Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

SIMULATION OF TRANSPORT PROTOCOLS OVER WIRELESS COMMUNICATION NETWORKS

Hala ElAarag
Mostafa Bassiouni

School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816, U.S.A.

ABSTRACT

In this paper, we describe a general-purpose communication network simulator that we designed to examine the performance of transport protocols over wireless networks. The general-purpose simulator can be used to study various aspects of performance of communication networks. In particular, we use it to examine, evaluate and predict the performance of wireless networks under a variety of the most commonly used transport protocols. We also examine the use of newly suggested protocols specifically designed for wireless networks. In this paper, we present our models for each protocol. We discuss various performance measures that can be studied using our simulator. These measures are hard to evaluate with analytical models.

1 INTRODUCTION

The explosive growth of wide-area cellular systems and local-area wireless networks are just the beginning of "the wireless revolution". Mobile computers and their wireless communication links will be an integral part of the future internetworks. Making truly tetherless computing possible demands that we carefully evaluate, enhance and perhaps re-design our networks, systems, algorithms, and applications.

Generally, wireless networks are composed of a wired backbone network and a wireless network. The wireless network is geographically divided into cells, each of which contains a base station (BS) that provides a connection end-point for the mobiles. The base stations are connected to the wired infrastructure with fixed hosts (FH). They provide a gateway for communication between the wireless network and the backbone interconnect. Figure 1 illustrates a typical wireless network topology. Mobile networks are fundamentally different from conventional wired computer networks. One reason, among many others, is that wired links have low bit error rates, as opposed to wireless links that suffer from high bit error rates (BER).

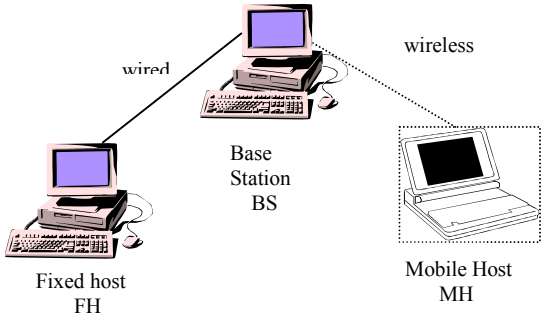

Figure 1: Typical Wireless Network Topology

UDP and TCP form the very core of today's Internet. However, they were designed for networks made up of fixed hosts and wired links. Thus when a packet is lost regular TCP, for example, assumes that it is due to congestion. If regular TCP is used on a mobile network, it will encounter packet losses that may be unrelated to congestion. Nonetheless, these losses will trigger congestion control procedures at the fixed host. These procedures will result in significant reductions in throughput and unacceptable interactive delays for active connections. Thus severely degrading performance.

Several modifications have been proposed to the TCP loss-recovery and congestion control mechanism to improve data throughput in the event of random loss. Reno TCP (Jacobson 1990, Stevens 1997) is an example of that. It is the most widely used version of TCP on the Internet.

Several researchers proposed protocols to enhance the performance of TCP on wireless networks. One example is a mechanism called split TCP (Yavatkar and Bhagawat 1995, Bakre and Badrinath 1997, Brown and Singh 1997).

2 GENERAL PURPOSE COMMUNICATION NETWORK SIMULATOR

We developed a general-purpose discrete event simulator that models a multi-stage, multi server queuing network. We used Concurrent C (Gehani 1986) as our simulation language. Concurrent C is an extension of the C programming language that provides concurrent programming facilities. A concurrent C program is structured as a set of processes that execute concurrently. In our simulator, we model each queue and each server as a Concurrent C process. This is a natural way to simulate communication networks. In real networks, queues and servers run independently and interact when necessary, so as the processes. For example, the simple queuing system in Figure 2 uses three processes: one for the source, one for the queue, and one for the server. Each process performs a well-defined series of operations independently. Sometimes one of these operations involve interacting with another process, e.g. a server taking the next item form the queue.

