The TCP transmitter window $W(t)$ increases by 1 every time a packet is acknowledged. This effectively means that the window doubles every round trip time. This phase continues till the window reaches the slow-start threshold $W_{th}(t)$. This phase results in an exponential growth of the window (see Figure 1).

(ii) **Congestion Avoidance Phase**: Following slow-start, the TCP transmitter probes for extra bandwidth in the network and hence, the window grows slowly. Each time a packet is acknowledged, the window grows by $\frac{1}{W(t)}$, i.e., roughly, the window grows by 1 every round trip time. This phase continues till loss occurs, or the window reached its maximum possible value (W_{max}).

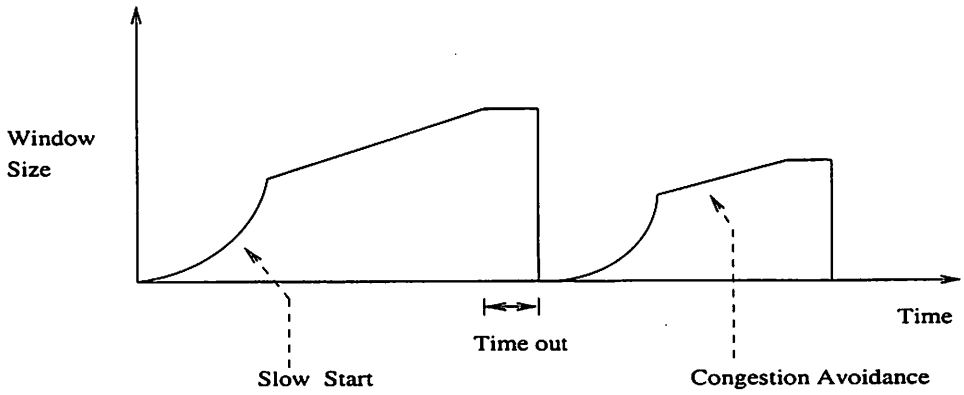

Figure 1: Typical TCP congestion window evolution showing the slow start, congestion avoidance, and loss recovery phases for TCP-OldTahoe

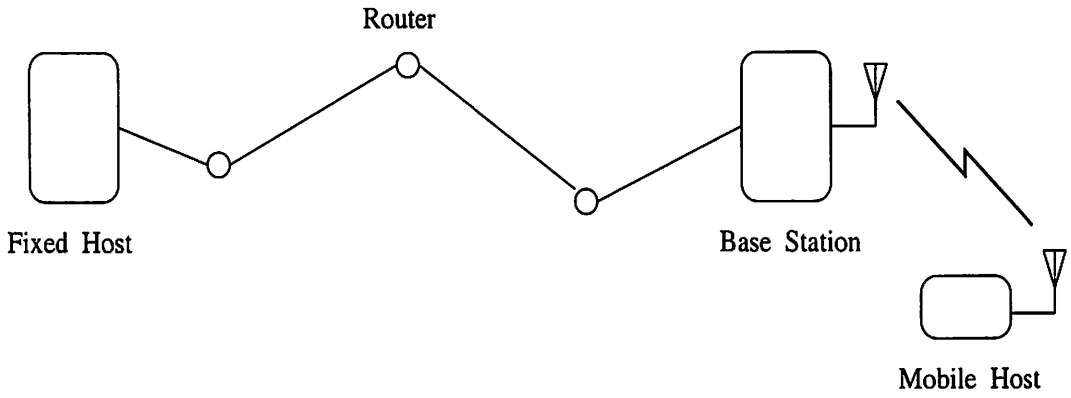

Figure 2: The network consists of many wire-line links connected by routers and a single wireless link

(iii) **Loss Recovery**: There are four versions of TCP that differ primarily in their loss recovery mechanisms. In the earlier versions of TCP, if a packet loss resulted in a TCP timeout, the congestion window was cut back to 1 and the slow start phase re-initiated. In the later versions of TCP the *fast retransmit* and the *fast recovery* algorithms (see [15]) were implemented where loss *does not* necessarily cause a new slow-start; instead, the transmitter tries to recover in the congestion avoidance phase itself.

3 Network Topology

In this section, we present the system that we consider for the proposed protocol.

