

Performance evaluation of TCP implementations in wireless networks

Hala ElAarag and Mostafa Bassiouni *

School of Computer Science
University of Central Florida
Orlando, FL 32816

ABSTRACT

The performance of TCP has been well tuned for traditional networks made up of wired links and stationary hosts. Mobile networks, however, differs from conventional wired computer networks and usually suffer from high bit error rates and frequent disconnections due to handoffs. In this paper, we present simulation results for the performance of various TCP implementations in the presence of a wireless link. To concentrate on the mobility and reliability aspects of the wireless connection, our simulation tests used sufficiently large buffer sizes in the fixed host and the base station of the TCP connection. The results show that throughput of the TCP connection is largely influenced by the link-up period of the wireless link. By varying the link-up and the link-down periods, it is possible to obtain better throughput at higher disconnection probability. For example, the throughput of TCP Reno with disconnection probability of 28.6% and a link-up period of 5 is better than the throughput with disconnection probability of 9% and a link-up period of less than 3. The paper presents timing graphs tracing the movement of packets and acknowledgments between the fixed and mobile hosts. Dropped packets or acknowledgements shown in these graphs are the result of mobile disconnection or wireless bit errors and not because of buffer congestion. Unlike wired networks, Reno TCP was found to perform better than Sack in the wireless mobile environment.

Keywords: Transport control protocol, performance evaluation, wireless networks, mobile disconnection, bit error rate.

1 INTRODUCTION

Most current network applications that require reliable transmission use TCP as their transport protocol. Mobile computer users would like to use the same applications over the wireless link. TCP provides a reliable connection-oriented transport service due to its in-built algorithms for congestion control and avoidance. Stevens ¹ and Comer ² provide details of the TCP protocol. With the popularity of TCP in wired networks, and the increasing interest in mobile computers, many researchers ^{3, 4, 5, 6} are now interested in how TCP would perform in a mobile wireless environment.

In this paper, we investigate the performance of different TCP implementations in wireless environments. Four implementations are investigated in this paper: Tahoe, Reno, New-Reno and Sack. In the next sections, we describe the simulation environment used to evaluate the four implementations, discuss the parameter values used in our tests and present and analyze the performance results for the four implementations.

* Correspondence: elaarag@cs.ucf.edu, bassi@cs.ucf.edu

2 TCP IMPLEMENTATIONS

2.1. Tahoe TCP

The Tahoe TCP implementation consists of *slow start*, *congestion avoidance* and *fast retransmit* algorithms⁷. In the slow start algorithm the TCP sender starts with a congestion window (*cwnd*) equals to 1. It exponentially increases the window by incrementing *cwnd* for every received acknowledgement. When *cwnd* is equal to a threshold (*ssthresh*), the TCP sender enters the congestion avoidance where it increases the window linearly rather than exponentially. If the TCP sender did not receive an acknowledgment for a certain calculated period (timeout period) or it received three duplicate acknowledgements, it assumes that a packet has been lost. Thus, it enters the fast retransmit algorithm which sends the assumed lost packet, drops the *ssthresh* into half and activates the slow start algorithm.

2.2 Reno TCP

Reno TCP is like Tahoe TCP except it includes *fast recovery*^{8,9}. The TCP sender enters fast recovery if it timeouts or receives three duplicate acknowledgements. The sender then retransmits the lost packet and reduces *ssthresh* by half. Unlike Tahoe TCP, 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 that acknowledges the packet sent before the duplicate ACKs were received i.e. an ACK with a value higher than the highest seen so far.

2.3 New-Reno

This implementation is basically a modification over Reno TCP proposed by Hoe¹⁰. New-Reno helps improve the performance of Reno when multiple packets are lost from the same window. To achieve that, it modifies the action taken upon receiving a "new" ACK. Unlike Reno, New-Reno does not exit fast recovery unless all data transmitted at the start of the fast retransmission phase have been acknowledged. Thus, it only exits fast recovery when it receives an ACK for the highest sequence number sent. Thus, "new partial" ACKs (a new ACK that represent that some but not all of the outstanding packets were received) are used as indication that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. When multiple packets are lost in the same window, Reno recovers by waiting for serial retransmission timeout. This is because in Reno, the reception of a new partial ACK causes the sender to exit fast recovery by "deflating" the window.

2.4 Sack

The Sack TCP implementation¹¹ has been introduced to further improve TCP performance by allowing the sender to retransmit packets based on the selective ACKs provided by the receiver. The implementation constitutes a Sack field that contains a number of Sack blocks. The first block reports the most recently received packets. The additional blocks repeat the most recently reported Sack blocks. Sack TCP uses the basic congestion control algorithms and uses retransmit timeouts as a last option for recovery. The main difference is the way it handles the loss of multiple packets from the same window, in Fast Recovery. Like Reno, Sack enters fast recovery upon receiving duplicate ACKs; it retransmits a packet and cuts its congestion window in half. In addition to that, Sack has a new variable called the *pipe*, and a data structure called the *scoreboard*. The *pipe* is incremented when the sender sends a new or a retransmitted packet. It is decremented when the receiver receives a new packet. This is indicated when the sender receives a duplicate ACK with a SACK option. The *scoreboard* stores ACKs from previous Sack options, allowing the sender to retransmit packets that are implied to be missing at the receiver. Like New-Reno, the sender exits fast recovery when all the outstanding data are acknowledged.