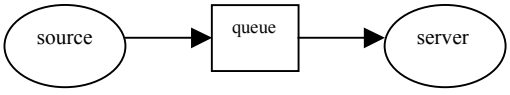

Figure 2: A Simple Queuing Network

One important step in Concurrent C programming is to identify the interactions between the processes and map them into transactions calls. It is convenient to divide the processes into two categories: active processes such as sources and servers, and passive processes such as queues. Passive processes usually wait for requests from active processes. For example, the queue process has two types of transactions: a "put" transaction and a "take" transaction.

2.1 STRUCTURE OF THE SIMULATION PROGRAM

Our simulation program has seven basic types of processes.

The **scheduler** process manages the simulated time. It maintains the simulated clock and advances it appropriately. The simulated clock is independent of the real-time clock. For each delay request from a process, the scheduler determines the simulated time at which to re-activate the process, and saves this in an "activation request" list. When all the processes are waiting, the scheduler picks the next process to run, advances the simulated clock to that time, and reactivates the process. The simulated clock advances only when all processes are waiting; thus any computation done by a process takes place in zero simulated time. If several processes are waiting to be reactivated at the same simulated time, the scheduler awakens all of them simultaneously. Thus at any given time each process has to be in one of three states. Waiting for an explicit delay from the scheduler (*waiting*), computing in zero simulated time (*active*), or waiting for an event other than a delay request from the scheduler (*passive*).

The **queue** process can have several clients. Clients are either consumers ("takers") or producers ("putters"). For example, a source process is a producer, a server is a consumer for its input queue and a producer for its output queue. The queues are of finite capacity with a DROPTAIL policy. That is, arriving packets more than the maximum number of packets that the queue can hold are dropped.

The **source** process generates the packets according to the transport protocol used. We implemented several transport protocols. User Datagram Protocol (UDP) and several implementations of Transmission Control Protocol (TCP): Tahoe TCP, Reno TCP and split TCP. These protocols are described in Section 3.

The **server** process is a process that merely transfers packets from their input queue to their output queue(s). If the server is defined to be a base station then it also determines which packets are lost due to the wireless link BER. Packets are lost due to the wireless (lossy) link according to an exponential distribution. The choice of the exponential distribution for packet loss is motivated by the work in (Balakrishnan et al. 1997). Balakrishnan et al. showed that the performance results using an exponential distribution for errors are also applicable under other patterns of wireless loss. Bursts are also possible in this error model because the standard deviation of the exponential distribution is equal to its mean.

The **sink** process can be one of two kinds. A TCP sink or a UDP sink. The TCP sink (MH) is responsible for returning ACKs to the peer TCP source after processing it. It generates one ACK per packet received. The ACK returned represents the last in order packet it received. The UDP sink discards each packet after processing it.

The **source-server** process performs the job of a source and a server that we described earlier. It generates packets according to the transport protocol used and also determines if the packets are lost due to the wireless link BER. The server-source is used in our split TCP model as explained in Section 3.3.

The **main** process creates the other processes and connects them together appropriately. Examples of parts of the main process are provided in the next Section.

3 TRANSPORT PROTOCOLS

3.1 UDP

UDP is an unreliable, connectionless transport protocol. It does not use acknowledgements to make sure messages arrive, it does not order incoming packets and it does not provide feedback to control the rate at which information flows between the machines. Thus, UDP packets can be lost or arrive faster than the recipient can process them. Figure 3

shows the model for UDP for the network topology in Figure 1. Packets are issued at FH (a source process) according to a Poisson distribution, λ represents the average of the inter-arrival time of the packets at FH. The service rates at BS (a server process) and MH (a sink process) are exponentially distributed with means 1/μ1 and 1/μ2 respectively. Packets are lost due to the wireless (lossy) link according to an exponential distribution with mean (1/p). The following code is part of the main process that creates the topology of Figure 3:

```
/* create queues and servers */
 q1 = create queue(s, queue_size, makeName("Q1"));
 q2 = create queue(s, queue_size, makeName("Q2"));
 create source(s, q1, iit, servt1, sim_time, makeName("Fixed host"));
create server(s, q1, q2, servt2, d2,p,makeName("Base Station"));
create sink(s, q2, c_nullpid, servt2,d2, makeName("Mobile host"));
```