Consider a TCP connection over a network as shown in Figure 2. We first assume that the wireless link has no errors (which is the same a a purely wire-line case). Here when congestion occurs at a router in the network, buffers overflow causing packet loss. This causes the TCP window to cut back (see Section 2) thereby reducing congestion. However, in practice, the wireless link is lossy. Hence, packet loss occurs even in the absence of congestion. This is interpreted wrongly by TCP as congestive loss causing the TCP window to cut back when, ideally, it should have just retransmitted the lost packet keeping the window unchanged. Hence, TCP throughput suffers in heterogeneous networks. Link level recovery schemes where ARQ and FEC are used over the wireless

link to shape it so that it appears like a lossless link of smaller capacity and larger delay, have been proposed to address this problem. The relative performance of these schemes is compared in[1, 2, 6]. We propose a new link level protocol using an *arithmetic code* based ARQ scheme ([7], [9]) which could result in improved performance of TCP.

An existing state-of-the-art solution that works at the link layer, into which our proposal can be integrated, is the Berkeley SNOOP Protocol. We briefly describe portions of the protocol that are relevant to the integration with our system.

3.1 SNOOP Protocol

In the earlier section, we saw that a TCP-aware link layer protocol achieved significant performance improvement over regular TCP over wireless links. This prompted the authors in[1] to come up with such a scheme, namely, the *SNOOP protocol.*

This runs primarily at the base station to the wireless network. It monitors all the TCP packets and maintains a cache of all the packets sent from the Fixed Host (FH) that haven't yet been acknowledged by the Mobile Host (MH). It also keeps track of all the acknowledgments sent back to the FH and detects loss of packets either by arrival of duplicate acknowledgments, or by a local timeout. Upon detecting a packet loss, it retransmits the lost packet to the MH from its cache, if it has it cached and also hides the packet loss from the sender by filtering the duplicate acknowledgment and not forwarding it to the TCP transmitter.

4 Arithmetic Coding based Link-Level TCP-Aware Error Recovery Protocol

In this section, we discuss a new link-layer error recovery protocol that we propose, that is based on *continuous error detection* and has the potential to improve TCP throughput significantly over wireless links with small round trip delay.

Before we describe the protocol, we discuss Arithmetic Coding, which is used at the link level as an error detection mechanism in the proposed protocol.

Arithmetic coding is discussed extensively in [12]. Details on how it can be used for error detection are presented in [4]. In [7] and [9] it is shown that throughput performance of ARQ protocols can be improved by using continuous error detection.

4.1 Arithmetic Coding

Arithmetic coding is a data compression technique that encodes the data string by creating a code string that represents a fractional value on the number line between 0 and 1. The coding algorithm is symbol-wise recursive, i.e., it operates upon and encodes (decodes) one data symbol per iteration or recursion. On each iteration, the algorithm successively partitions an interval of the number line between 0 and 1, and retains one of the partitions as the new interval. Thus, the algorithm successively deals with smaller intervals, and the code string, viewed as a magnitude, lies in each of the nested intervals. The data string is recovered by using magnitude comparisons on the code string to recreate how the encoder must have successively partitioned and retained each nested subinterval.

This can be best understood with an example as shown in Figure 3. Consider four symbols, a, b, c and ϵ with probabilities of occurrence 0.125,0.25, 0.5 and 0.125. The example shows how a symbol stream bbc..... is encoded. After encoding the first symbol b, the fractional number will lie in the region $[0.125, 0.5)$. The next symbol would divide the coding space for b (i.e., the region $[0.125, 0.5)$) in the ratio of the symbol probabilities, and so on.

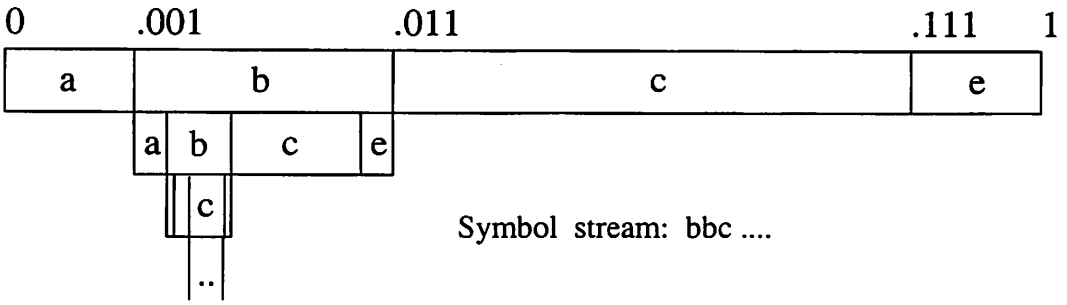

Figure 3: An example of Arithmetic encoding.