3 SIMULATION ENVIRONMENT

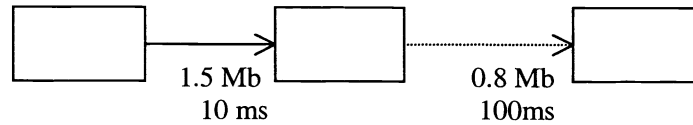


Figure 1. Simulation Topology

The network topology used in our performance tests is similar to what is often used in the literature³. It consists of a fixed host (FH), a base station (BS) and a mobile host (MH) as shown in Figure 1. The wired link between FH and BS has a bandwidth of 1.5Mb and delay (D1) of 10ms. The corresponding figures for the wireless link between BS and MH are a bandwidth of 0.8Mb and a delay (D2) of 100ms. FH is the source of the TCP connection and MH is the sink. FH and BS use a DropTail queue; BS merely transfers data received from FH to MH. Bulk data are transferred via ftp from FH to MH. The error model of the wireless link is characterized by two states: a good state and a bad state. In the good state, the wireless link does not suffer any losses, while, in the bad state the packets get corrupted. Both states are exponentially distributed with means α_g and α_b respectively. MH has an exponential disconnection probability model. The link-up state, where MH is connected is exponentially distributed with mean link-up time (m_1). The link-down state, where MH is disconnected is also exponentially distributed with mean link-down (m_2). MH has a disconnection probability p , where $p = m_2 / (m_1 + m_2)$. The simulation model has been implemented using the Network Simulator (NS) from Lawrence Berkeley Labs. The following are the parameters used in the simulation tests reported in this paper (unless otherwise stated):

```
window size = 50
ssthresh = 32
Packet size = 1024 bytes
size of ACK = 40 bytes
queue size = 50
```

The large queue size allows us to focus only on the loss of packets due to link error rate and mobile disconnections. Times reported in this paper are calculated to the nearest 100ms.

The graphs are generated by tracing the packets between FH and MH. The x-axis shows the time of arrival or departure of the packets. The y-axis shows the packet number *mod* 100. The square on the graph represents the enqueueing of packet at FH and the arrival at BS. Thus, when a packet experience no queuing delay, it generates two squares so close on the graph that they might appear as a single mark. Packets having queuing delay at FH generate two squares having the same packet number spaced by the delay on the link plus the queuing delay at FH. The circle represents the ACK received by FH. This ensures the end-to-end semantics of TCP. The "x" on the graph represents the dropped packets, and the square with line in the middle represents the dropped ACKs. Dropped packets and ACKs are due to the disconnection of MH.

4 PERFORMANCE RESULTS

Bit errors on wireless links are found to be bursty and frequent. We shall start by investigating the effect of BER on the performance of TCP. Recent studies⁵ indicate that BER may be no worse than 10^{-5} . Forward Error Correction codes (FEC) and retransmission schemes at the link layer may reduce the BER

by one (possibly two) orders of magnitude ⁶. Table 1 shows the results of our simulation for Tahoe TCP with BER of 10^{-5} and 10^{-6} . Accordingly, in the former case for TCP packets (BS to MH) $\alpha_g=100$, $\alpha_b = 8$ and for ACKs (MH to BS) $\alpha_g=1000$, $\alpha_b = 3$. In the latter case, for TCP packets $\alpha_g=1000$, $\alpha_b = 8$ and for ACKs $\alpha_g=10000$, $\alpha_b = 3$ where the values of α_g and α_b are in packets and are subject to the following relationship:

$$\alpha_b = \text{BER} * \text{packet size in bytes} * 8 * \alpha_g \quad (1)$$

For ACK parameters, the packet size is replaced by Ack size in relation (1) above. Bit errors cause packets to get corrupted, resulting in lost TCP packets and ACKs. Consequently, they have a negative impact on the performance of TCP. Decreasing BER by one order of magnitude significantly enhances the performance of TCP as illustrated in Table 1. It is noticed that it caused the throughput to double. Throughput is computed as the ratio between the acknowledgement packets received by the source and the total connection time. Higher BER causes the sender to have smaller windows resulting in low throughput. The goodput increased with lower BER. Goodput is determined as the ratio between the amount of useful data transmitted by the source and the total amount of data transmitted by the source. The useful data is determined by the difference between the total data packets sent and the packets resent by the source. The lower the BER, the less packets/ACKs dropped, the less they are retransmitted, resulting in better goodputs. We define “transfer time” as the time it takes the receiver to receive a fixed number of packets, say 5000 packets. We calculate the transfer time by finding the time the sender receives an ACK for packet 5000. With one order of magnitude less BER, TCP was able to send 5000 packets in almost half of the time.

	BER = 10^{-5}	BER = 10^{-6}
Throughput	39.439	87.455
Goodput	0.9892	0.999
Transfer time of 5000 pkts.	123.847	58.032

Table 1: Effect of BER on Performance of Tahoe TCP

We ran experiments on four different implementations of TCP: Tahoe, Reno, New-Reno and Sack. For each implementation we had two scenarios: scenario1 had a disconnection probability $p= 9\%$ (link-up period = 1.0s, link-down period = 0.1s), and scenario2 had a disconnection probability $p= 28.6\%$ (link-up period = 5.0s, link-down period = 2.0s), which is much larger than the first scenario.