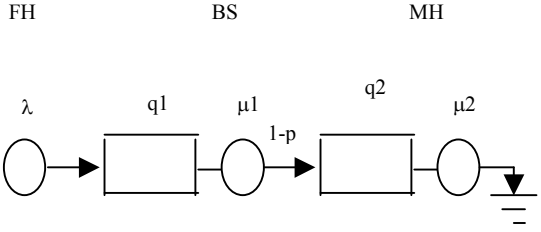

Figure 3: UDP Model

3.2 TCP

TCP is a connection-oriented protocol. It is a reliable transport protocol that adapts to the network requirements. It regulates the number of packets it sends by inflating and deflating a window. To do that the TCP sender uses the cumulative acknowledgements (ACKs) sent by the receiver. TCP also adapts to problems on the wired link. The main problem is the delay caused by packet losses due to congestion. Our TCP model is shown in Figure 4. This is illustrated by the following code of the main process:

```
/* create queues and servers */
q1 = create queue(s, queue_size, makeName("Q1:FH_BS"));
q2 = create queue(s, queue_size, makeName("Q2:BS_MH"));
q3 = create queue(s, queue_size, makeName("Q3:MH_BS"));
q4 = create queue(s, queue_size, makeName("Q4:BS_FH"));
create source(s, q1,q4, servt1, d1,nGen,implem,init_cwnd, dupack_thresh,
ssthresh,W_max, makeName("Fixed host"));
create server(s, q1, q2, servt2, d2, p,makeName("Base Station"));
create sink(s, q2, q3, servt3, d2, makeName("Mobile host"));
create server(s, q3, q4, servt4,d1, p,makeName("Base Station_ack"));
```

3.2.1 Tahoe TCP

The congestion control scheme in Tahoe TCP in (Jacobson 1988) implementation has three main parts:

1. Slow-start
2. Congestion avoidance
3. Fast Retransmit.

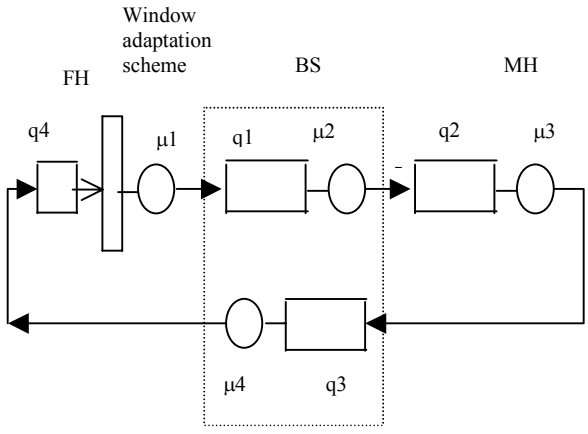

Figure 4: TCP Model

The **Slow-start algorithm** works as follows: the TCP sender starts with a congestion window (*cwnd*) that is equal to 1. For each received ACK, TCP exponentially increases the window until it is equal to a threshold (*ssthresh*), then it enters the **congestion avoidance** phase where it continues to increase its *cwnd* linearly until it reaches the receiver's maximum advertised window.

TCP continually measures how long acknowledgements take to return to determine which packets have reached the receiver, and provides reliability by retransmitting lost packets. For this purpose, it maintains a running average of this delay (round trip delay) and an estimate of the expected deviation from this average. If the current delay is longer than the timeout interval, TCP assumes that the packet was lost. TCP then retransmits the lost packet.

