Modified Tahoe TCP for Wireless Networks Using OPNET Simulator

M. N. Akhtar, M. A. O. Barry and H. S. Al-Raweshidy

Department of Electronic Engineering, University of Kent, U.K.

Email: {mna5, mb43, h.al-raweshidy}@kent.ac.uk

ABSTRACT

This paper describes a modification to the Transmission Control Protocol's (TCP) Tahoe flavour for use in Wireless networks. It shows that by slightly modifying the algorithm of the Tahoe TCP, it can be made to respond better to wireless links, while maintaining its advantages on the wired networks at the same time. This is certainly a very desirable feature as the conventional TCP in most cases contradicts to the demands of the wireless links of the network.

1. INTRODUCTION

In theory, transport protocols should be independent of the technology of the underlying network layer. It should not care whether IP is running over fibre or radio. But in reality, it does matter, since most TCP implementations, that have been designed for wired networks, don't cope very well on wireless networks.

TCP was designed with the notion in mind that wired networks are generally reliable and any segment loss in a transmission is due to congestion in the network rather than an unreliable medium. This notion doesn't hold in wireless parts of the network. Wireless links are highly unreliable and they lose segments all the time due to a number of factors. These include fading, interference, higher bit error rate and mobility related processes such as handovers.

In this paper, we will first make a comparison between Reno, Tahoe and SACK (Selective Acknowledgements) TCP flavours in a wireless environment. Our experiments show that Tahoe copes well in a high segment loss environment. So we will then suggest some modifications in the congestion control mechanism of the Tahoe flavour of TCP and compare its results with the unmodified version.

Two algorithms, Slow Start and Congestion Avoidance, modify the performance of TCP's sliding window to solve some problems relating to congestion in the network. The idea of congestion control is for the sender to determine the capacity that is available on the network. The sender, in standard TCP implementation, keeps two state variables for congestion control: a slow-start/congestion window, *cwnd*, and a threshold size, *ssthresh*. These variables are used to switch between the Slow Start and Congestion Avoidance algorithms.

Slow Start provides a way to control initial data flow at the beginning of a communication session and during an error recovery. This is based on received acknowledgements.

Congestion Avoidance increases the *cwnd* additively, so that it grows by one segment for each round trip time. These two algorithms are implemented together and work as if they were one. The combined effect of these two algorithms on the *congestion window* is shown in Figure 1.

After this brief overview the rest of the paper is arranged as follows: Section 2 describes the simulation settings used for the experiments. Section 3 shows the simulations done with three different flavours of TCP i.e., Reno, Tahoe and SACK. In 4, we discuss the relative performance of these three types and 5 shows the modifications done on Tahoe along with the results and recommendations for future study.

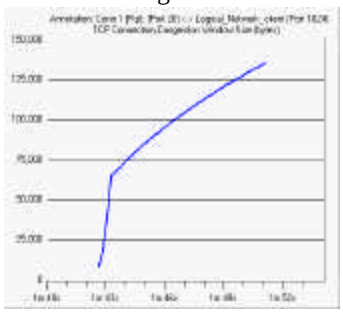

Figure 1: Congestion Window with Slow Start and Congestion Avoidance

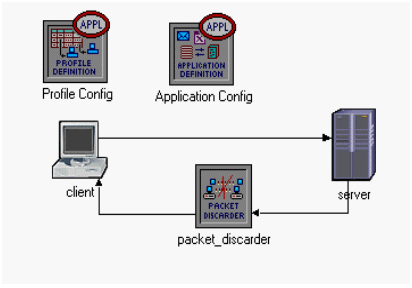

Figure 2: Simulation Scenario

2. SIMULATION SETTINGS

Simulations were run with OPNET Modeler [1]. The simulated scenario is shown in Figure 2. It has two stations: a client and a server connected by a 1.5 Mbps line. In between the two stations is a Packet Discarder which discards packets going through it. The value of the number of packets to be discarded is set at the start of the simulation. We have set the packet discarder to drop 4 packets in a 0.5 sec period, assuming that a handover, on an average, lasts for 0.5 sec and there are 4 packet drops in a handover. We use the existing TCP model provided in OPNET which has two process models: *tcp_manager_v3* and its child process *tcp_conn_v3*. The scenario is set to simulate a 1.6 MB file transfer from server to client using FTP. The main statistics gathered were the TCP *congestion window size* for the server and the *FTP download response time* for the client.

