PERFORMANCE COMPARISON OF TCP MECHANISMS OVER SATELLITE NETWORKS

Hashim Zafar[1], Raed A Abd-Alhameed[2], and Haseeb Zafar[3]

[1]Delivery & Support, Optimi MENA, Dubai, UAE
hashim.zafar@optimi.com

[2]School of Engineering, Design and Technology
University of Bradford, Bradford, UK
r.a.a.abd@bradford.ac.uk

[3]Department of Electronic & Electrical Engineering
University of Strathclyde, Glasgow, UK
haseeb@eee.strath.ac.uk

ABSTRACT

The performance of TCP over satellite links has been suboptimal due to a variety of protocol algorithm and configuration issues. Connection speed over satellite networks is rather slow and, in order to preserve good link quality, packet loss is interpreted as an indication of network congestion. Appropriate back-off mechanisms are therefore, implemented to prevent any further overload. However, these back-off strategies are not helpful when packet losses are due to transmission errors, particularly if these are bursty, which reveals the need for TCP/IP enhancements when used over satellite links. High efficiency on fast satellite links may eventually require new protocol modifications, but full link utilization may be achievable using the TCP performance enhancements that have already been approved or that are currently in the standards process. This paper describes different types of TCP standards and compares their performance using the OPNET simulator to show how improvements may be achieved.

Index Terms—TCP (Transmission Control Protocol), IP (Internet Protocol), satellite networks, OPNET

1. INTRODUCTION

Transmission Control Protocol (TCP) is a transport layer protocol which controls the transmission of data between two end nodes over an IP network. TCP has two primary functions: reliable delivery of data from sender to receiver, and active management of the transmission rate to avoid overly congesting the network. There are several schemes proposed to improve the performance of TCP over satellite links [1][2].

TCP is the most commonly and widely used transport protocol in the Internet, due to simplicity of installation, remote accessibility, low cost of operation/maintenance and availability of existing setup infrastructure. The provision of TCP/IP services over satellites is useful for areas or countries that have little terrestrial infrastructure. This contrasts with the situation in real-time transfer running over UDP, with voice over IP as a particular example. With no feedback mechanism, UDP itself has no adverse reactions to satellite conditions and works just as well over satellite as any other type of link. Unfortunately, unlike the limitations of TCP over satellite links, which can be rectified by using protocols better optimized for satellite link conditions, the problems UDP experiences over satellite paths are inherent to the speed of light. Since there are no ways of making light go faster, the only realistic solution is to bring the satellite closer to the ground to reduce the latency.

The rest of the paper is organized as follows. Section-2 presents comparison of TCP standards. The simulation framework based on OPNET[9] is presented in Section-3. Performance comparisons are presented in Section-4. Concluding remarks are made in Section-5.

2. COMPARISON OF TCP STANDARDS AND FLAVOURS

The original design of TCP was intended to support rapid growth and diversity on the Internet. However, it had some “flaws” in its congestion control engine and congestion fall-down was experienced in 1986 [3]. An investigation resulted in the design of new congestion control algorithms, now an essential part of TCP. Since the development of the basic TCP congestion control algorithm, known as TCP Tahoe [3][4] in 1988, a number of variations of congestion control have been studied and proposed. These variations modify the slow start phase, the congestion avoidance phase, or the response phase. Different types of TCP standards are described as follows:

2.1. Duplicate Acknowledgements & Fast Retransmit

A TCP sender detects loss when it times out waiting for an acknowledgement (ACK). In addition to this, duplicate ACKs (*dupacks*) can be used to detect losses. If a TCP receiver receives an out of order segment, it straight away sends back a *dupack* to the sender. The *dupac*k indicates the byte number expected. The fast retransmit algorithm uses these *dupacks* to make retransmission decisions. If the sender receives n *dupacks* (n=3 is typically chosen to prevent spurious retransmissions caused by out of order delivery), it assumes loss and retransmits the lost segment without waiting for the retransmit timer to go off. TCP then reduces the slow start threshold (*ssthresh*) to half the congestion window size (*cnwd*), and resets *cwnd* to one segment. TCP Tahoe included fast retransmit in addition to slow start and congestion avoidance.