TCP also assumes that the packet was lost if the sender receives a number of duplicate acknowledgements (usually three). This is because the receiver acknowledges the highest *in-order* sequence number. If it receives *out-of-order* packets, it also generates acknowledgements for the same highest *in-order* sequence number and that results in duplicate acknowledgements. TCP then activates the **Fast Retransmit algorithm**. The Fast Retransmit algorithm assumes that the missing packet starts with the sequence number immediately after the number acknowledged by the duplicate ACK' s, and thus retransmits it. TCP reacts to any packet lost by:

1. Dropping *ssthresh* into half the current window or 2 (whichever is larger) to reduce the amount of data.
2. Resetting its transmission (congestion) window size to 1, thus activating the slow-start algorithm to restrict the rate at which the window grows to previous levels.
3. Resetting the retransmission timer to a backoff interval that doubles with each consecutive timeout according to Karn's exponential timer

backoff algorithm (Karn and Partridge 1991). This also results in the reduction of the traffic load at the intermediate links and therefore controls the congestion in the network.

Roundtrip Time Estimation and Retransmission Timer Selection: A timeout process is created by the source process to calculate the timeout timer. Four variables are used to estimate the roundtrip time and set the retransmission timer (Fall and Floyd 1996): *rtt, srtt*, *rttvar, and backoff*. Roundtrip time sample arrives with new ACKs. The *rtt* sample is computed as the difference between the current time and a time field in the ACK packet, which is equal to the time the packet was issued at the source. When the first sample is taken, its value is used as the initial value for *srtt.* Half the first sample is used as the initial value for *rttvar.* For subsequent samples, the values are updated as follows:

$$srtt = \frac{7}{8} \times srtt + \frac{1}{8} \times rtt$$

$$rtt\,\mathrm{var} = \frac{3}{4} \times rtt\,\mathrm{var} + \frac{1}{4} \times \left| rtt - srtt \right|$$

The *backoff* is initially 1. The retransmission timer is set to the current time plus

$$\mathrm{backoff} \times \left(\mathrm{srtt} + 4 \times \mathrm{rttvar} + 1 \right)$$

The *backoff* factor doubles each time a timeout occur to a maximum of 64 (Karn's exponential timer backoff (Comer 1991).

3.2.2 Reno TCP

Reno TCP is like Tahoe TCP except it includes *fast recovery*. The TCP sender enters fast recovery if it receives three duplicate acknowledgments. The sender retransmits the lost packet and reduces *ssthresh* by half. Unlike TCP Tahoe, the sender then does not enter slow start. It reduces the value of the congestion window *(cwnd)* by half, then increments it by one for each duplicate acknowledgement received. When a "new" ACK is received, the sender exits fast recovery, sets *cwnd* to *ssthresh* and enters the congestion avoidance phase where it increases the window linearly. A "new" ACK is an ACK with a value higher than the highest seen so far i.e. a non-duplicate ACK.

3.3 Split TCP

The main idea behind the split connection approaches is to isolate mobility and wireless related problems from the existing network protocols. This is done by splitting the TCP connection between the mobile host and the fixed host into two separate connections: a wired connection between the fixed host and the base station, and a wireless connection between the base station and the mobile host. In this way, the wired connection does not need have to include changes in the existing software on the fixed host while the wireless connection can use a mobile protocol specialized to provide better performance.