The disconnection probability in scenario2 is more than three times as that of scenario1. Nevertheless, we found that the throughput of Scenario 1 is about 5 times better than that of Scenario 2. The transfer time of Scenario 1 was found to be about 8.4 times smaller than that of Scenario1. Thus the performance of the network does not only depend on the disconnection probability but more on the length of the mean of the link-up period. Scenario2 had a larger link-up mean, and that is why it resulted in a better performance, even though it had an unrealistically high disconnection probability. Figures 2 and 3 illustrate the performance of Reno TCP when $p=9\%$ and $p=28.6\%$, respectively. For the sake of illustration, we ran the experiment for 10 seconds using a simple distribution for link-up/down times as follows: the simulation starts when the link is up. It stays up for 1 second in case of Figure 2 and 5.0 sec in case of Figure 3, then goes down for 0.1 sec in case of Figure 2 and 2.0 sec in case of Figure 3. This cycles till the end of the simulation. In figure 2, we notice that the reason for the bad performance of TCP in that scenario is that the sender does not have enough time to send packets to MH before it disconnects. Although it disconnects for a very short time, by the time the sender starts sending again, the mobile disconnects again, and so on. The short connection time (1 second) and the frequent disconnections made FH unable to open up a large window and send enough packets to MH. On the other hand, in scenario2,