2.2. Fast Recovery and TCP Reno

TCP Reno retained all the enhancements in TCP Tahoe but included a new algorithm, the fast recovery algorithm. Fast recovery is based on the fact that a *dupack* indicates that a segment has gone from the network. Hence, when the sender receives three *dupacks*, it retransmits the lost segment, updates *ssthresh*, and reduces *cwnd* as in fast retransmit. Fast recovery, however, keeps track of the number of *dupacks* received and tries to approximate the amount of outstanding data in the network. It inflates *cwnd* (by one segment) for each *dupack* acknowledged, thus maintaining the flow of traffic. Thus, when fast recovery receives an ACK for the segment whose loss resulted in the duplicate ACKs, TCP then deflates the window by returning it to *ssthresh* and enters the congestion avoidance phase.

If multiple segments are lost in the same window of data, on most occasions, TCP Reno waits for a retransmission timeout, retransmits the segment, and goes into slow start mode. This happens when, for each segment loss, Reno enters fast recovery, reduces its *cwnd* and aborts fast recovery on receipt of a partial ACK. A partial ACK is one that acknowledges some but not all, of the outstanding segments. After multiple such reductions, cwnd becomes so small that there are not enough dupacks for fast recovery to occur, and a timeout is the only option left.

2.3. TCP NewReno

A timeout affects the throughput of a connection in two ways. First, the connection has to wait for a timeout to occur and cannot send data during that time. Second, after the retransmission timeout occurs, *cwnd* is reduced to one segment. These events negatively affect the performance of the connection.

In Reno, partial ACKs bring the sender out of fast recovery, resulting in a timeout in the case of multiple segment losses. In NewReno, when a sender receives a partial ACK, it does not come out of fast recovery [5][6][7]. Instead, it assumes that the segment immediately after the most recently acknowledged segment has been lost, and hence the lost segment is retransmitted. Thus, in a multiple segment loss situation, NewReno does not wait for a retransmission timeout but continues to retransmit a lost segment every time it receives a partial ACK. Thus, fast recovery in NewReno starts when three duplicate ACKs are received and ends when either a retransmission timeout occurs or an ACK arrives that acknowledges all of the data up to and including the data that were outstanding when the fast recovery process began. Partial ACKs deflate the congestion window by the amount of new data acknowledgements and then add one segment and re-enter fast recovery.

Hoe [7] also suggests two additional algorithms as part of the original NewReno proposal. The first estimates the initial *ssthresh* by using the delay bandwidth product of the TCP connection (which estimates the number of segments that can be in flight). The second sends a new packet for every two duplicate ACKs received during fast recovery. These algorithms are still under investigation and are not part of NewReno as described in RFC 2582 [6].

2.4. TCP with Selective Acknowledgments

Another way to deal with multiple segment losses is to notify the sender which segments have arrived at the receiver. Selective acknowledgment (SACK) is a version of TCP that adopts this approach. The receiver uses each TCP SACK block to indicate to the sender one contiguous block of data that has been received out of order at the receiver. When SACK blocks are received by the sender, they are used to maintain an image of the receiver queues, that is, which segments are missing and which have arrived at the receiver. Using this information, the sender retransmits only those segments that are missing, without waiting for a retransmission timeout. Only when no segments need to be retransmitted are new data segments sent out [8].

The SACK implementation can still utilize the same congestion control algorithms as Reno (or NewReno). It resorts to the retransmission timeout mechanism to deliver a missing segment to the receiver if ACKs are still not received in time. The main difference between SACK and Reno is the behavior in the occurrence of multiple segment losses. In SACK, just like Reno, when the sender receives three *dupacks*, it goes into fast recovery. The sender retransmits the segment and halves *cwnd.* SACK maintains a variable called ‘pipe’ to indicate the number of outstanding segments that are in transit. In SACK, during fast recovery, the sender sends data, new or retransmitted, only when the value of pipe is less than cwnd, that is, the number of segments in transit are less than the congestion window value. The value of pipe is incremented by one when the sender sends a segment and is decremented by one when the sender receives a duplicate ACK with SACK showing new data have been received. The sender decrements pipe by two for partial ACKs [5]. As with NewReno, fast recovery is terminated when an ACK arrives that acknowledges all the data up to and including the data that were outstanding when the fast recovery procedure began.