4.2 Integrating Error Detection into Arithmetic Coding

As a coding scheme, arithmetic coding suffers from the problem that even a single inverted bit at the receiver (i.e, a single bit error) corrupts all subsequent bits. This can be used to our advantage when our purpose in using arithmetic coding is not source coding, but continuous error detection.

The basic idea is to introduce redundancy by adjusting the coding space so that some parts are never used by the encoder (see [4]). During decoding, if the number defined by the received encoded string ever enters this "forbidden region", then a communication error must have occurred and the decoder can ask for a retransmission.

For example, in Figure 3, say the symbol ϵ is the forbidden symbol that is never used by the encoder. Then, whenever the decoder decodes ϵ, we know that an error has occurred.

By increasing the amount of coding space that the forbidden symbol occupies, it is possible to make statistical guarantees about where the errors may have occurred. That is, it is possible to isolate the location of the error in a statistical sense, to any desired "confidence level" to the previous n bits, where n depends on the amount of invested excess redundancy and the desired confidence level.

Suppose an error occurs at a particular bit position. Then, the error will be detected after n bits with probability $1-(1-\epsilon)^n$ with confidence level $100(1-\delta)\%$. These quantities are related by the expression (see [7], [9]):

$$n = \frac{log_2(\delta)}{log_2(1-\epsilon)} \tag{1}$$

Using arithmetic coding for error control has several advantages:

- The amount of redundancy included in encoded messages can be controlled as a single tunable parameter of the coding process, and, if necessary, varied adaptively to accommodate prevailing channel conditions.
- Error checking can take place continuously as each input bit is processed, so that errors are located quickly, without having to wait for the end of the block as in block codes like CRC.
- An error's location can be pinned down to a small interval to a specified accuracy, thus reducing the amount of material that requires retransmission.

4.3 Protocol Description

In this section, we propose a new protocol that exploits the advantages of continuous error detection mentioned in the earlier section. We use a link-level TCP-aware scheme for this purpose, motivated by its superiority as suggested in[1] and use ideas similar to the SNOOP protocol for ACK filtering.