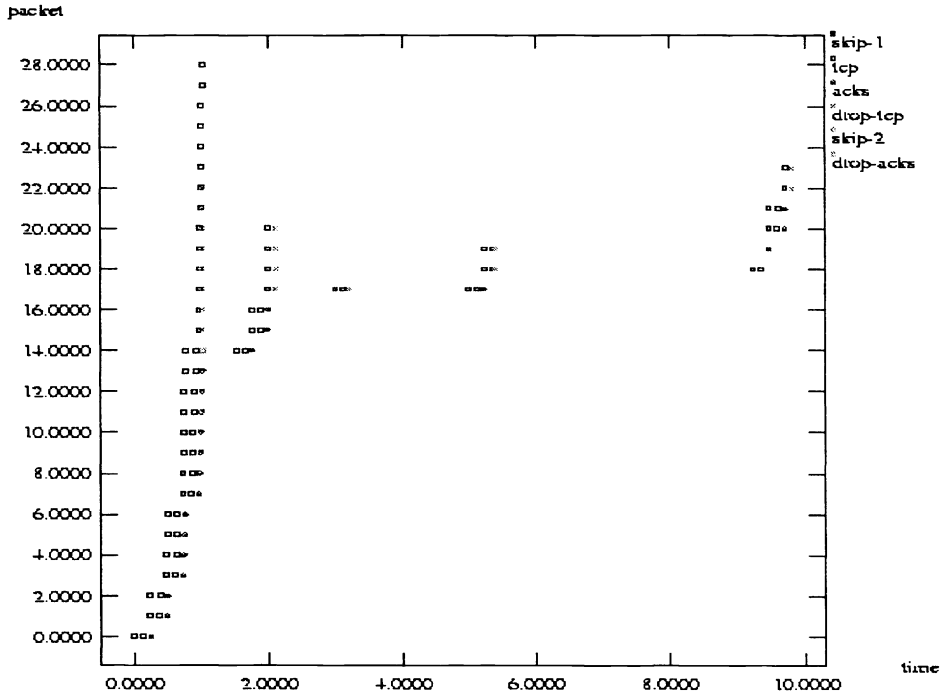


Figure 2: Reno TCP with p=9%

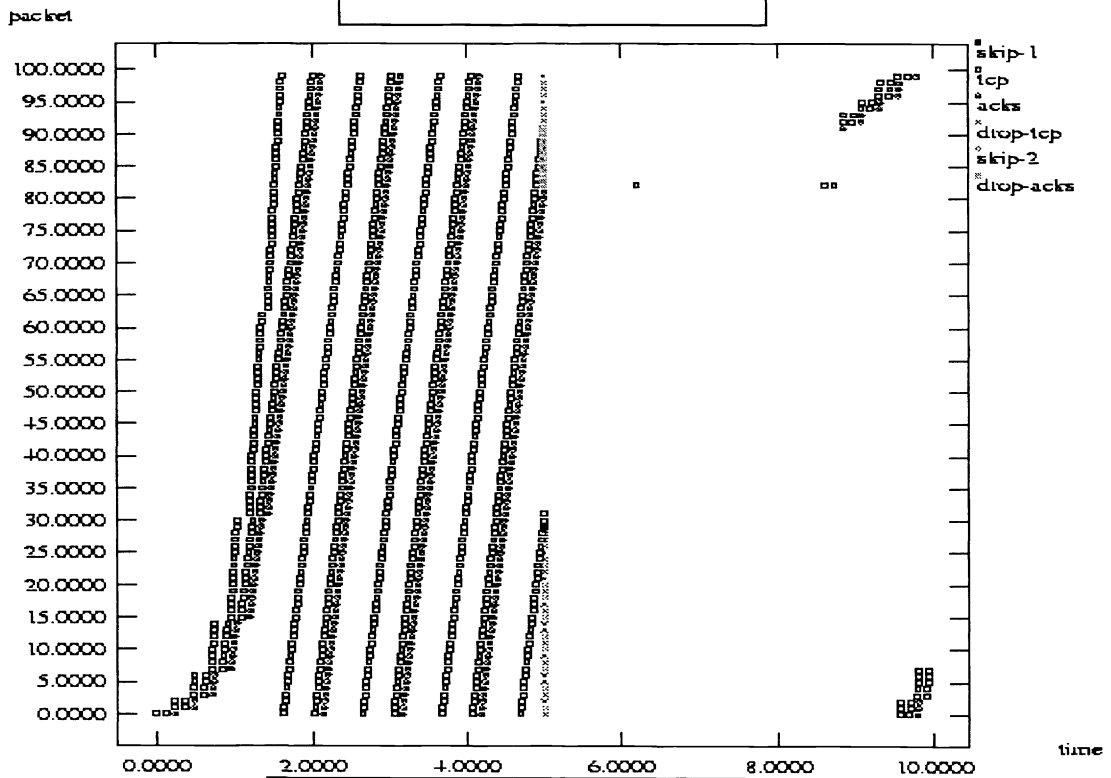


Figure 3: Reno TCP with p=28.6%

we notice from Figure 3, that there is a long connection time for FH to send packets before the mobile disconnects. Even though, the mobile disconnects for a long period (2 seconds), but the frequency of the disconnection in this scenario was much less than the first. To study the effect of the link up period, for a disconnection probability of 9%, we varied the mean of the link-up period and found that the larger the mean of the link-up period the better the throughput, the goodput and the transfer time. The performance of TCP with a disconnection probability for the mobile of 9% can be better than that with a disconnection probability of 28.6% only with a suitable mean of link-up period.

We repeated the experiments for Tahoe, New-Reno and SACK TCP and they provided similar results. Fig. 4 shows that for the same disconnection probability, the higher the mean of the linkup, the more the throughput of TCP Reno. The throughput of TCP with disconnection probability $p=28.6\%$ ($m1=5, m2=2$) is better than the throughput with $p=9\%$ and $m1$ is less than 3. Similar results were obtained for goodput. The higher $m1$ is, the higher the goodput of the network. We found that the goodput with $p=28.6\%$ ($m1=5, m2=2$) is better than the goodput with $p=9\%$ and $m1$ is less than 5. Also the higher $m1$, the less the transfer time. To transfer 5000 packets, the value of the transfer time decreases as $m1$ increases then becomes constant (52.29s) when $m1$ becomes greater or equal to 5. In this latter case, the TCP connection does not suffer any disconnection while sending the 5000 packets.

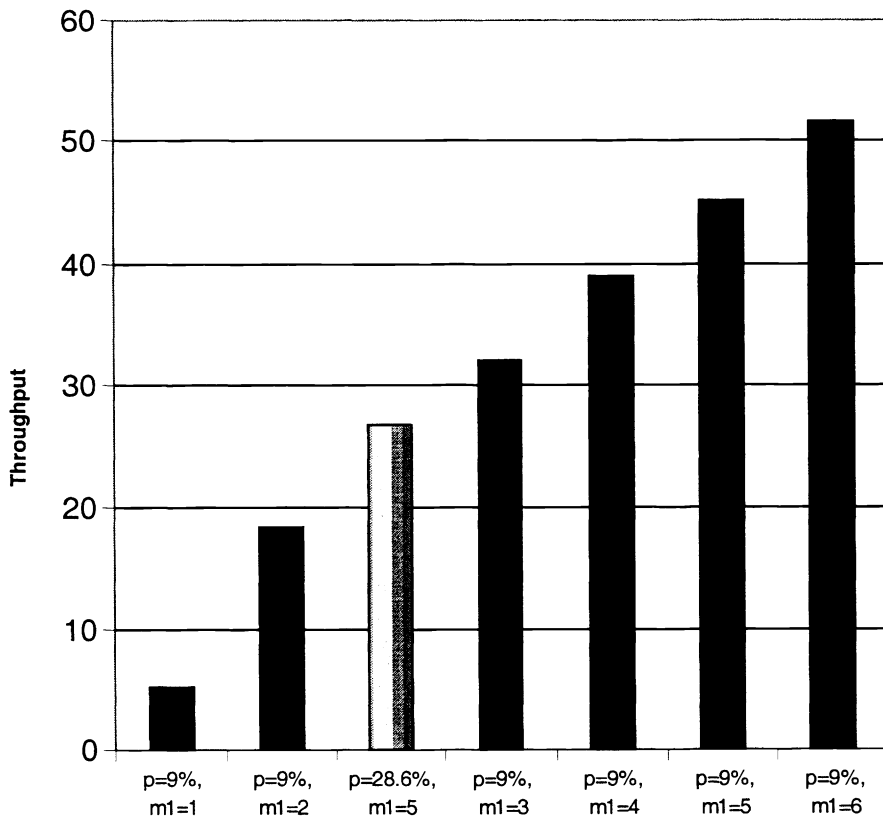


Figure 4. Throughput of Reno TCP at various $m1$ values

It is noticed from our simulations, that the disconnection of the mobile has a greater impact on the performance of TCP than BER. Disconnections, generally can be of the order of several seconds. They can even last for 1 minute. This result in the loss of great number of contiguous data packets and ACKs, causing frequent timeout of the TCP connection, resulting in a great reduction of its performance.