3. NETWORK SIMULATIONS

3.1 Overview

Slow Start and Congestion avoidance algorithms provide features like flow control in error recovery situations and adaptation to network conditions. However, when an error occurs, these algorithms respond relatively slowly in recovering the network back to its original throughput. Fast Retransmit and Fast Recovery are two other algorithms which help the network get back to its original throughput. These two algorithms are briefly explained as follows.

3.1.1 Fast Retransmit

In case of a segment loss, an immediate acknowledgement is sent. This is also called a duplicate acknowledgement. Fast Retransmit states that if 3 acknowledgments are received before the retransmission timeout, the sender then has to retransmit the lost segment immediately. From this point in time until an acknowledgement for new data arrives, every duplicate acknowledgement triggers a new data transmission. Thereafter, the *cwnd* is set back to its initial value and *Slow Start* is initiated.

3.1.2 Fast Recovery

This algorithm is based on the notion that if duplicate acknowledgements are being received, then the network is not fully congested. Thus there is no need to abruptly reduce the flow and initiating a *Slow Start* [2]. Hence, on receiving 3 duplicate acknowledgements, the *cwnd* is set to half of its previous size and *Congestion Avoidance*, instead of *Slow Start*, is performed. The combinations of Fast Retransmit and Fast Recovery give rise to the different flavours of TCP namely Reno, Tahoe and SACK.

3.2 Reno TCP

The flavour that uses both Fast Retransmit and Fast Recovery together is called Reno TCP [3]. Reno performs well until only one segment is lost, but in a wireless scenario, there will be multiple segment drops in a handover and also in the unreliable media. In case of multiple segment losses, the *cwnd* closes to half of its previous size for each segment drop. Furthermore, the recovery is further delayed because only the Congestion Avoidance algorithm is functioning without the added benefit of Slow Start. The comparison of Reno TCP with 1 and 4 segment drops is shown is Figure 3. We can observe that Reno takes a considerably more time to recover when more than one segment is dropped.

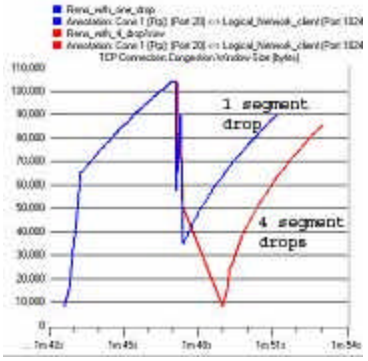

Figure 3: Reno with one and four drops

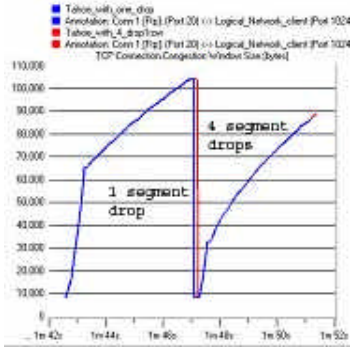

Figure 4: Tahoe with one and four drops

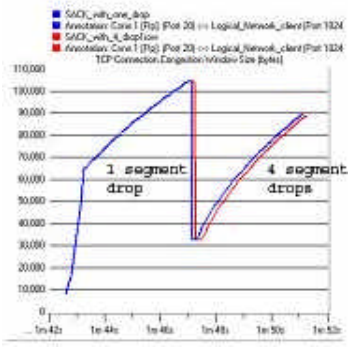

Figure 5: SACK with one and four drops

3.3 Tahoe TCP

Fast Recovery algorithm does not allow the *cwnd* to open exponentially. Instead, it applies the much slower Congestion Avoidance mechanism [3].Due to this; problems arise in Reno TCP when multiple segments are lost.

To avoid this problem, Tahoe TCP does not apply Fast Recovery. When this flavour is used, Fast Retransmit is the only algorithm applied. As illustrated in Figure 4, there is very little difference between 1 and 4 segment drops in the Tahoe TCP environment.