In our split mechanism model, the TCP connection is split at BS. There are two TCP connections: one connection between FH and BS where FH is the source and BS is a sink, and another connection between BS and MH where BS is the source (server-source process) and MH is a sink. Bulk data are transferred via ftp from FH to BS and from BS to MH. Packets are deleted from the buffer only when BS receives an Ack from MH indicating that the packet successfully arrived. Since Acks are cumulative, the arrival of a new (non duplicate) Ack causes the removal of all packets in the buffer with packet numbers less than or equal to the number of the arriving Ack. Our model for split TCP is shown in Figure 5. It is illustrated by the following code from the main process:

```
/* create queues and servers */
q1 = create queue(s, queue_size, makeName("Q1:FH_BS_ack"));
q2 = create queue(s, queue_size, makeName("Q2:BS_MH"));
q3 = create queue(s, queue_size, makeName("Q3:MH_BS"));
q4 = create queue(s, queue_size, makeName("Q4:BS_FH"));
q5 = create queue(s, queue_size1, makeName("Q5:FH_BS"));
buff=create queue(s,buff_size, makeName("BS: buffer"));

 ss=create server_source(s, q2, q5,q3,buff, servt2,d2, p,nGen, implem,
init_cwnd, set_ssthresh, dupack_thresh,
ssthresh_wireless,W_max,nPkts,makeName("Base Station"));
create source(s, ss,q4,q1,q5, servt1,d1, nGen,implem, dupack_thresh,
ssthresh_wired,W_max, makeName("Fixed host"));
 create sink(s, q2, q3, servt3,d2, nPkts, makeName("Mobile host"));
 create sink(s, q1, q4, servt4,d1, nPkts, makeName("Base Station_ack"));
```

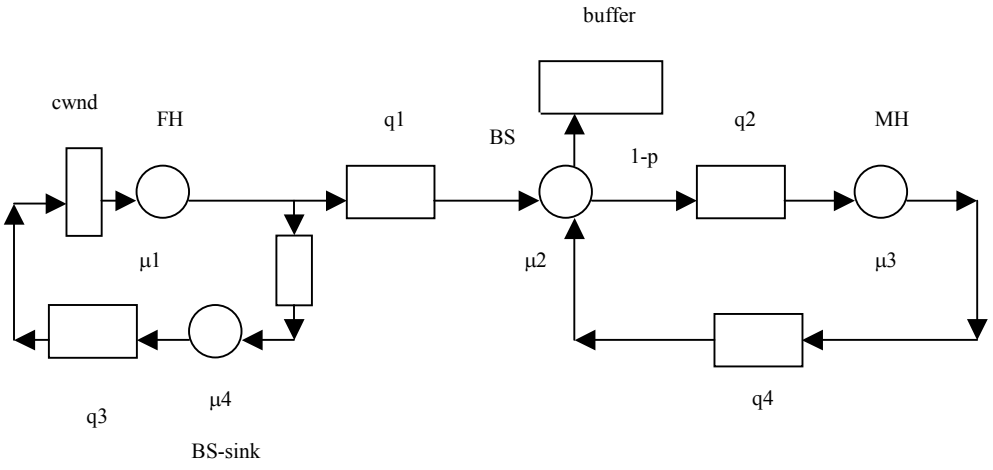

Figure 5: Model of Split TCP Connection

4 PERFORMANCE MEASURES

There are many performance measures that we can study using our simulator. These measures are hard to obtain using analytical models. This makes simulation very appealing in the study of communication networks. Some of the measures we use are throughput, goodput, average packet delay, transfer time, inter-arrival time of packets and its jitter. The following is our definition to these measures:

- Throughput is the number of packets per second received by MH.
- Goodput is the number of useful data to the total data sent. Goodput is an indication of the utilization of the network.
  - Wireless goodput is the ratio between the number of packets received by MH to those sent by BS. That is the goodput of the wireless link.
  - Wired goodput is the number of packets received by BS to that sent by FH.
- Average packet delay is the average time the packet stays in the system. It is calculated from the time the packet is issued at FH until the time its corresponding ACK is received.
- The transfer time is the time needed to transfer certain number of packets.
- the inter-arrival time of packets is the average inter-arrival time of packets arriving at MH
- delay jitter is the standard deviation of the inter-arrival time of packets. The latter two measures are very important to delay sensitive data like multimedia application that have strict requirements.