Combining this observation with the results of the effect of the link-up period, we can conclude that the link-up period has the dominant effect on the performance of TCP.

We next investigate the performance of the different TCP implementations in the case that the wireless network suffers only from mobile disconnections and no wireless link errors are incurred. Figures 5-8 show the behavior of Tahoe, Reno, New-Reno and Sack, respectively, for this case.

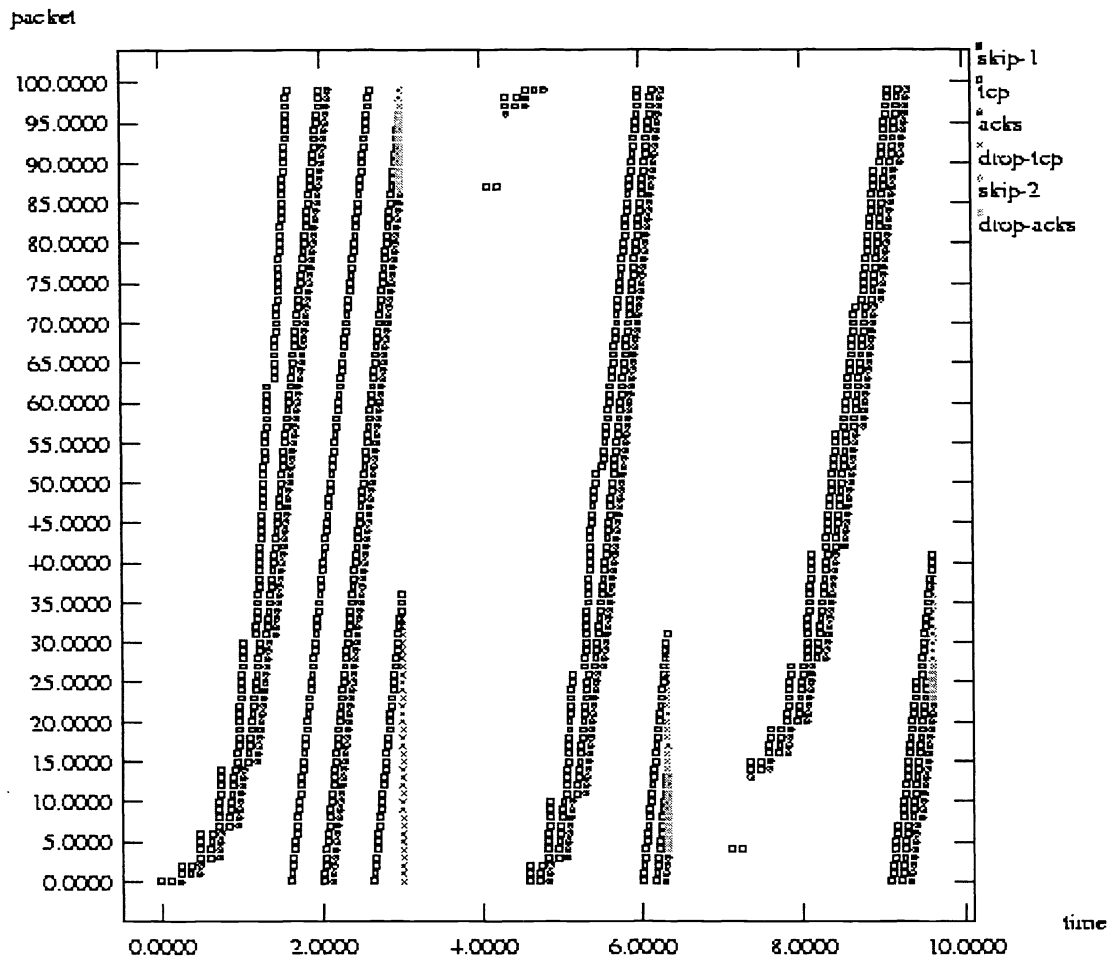


Figure 5. Tahoe TCP

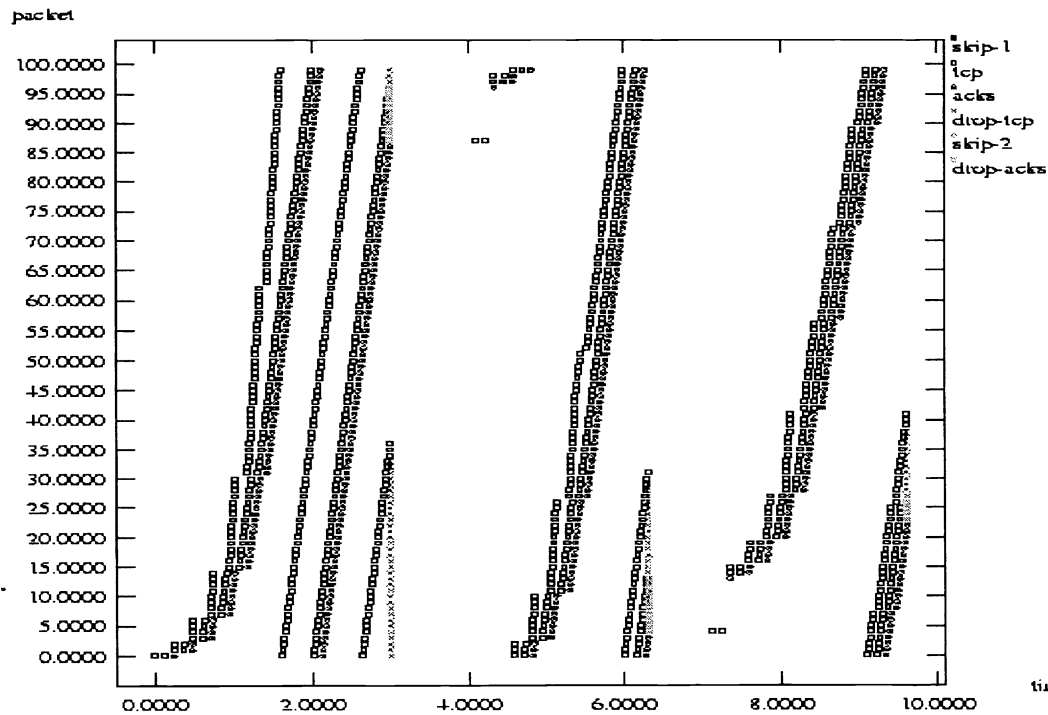


Figure 6. Reno TCP

The figures show a simulation of 10 sec for illustration. In Figure 5 for Tahoe TCP, the connection starts with the link up. The sender runs the slow start algorithm with ssthresh=50, which increases the window exponentially from 1-50 sending packets 0-112. Then it enters the congestion avoidance algorithm, which increases the window linearly, sending packets 113-235. At time 3.0 sec, the connection to the mobile is lost. This causes ACKs 187-196 and packets 197-233 to be lost. The mobile is connected at time 3.3, nevertheless, the sender will not send any packets till time 4.105 sec, when it timeouts. The sender then assumes that the packets are lost due to congestion. It reacts by setting ssthresh to the value

$$\max(\min(\text{cwnd}, \text{window}), 2)$$

then enters slow start, shrinking its cwnd to 1. It resends packet 187. Although, the receiver successfully received packets 187-196, the sender had no way to know about it because the corresponding ACKs thereceiver sent were lost. The receiver then acknowledges packet 196 (last one it received) causing the sender to increase its window to 2 and sends packets 197, 198. The sender continues to increase its window size exponentially until it reaches ssthresh (=25 now), then linearly until the mobile disconnects again at time 6.3 causing ACKs 304-313 and packets 314-329 to be dropped. And the cycle continues.

Figures 6, 7 and 8 show the behavior of Reno, New-Reno and Sack respectively. Like Tahoe TCP, they start in slow start until ssthresh is reached then they enter congestion avoidance. When the mobile disconnects, and a number of contiguous packets and ACKs are lost, they had to wait till they timeout because there were no duplicate ACKs received. As a result, each did not execute its special fast recovery algorithm. Consequently, they all had to recover by dropping their ssthresh to half current window size, their cwnd to 1, and entering slow start causing their performance to be similar to Tahoe TCP.

Fall and Floyd¹² compared the performance of Tahoe, Reno, New-Reno and Sack in a wired environment. They made simulations varying the number of packets dropped in one window from 1 to 4. They showed that Reno TCP behaves real badly when more than one packet is dropped in one window, while SACK performs best. In the case of a wireless network, we showed that when the mobile disconnects, most probably there are multiple packets (and ACKS) dropped in the same window. While, the number of packets dropped from the same window will vary depending on the window size and the length of the disconnection, most probably those packets are found to be contiguous. Frequent disconnections will cause serial timeouts that are very harmful to the performance of TCP irrespective of the implementation of the protocol. Mobile disconnections caused the performance of Reno, New-Reno, Sack to drop to the performance of Tahoe TCP. Hence, it causes the great deterioration of the performance of TCP irrespective of its implementation (all implementations have the same behavior).