3.4 SACK TCP

As with Reno, Selective ACK (SACK) TCP also uses Fast Retransmit and Fast Recovery. It also suffers from the same problems with multiple losses. However, the advantage of SACK is that the receiver informs the transmitter of the segments it successfully received. This is done so that retransmission only takes place for those segments that have not been acknowledged. Thus the number of retransmitted segments in the network is significantly reduced [3]. As shown in Figure 5, the difference between one and four dropped segments in case of SACK TCP is not as significant, as in Reno TCP. This is because of the selective acknowledgements being sent. The graphs of SACK and Tahoe in Figure 7 do not differ very much at this stage, but the difference becomes more pronounced when there are more segments dropped as would be the case in wireless links.

4. COMPARISON OF THE THREE FLAVOURS

From Figure 6 it can be seen that SACK TCP performs better than Reno and Tahoe when there is only one segment dropped. Reno and SACK outperform Tahoe because Tahoe goes into Slow Start after retransmitting the lost segments and in the process, the *cwnd* closes all the way to its initial value. In case of Reno and SACK, the *cwnd* is reduced to half of the value it had before the error.

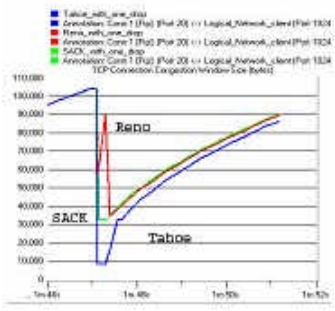

Figure 6: Congestion Window comparison with 1 segment drop: Reno, Tahoe, SACK

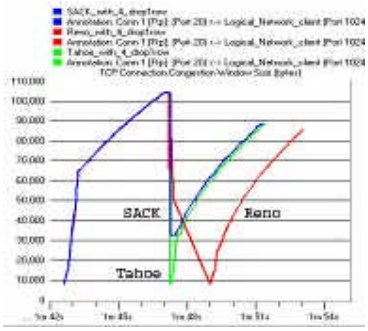

Figure 7: Congestion Window comparison with 4 segment drops: Reno, Tahoe, SACK

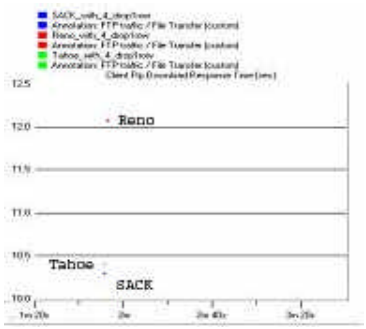

Figure 8: Comparison of client FTP download response time with 4 drops

Figures 7 and 8 show the comparison with 4 segment drops. It is clear that the performance of Tahoe is now quite similar to that of SACK. The Tahoe sender recovers with a Fast Retransmit followed by Slow-Start regardless of the number of segments dropped [4]. The Reno implementation without SACK gives optimal performance when a single segment is dropped. With two segment drops we found that the sender goes through Fast Retransmit and Fast Recovery two times in succession, which unnecessarily reduces the *cwnd* twice. As seen in Figure 7, for the scenario with four segment drops, the Reno sender has to wait for a retransmit timer to recover.

As expected, the SACK TCP recovers from the four segment drop scenario without having to wait for a retransmit timeout. As the errors keep on increasing as in the case of wireless links, the performance of SACK deteriorates as compared to Tahoe. Simulations were run with 10 segment drops and it was found that Tahoe gave the best performance. Since our scenario includes wireless links, there will be lots of segment drops due to handovers as well as unreliable media, therefore from the comparison it has been deduced now that Tahoe will be the best option.

5. MODIFIED TAHOE TCP

5.1 Concept

As wireless transmission links are very unreliable and are prone to higher segment loss mostly unrelated to congestion. Therefore, resending lost segments without delay will facilitate faster recovery without unnecessarily waiting for congestion avoidance procedures, [5]. For example, if 20% of all segments are lost, then if the sender transmits 100 segments/sec, the throughput will be actually 80 segments/sec. Now if the sender slows down to 50 segments/sec, the throughput will become 40 segments/sec.