5 EXAMPLE SIMULATION

We ran experiments to evaluate and compare the performance of the different transport protocols over wireless networks. We tested the protocols and different traffic loads and protocol and network parameters. This Section illustrates an example of a simulation we ran to study the effect of the different BER of the wireless link on Tahoe TCP (end-to-end) and split TCP. We considered BER ranging between 10^{-6} and 10^{-4} corresponding to packet loss probability of 0.001 and 0.1 respectively. We chose the network topology in Figure 1. This topology is motivated by the recent experiments of TCP performance over wireless mobile links; see for example (Bakshi et al. 1997, Balakrishnan et al. 1997). The simulation environment used by Fall and Floyd (Fall and Floyd 1996) to compare several TCP implementations on a wired network is also similar to that of Figure 1 but with BS replaced by a finite-buffer drop-tail gateway and MH replaced by a wired data receiver. In our simulation environment, the wired link between the fixed host and the base station is of bandwidth 1.5Mb and delay (D1) of 10ms. The wireless link between the base station and the mobile host is of bandwidth 0.8Mb and delay (D2) of 100ms. For TCP parameters, we had ssthresh = 32, Wmax = 50, duplicate Ack threshold = 3, packet size = 1000 bytes, Ack size = 40 bytes, and Queue sizes = 50 packets. Bulk data are transferred via ftp from FH to MH. Figures 6 and 7 show some of our results.

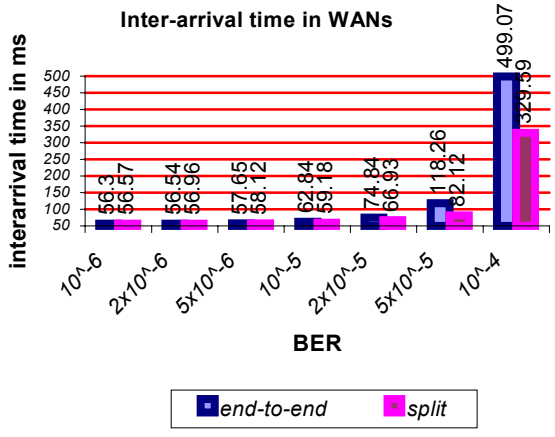

Figure 6: Inter-arrival Time of Packets. vs. BER of End-to-end and Split Mechanisms in WANs

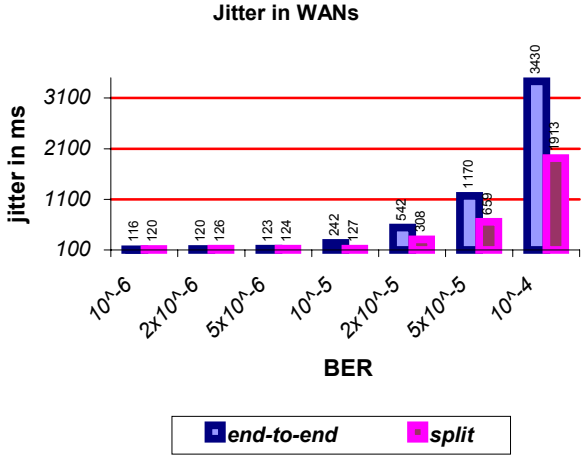

Figure 7: Jitter vs. BER of End-to-end and Split Mechanisms in WANs

From our simulations, we notice that the higher the BER, the less the throughput and the wireless (and wired in case of end-to-end) goodput, and the higher the average inter-arrival time of packets at MH and its jitter, for both split and end-to-end TCP in both LANs and WANs. We also notice that the impact of increasing BER on split and end-to-end TCP was the same in LANs and WANs. Increasing BER one order of magnitude from 10^{-6} to 10^{-5} has a slight impact on performance of both implementations. Both end-to-end and split TCP implementations had comparable performance, in terms of the four measures we used here, for BER higher than 10^{-5} in LANs and WANs. Increasing BER one order of magnitude from 10^{-5} to 10^{-4} has a tremendous impact on performance. Split TCP resulted in better overall performance for BER higher than 10^{-5}. For example, the throughput of split at BER = 2 x 10^{-5} in LANs was 91.64 pkts/sec, almost double that of end-to-end, of throughput 53.35 pkts/sec. There is also a colossal difference between the performance of split and end-to-end in terms of the two multimedia measures (inter-arrival time of packets and its jitter). Split TCP behaved much better in those two metrics which makes it more suitable for the transfer of multimedia applications as shown in Figures 6 and 7. Also the split-connection isolated the TCP source (FH) from the wireless losses. BS performed all the retransmissions resulting from the packet losses on the wireless link, resulting in a wired goodput equal to 1.0. We also need to note here that in case of split connection, the congestion window of FH becomes large and does not suffer any deflating, thus stays large resulting in high bandwidthutilization on the wired link. Contrary to the congestion window of FH in case of end-to-end, where it fluctuates rapidly (ElAarag and Bassiouni 1999).