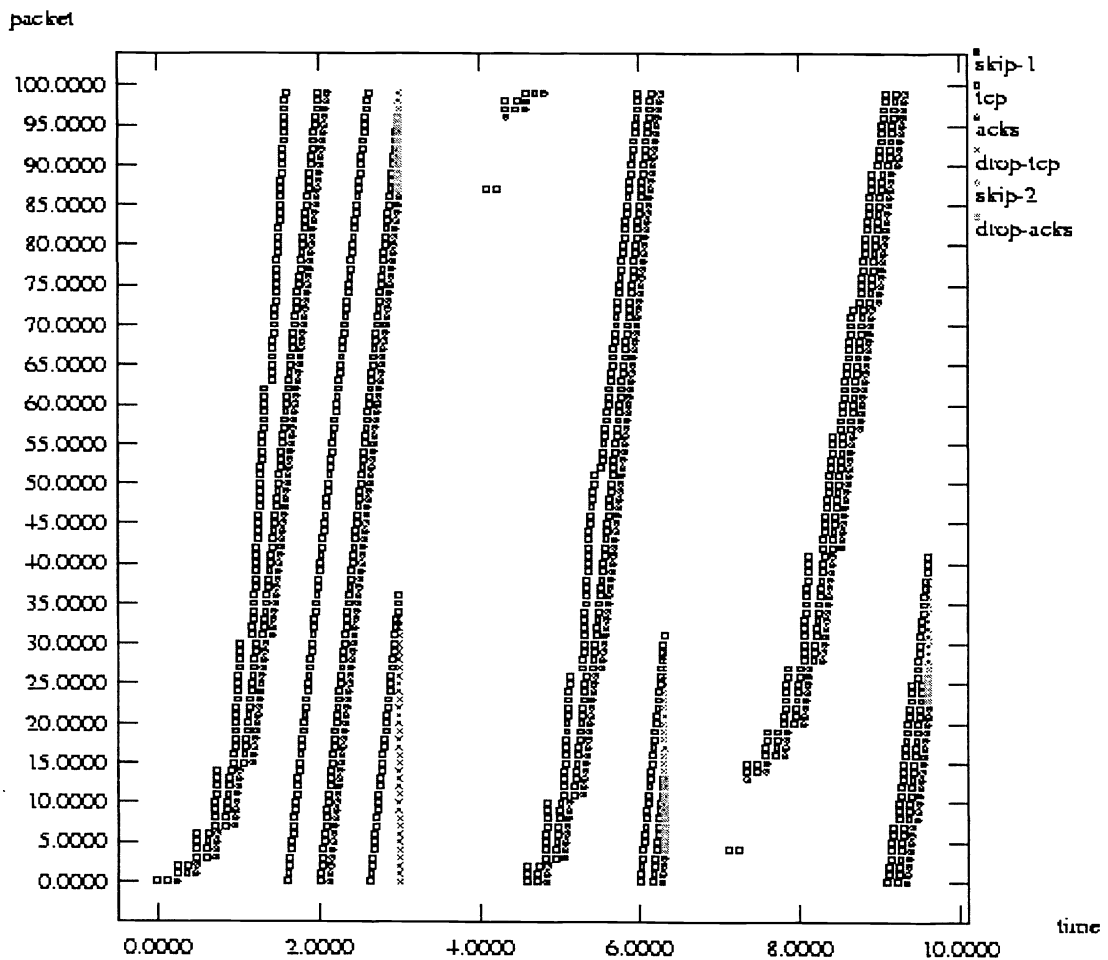


Figure 7. New-Reno TCP

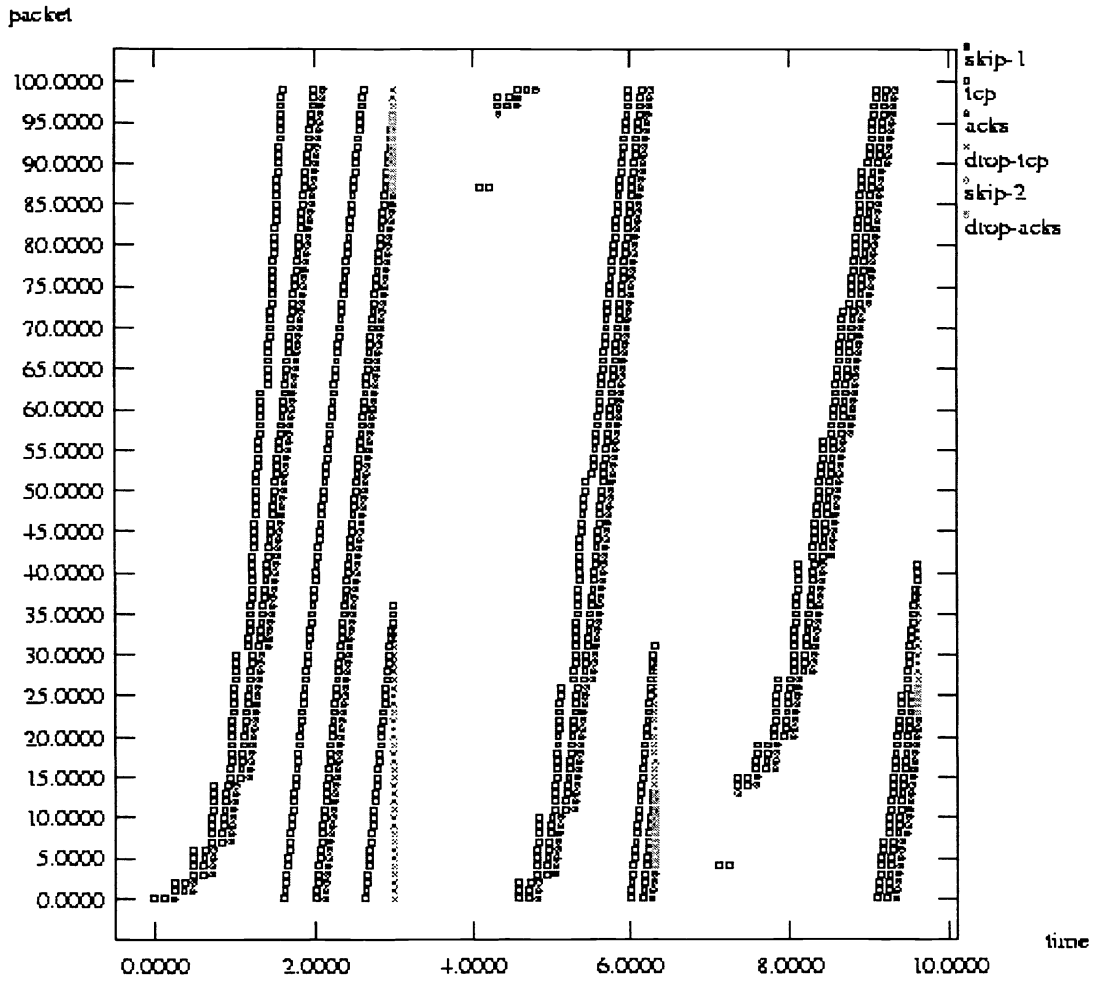


Figure 8. Sack TCP

In conclusion of this section, we compare the performance of the four TCP implementations in a wireless mobile network, considering both mobile disconnections and bit error rates. The wireless link has BER of 10^{-5} . Thus, for TCP packets (BS to MH) $\alpha_g=100$, $\alpha_b=8$ while for ACKs (MH to BS) $\alpha_g=1000$, $\alpha_b=3$. For the mobile disconnectivity $m1=2.0$ and $m2=0.3$. Performance results for a runtime of 1600 seconds are presented in Table 2.

	Tahoe	Reno	New-Reno	Sack
Throughput (pkt/sec)	9.47	11.990625	10.69	10.523125
Goodput	0.91197347	0.91480524	0.9114321	0.916
Transfer time for 9210 packets in sec.	880.888	832.432	915.391	965.035
Average packet delay in second	0.426973	0.407861	0.424393	0.464114

Table 2. Performance Measures when BER= 10^{-5} , $m1=2.0$ and $m2=0.3$

Table 2 shows that Reno TCP has the best overall performance over the other three implementations. It has the most throughput, least transfer time and least average packet delay. It also has a very comparable goodput to Sack TCP.

5 CONCLUSION

In this paper, we studied the effect of wireless link bit error rates, disconnection probability and link-up of the mobile on the performance of four TCP implementations, namely Tahoe, Reno, New-Reno and Sack. To concentrate on the mobility and reliability aspects of the wireless connection, our simulation tests used sufficiently large buffer sizes in the fixed host and the base station of the TCP connection. The paper showed by varying the link-up and the link-down periods, it is possible to obtain better throughput at higher disconnection probability. For example, the throughput of TCP Reno with disconnection probability of 28.6% and a link-up period of 5 was shown to be better than the throughput with disconnection probability of 9% and a link-up period of less than 3. The paper presented timing graphs tracing the movement of packets and acknowledgments between the fixed and mobile hosts. Dropped packets or acknowledgments shown in these graphs were the result of mobile disconnection or wireless bit errors and not because of buffer congestion. Our tests used two scenarios; the first scenario compared Tahoe, Reno, New-Reno and Sack in the case where the wireless network suffered only mobile disconnections and no bit errors. In this case, mobile disconnections caused multiple contiguous