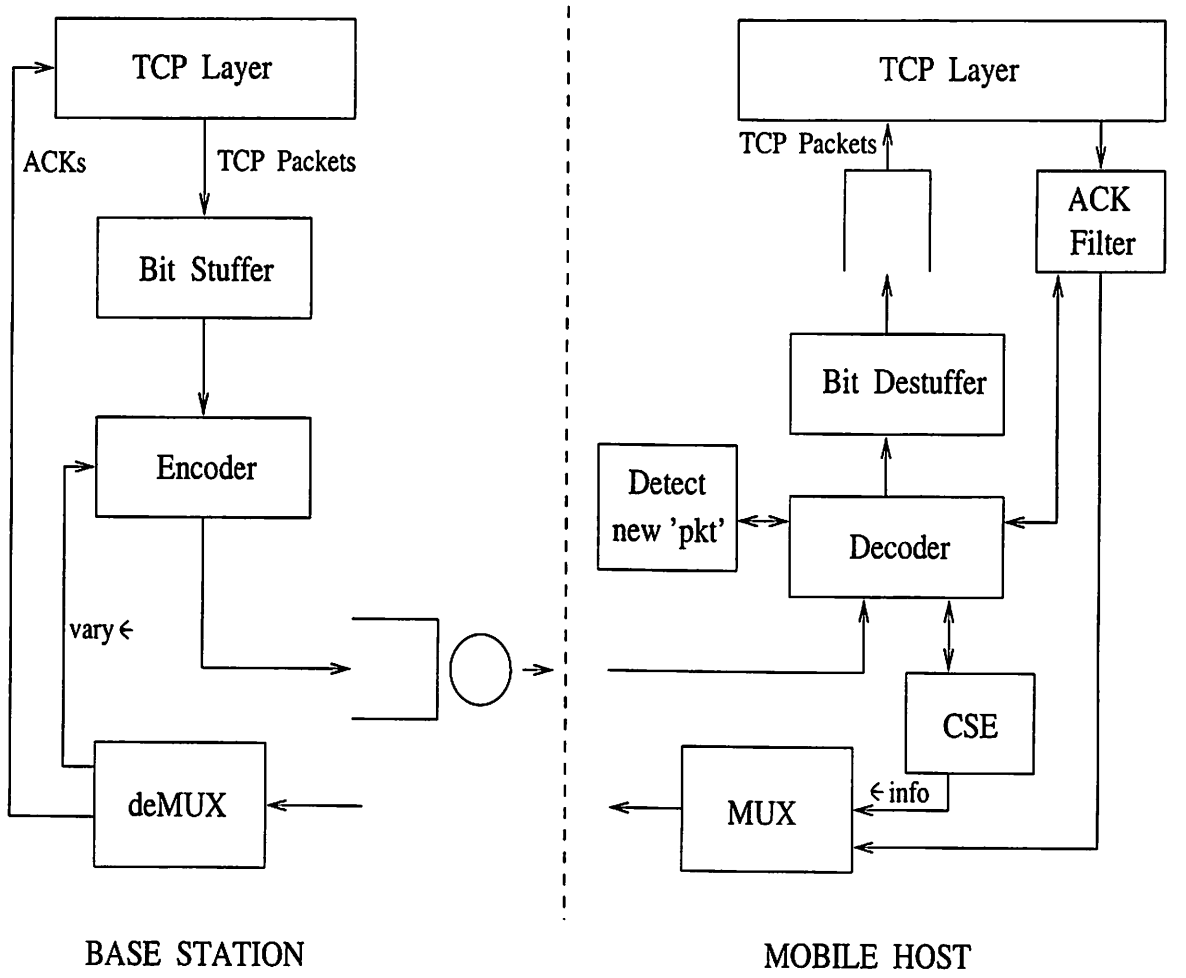

Figure 4: Schematic diagram of the link layer ARQ protocol.

The TCP layer packetizes the data and sends it to the arithmetic encoder at the link level. The encoder encodes the stream using a redundancy ϵ that is known to both the encoder and decoder. The encoded stream is now passed on to the transmit buffer from where it is sent on the air to the receiver.

At the receiver, the arithmetic decoder keeps track of symbols decoded and is on the lookout for the forbidden symbol. If there are no errors in a stream of n bits (where n is as defined in Section 4.2), the decoder passes the bits onto a receive buffer which then sends the packets up to the TCP layer at the receiver.

If there is an error, it flushes its buffer and asks for a retransmission of the past n bits using some redundancy ϵ that it specifies.

Meanwhile, the ACK filter keeps track of the duplicate acknowledgments and suppresses them while the link level tries to recover the lost bits through ARQ.

One issue that arises is how to distinguish between the retransmit stream and the original stream of data. This can be done in two ways. One is to do it as shown in Figure 4. This uses a specific header for the retransmit stream which the decoder can recognize and which can also include details like how many bits are being sent. To ensure that this header sequence does not occur in the data stream itself, we use a bit-stuffer at the base station before encoding and undo it at the receiver before sending the packets up to the TCP layer. Another way of doing this would be to have a specific symbol in the coding space reserved as a control symbol to identify retransmits and any other control information that the base station might want to send to the receiver.

The exchange of control information also necessitates the use of a MUX at the receiver and a deMUX at the base station to separate the ACKs which need to be forwarded to the TCP layer at the base station, from the control information that have to go to the encoder.

An important consideration is that of controlling the redundancy to make full use of this degree of freedom that we have. One example would be to detect when a channel comes out of a deep fade. We know that more the redundancy, faster the error detection. When we detect errors at the receiver even after a couple of retransmits locally from the base station, we know that the channel is in a fade and so the channel error probability is very high. At such times we could increase ϵ to some large fraction like 50% and keep checking for the frequency of errors at the receiver. When we begin to notice that the frequency of errors has gone down after some time, we can immediately reduce the redundancy to the lower set of values that are used during normal transmission.

This idea could be used during retransmits also. When a retransmit is done, it could be done at a higher redundancy than the current ϵ to make the next error detection faster. This will help as we have only a fixed number of local retransmits available before the TCP timer expires, and we would want to have as many tries as possible to avoid a retransmit all the way from the TCP sender which would result in window cutbacks.