6 CONCLUSION

In this paper we presented a general-purpose discrete event network simulator. The simulator can be used to study various aspects of communication networks. We used this simulator to study the performance of several transport protocols over wireless networks. We presented numerous performance measures that we can study using the simulator. These measures are hard to study with analytical models.

ACKNOWLEDGMENT

This work has been supported by ARO under Grant No. DAAH04-95-1-0250 and by an I-4 Corridor grant from Harris Corporation and the State of Florida. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agencies or the University of Central Florida.

REFERENCES

Bakre A. and Bardinath B.R. 1997. Implementation and performance evaluation of indirect TCP, *IEEE Transactions on Computers*, Vol. 46, No. 3, March 1997, pp. 260-278.

Bakshi B. et al. 1997. Improving performance of TCP over wireless networks, *17th International Conference on Distributed Computing Systems*, pp.365-373.

Balakrishnan H., Padmanabhan V., Seshan S., and Katz R. 1997. A comparison of mechanisms for improving tcp performance over wireless links. *IEEE/ACM Transactions on Networking*, Vol.5, No. 6, Dec. 1997.

Brown K. and Singh S. 1997. M-TCP: TCP for mobile cellular networks. *Computer Communication Review*, July 1997, pp. 19-43.

Comer D. 1991. *Internetworking With TCP/IP,* Vol.1, 2 &3, New York: Prentice Hall.

ElAarag H. and Bassiouni M. 1999. Transport control protocols for wireless connections. *IEEE 49th Vehicular Technology Conference*, pp. 337-341.

Fall K. and Floyd S. 1996. Simulation based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, 26(3), pp.5-21.

Gehani N. H., and Roome W. D. 1986. Concurrent C *Software- Practice and Experience*.

Jacobson V. 1988. Congestion avoidance and control". *SIGCOMM Symposium on Communications Architectures and Protocols,* pp. 314-329. <ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z> for an updated version.

Jacobson V. 1990. Modified TCP congestion avoidance algorithm. Technical Report 30, April 1990. Available at ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt

Karn P., and Partridge C. 1991. Improving round-trip time estimates in reliable transport protocols. *ACM Transactions on Computer Systems*, 9(4):364-373.

Stevens W.R. 1994. *TCP/IP Illustrated*, vol.1, New York: Addison-Wesley.

Yavatkar R. and Bhagawat N. 1995. Improving end-to-end performance of TCP over mobile internetworks. *Proceedings of the Workshop on Mobile Computing Systems and Applications,* Santa Cruz, CA, pp. 146-152.

AUTHOR BIOGRAPHIES

HALA A. ELAARAG is a Ph.D. candidate in the school of Electrical Engineering and Computer Science at the University of Central Florida. She received her M.Sc. and B.Sc. from Alexandria University, 1991, 1989 respectively. She is IEEE member. Her research interests include wireless networks, parallel and distributed systems, RISC architectures. Her email and web addresses are <elaarag@cs.ucf.edu> and <www.cs.ucf.edu/~elaarag/>.

MOSTAFA BASSIOUNI is a professor in the school of Electrical Engineering and Computer Science at the University of Central Florida. His research interests include distributed simulation, computer networks, and real-time protocols. His email and web addresses are <bassi@cs.ucf.edu> and <www.cs.ucf.edu/csdept/faculty/bassi.html>