packets and Acks to be lost. We showed that all four implementations behaved the same similar to Tahoe TCP. Reno, New-Reno and Sack did not get to execute their special fast recovery algorithms as they always detected packet loss by timing out. One improvement here is to have Reno, New-Reno and Sack go through fast recovery instead of slow start after a timeout. The other scenario, we simulated a wireless network suffering both bit errors and mobile disconnections. We showed that Reno TCP had the best overall performance over the other three implementations. This was unlike wired networks where Sack always performed better than Reno TCP.

ACKNOWLEDGMENT

This work has been supported by ARO under grant no. DAAH04-95-1-0250 and by an I-4 corridor grant from the State of Florida and Harris Corporation. 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

1. Stevens W.R., *TCP/IP Illustrated*, vol. 1, Addison -Wesley, 1994.
2. Comer D., *Internetworking With TCP/IP*, Vol.1, 2 &3, Prentice Hall, 1991.
3. Bakshi B. et al., "Improving Performance of TCP over Wireless Networks", 17th International Conference on Distributed Computing Systems", pp. 365-373, 1997.
4. Brown, K., and Singh S., " M-TCP: TCP for Mobile Cellular Networks", ACM SIGCOMM Computer Communication Review, pp. 19-43, July 1997.
5. Eckhardt D., and Steenkiste P., "Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network", Proceedings ACM SIGCOMM, pp.243-254, October 1996.
6. Ayanoglu E., et al., "AIRMAIL: A Link-Layer Protocol for Wireless networks", Journal of Wireless Networks, Vol. 1, pp. 47-60, 1995.
7. Jacobson V. "Congestion Avoidance and Control" SIGCOMM Symposium on Communications Architectures and Protocols, pp. 314-329, 1998. <ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z> for an updated version.

8. Jacobson V. " Modified TCP Congestion Avoidance Algorithm" Technical Report 30, April 1990. Available at <ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt>
9. Stevens W.R. " TCP Slow Start, Congestion Avoidance, Fast Retransmission, and Fast Recovery Algorithms", IETF, RFC 2001, January 1997.
10. Hoe J. C., " Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *ACM SIGCOMM* 1996, pp.270-280.
11. Mathis M., Mahdavi J., Floyd S., and Romanow A., " TCP Selective Acknowledgemnet Options", IETF, RFC 2018, October 1996.
12. Fall K. and Floyd S., " Simulation based Comparisons of Tahoe, Reno, and SACK TCP", *ACM Computer Communication Review*, 26(3), pp.5-21, 1996.