The Channel State Estimator (CSE) block in Figure 4 looks at the frequency of errors over some time interval in the past and decides what ϵ the encoder should use for the next transmission. This is then conveyed to the encoder as a part of the control information.

The ACK filter block serves two purposes. Firstly, it keeps track of the packets that are being sent up and the ACKs that are going down. If it detects a duplicate acknowledgment, it suppresses it to allow time for local retransmits from the base station. This feature is similar to the one in SNOOP protocol. In addition to this, it can help the decoder to see if the rare event of it not having detected an error (the other 100δ% of the confidence level discussed in Section 4.2)has actually occurred. It could detect this, if, a packet that it had sent up as being correct, is requested for a retransmit. In such a case, it could adapt its redundancy correspondingly, and ask for a retransmit from the base station buffer starting from this packet onwards.

There could be another rare event that could upset this scheme of things; the case

where the original TCP packet that came to the base station from the TCP sender suffered some error in transit. This would create a retransmit request from the receiver which is not the mistake of the wireless link, but of some other wired link. We do not want the link-level scheme to react to this. So, we have a CRC check done on the TCP packet at the base station itself before sending it down to the encoder. This ensures that all packets which are transmitted on the link are "good". An additional advantage of this is that as the base station drops this packet, the link bandwidth is not wasted by sending a "bad" packet.

There are several other ideas that could be incorporated into this framework to make the protocol more robust and efficient. An interesting idea would be to send incremental parity as retransmit info in an attempt to correct this error (see [5]). This would mean that instead of sending all the requested bits when error occurs, we only send the parity bits for the data in error. This could be integrated in an optimal fashion to result in a hybrid ARQ/FEC scheme which might work better than either of the two individually depending on the situation.

We hope that this framework will result in better TCP throughput than the schemes which have so far been proposed. We plan to simulate this scheme shortly.

References

[1] Hari Balakrishnan et. al., " A Comparison Mechanism for Improving TCP Performance over Wireless Links" *IEEE Tran. on Networking, December 1997.*

[2] Hari Balakrishnan et. al., " TCP Improvements for Heterogeneous Networks: The Daedalus Approach" Manuscript, University of California, Berkeley.

[3] T. Bell, J. Cleary, and I. Witten, *"Text Compression"* Prentice Hall, 1990.

[4] C. Boyd, et. al., "Integrating Error Detection into Arithmetic Coding" IEEE Tran. on Comm., pp. 1-3, Vol 45, No. 1, Jan 1997.

[5] V. Chande, H. Jafarkhani, & N. Farvardin, "Joint Source-Channel Coding of Images for Channels with Feedback" *ITW 1998, San Diego, California.*

[6] H. Chaskar, T. V. Lakshman, U. Madhow, "TCP over Wireless with Link Level Error Control: Analysis and Design Methodology" To appear in *IEEE Tran. on Networking.*

[7] J. Chou & K. Ramchandran, " Improving Throughput Performance in ARQ Protocols Through Use of Continuous Error Detection" Manuscript, University of Illinois at Urbana-Champaign.

[8] Van Jacobson, " Congestion avoidance and control" *Proc. ACM Sigcomm'88,* August 1988.

[9] I. Kozintsev, J. Chou and K. Ramchandran, "The Role of Arithmetic Coding Based Continuous Error Detection in Digital Transmission Systems" Manuscript, University of Illinois at Urbana-Champaign.

[10] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link" To appear in *IEEE Tran. on Networking.*

[11] A. Kumar & J. Holtzman, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Mobile Radio Link" *SPCOM '97, Bangalore, India, August 1997.*

[12] Glen G. Langdon, Jr., "An Introduction to Arithmetic Coding" *IBM J. Res. Develop., Vol 28, No 2, March 1984, pp 135-149.*

[13] T.V. Lakshman and U. Madhow, " Performance Analysis of Window-Based Flow Control using TCP/IP: the Effect of High Bandwidth-Delay Products and Random Loss" *IFIP Transactions* C-26, High Performance Networking V, pp. 135-150, North-Holland, 1994.

[14] Sheldon Ross, *"Stochastic Processes"* John Wiley &Sons, 1996.

[15] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms" *Network Working Group, RFC 2001 available at ftp://nic.ddn.mil/rfc.*

[16] M. Zorzi & R. R. Rao, "Effect of Correlated Errors on TCP" *CISS '97, Baltimore, March 1997.*