This theory is contrary to the mechanism used in wired networks where on losing a segment (mostly due to congestion), the sender has to slow down. In wireless, on losing a segment, the sender has resend quickly. It should be borne in mind that TCP is an end to end protocol and a connection will almost always be mixed (part

wired and part wireless). So, a balance has to be kept while making any modifications to suit the wireless links. This balance is between segments dropped due to conventional congestion as opposed to wireless related reasons like high bit error rate and mobility.

5.2 Modifications

Keeping the above considerations in mind, we have proposed some modifications to the Tahoe TCP model. The reason for choosing Tahoe is so that we can take advantage of the exponential window opening algorithm (Fast Retransmit) as well as eliminating Fast Recovery which slows down the *cwnd* recovery. The proposed modifications are:

a. Reducing the number of duplicate acknowledgements required from 3 to 2 before a Fast Retransmit takes place. This supports the theory of resending a lost segment without delay.

b. Raising the Slow Start initialisation point from 1 MSS to 3 MSS. This will prevent the *cwnd* from closing all the way and will aid a fast recovery of the *cwnd*.

c. Raising the Slow Start threshold from half the previous *cwnd* size to ¾ of the previous *cwnd*. This is done so that the process of slow start continues for a longer period and the *cwnd* gets back to its original size quickly.

5.3 Results

With the above mentioned changes applied, simulations were run and the results recorded and compared. As shown from Figures 9 and 10, the modified Tahoe TCP gives a reduced *congestion window recovery time* and a reduced *client FTP download response time*. Figure 9 also shows that the *cwnd* not only drops to a higher value than in the unmodified case but also continues the Slow Start process for a longer time. This has the dual effect of keeping a larger window open and also continuing to open it at a faster rate for a longer time. Due to this, the download response time has also decreased (Figure 10).

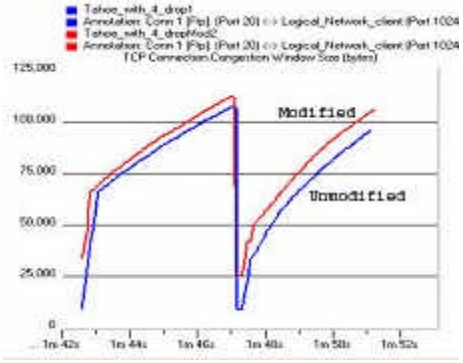

Figure 9: Congestion window comparison of modified and unmodified Tahoe TCP

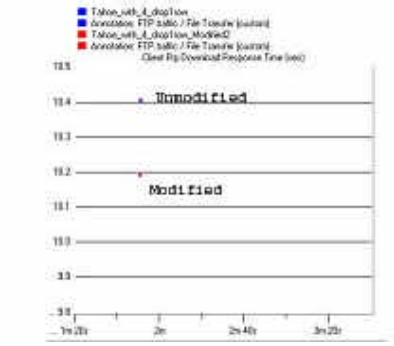

Figure 10: Comparison of client FTP download response time: modified and unmodified Tahoe

6. CONCLUSION

It can be observed from the discussed 3 modifications that the whole idea behind this exercise was to bring back the transmission rate to its original throughput after an error occurs and the TCP flow control mechanism is applied. For example in modification (a), we could have reduced the number of duplicate acknowledgements required to 1 instead of 2. But doing this would have rendered TCP useless in case of a congestion scenario on wired networks. Similarly in modifications (b) and (c), any more changes that are biased towards wireless links would have deteriorated the TCP performance on wired networks. For this reason, a balance between the two links environments was maintained, so that TCP is workable under mixed network.

In this paper a comparison of different TCP flavours was made by running simulations and collecting results. These showed that Tahoe works best under a wireless scenario. Then three modifications were made to Tahoe TCP and it has been shown that the modified version performed better in wireless links.

7. REFERENCES

[1] http://www.opnet.com

[2] "TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms". Network Working Group, RFC: 2001, W. Stevens, January 1997.

[3] "Simulation-based study of TCP flow control mechanisms using OPNET Modeler". G. Corral, A. Zaballos.

[4] "Simulation Based Comparisons of Tahoe, Reno and SACK TCP". K. Fall, S. Floyd, 1995.

[5] "Computer Networks", The Transport Layer, Andrew S. Tanenbaum, 3rd Edition.