3. SIMULATION FRAMEWORK

OPNET Modeler [9], developed by OPNET Technologies Inc., is a very popular network simulation tool used by many researchers and practitioners for TCP/IP network simulation. OPNET is also an object-oriented simulation tool and is a totally menu-driven package with many user friendly Graphical User Interfaces (GUIs) for model construction, data collection, and other simulation tasks.

The model shown in Figure 1 illustrates satellite communication between several file transfer protocol (FTP) clients and a server within three subnets connected through satellites. In the model, Belfast is connected with Bradford via a Geostationary Earth Orbit (GEO) satellite whereas London is connected with Bradford via a Low Earth Orbit (LEO) satellite. Wired links are used between the appropriate satellite and each subnet because it is easy to put delay on each link, and it is the effect of the latency of the link that is of most interest to the research.

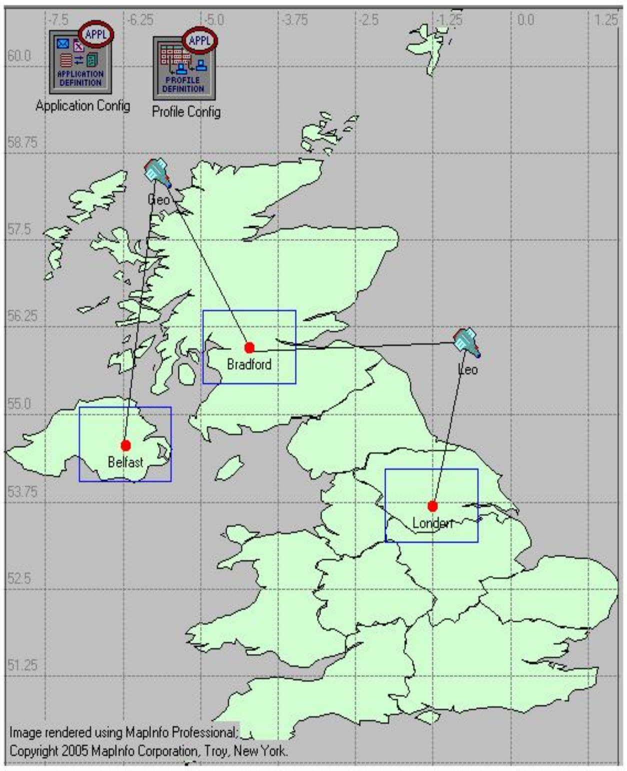

Figure 1 Network model

In this model, the delay on the Belfast-GEO-Bradford link is set to 250ms (typical GEO single trip delay), whereas on the Bradford-LEO-London link the delay is set to 10ms (approximation for LEO single trip delay). Also the packet discard ratio on GEO satellite is set to 0.5% while packet latency (delay on satellite) is 50ms. For the LEO satellite, the packet discard ratio is 0.05% and latency is 1ms. In wireless communications, due to nature of radio link many packets are lost, therefore making it necessary to set the discard ratio to a certain value.

4. PERFORMANCE COMPARISONS

4.1. Simulation Scenario

Four simulation scenarios (Tahoe, Reno, NewReno and SACK) have been used to evaluate the performance of TCP/IP over satellites. The objective of the simulations is to compare TCP Delay which is the overall delay experienced by a TCP packet, Segment Delay, Retransmission Count and Congestion Window Size. The application used in this connection is FTP. FTP performance is observed in the FTP server, in all simulation scenarios.

4.2. Simulation Results

Figure 2 shows the comparison of TCP Delay time for all the different scenarios. It is very obvious from the graph that Tahoe TCP has less delay time as compared to SACK, whereas Reno and NewReno present exactly the same time delay as Tahoe.

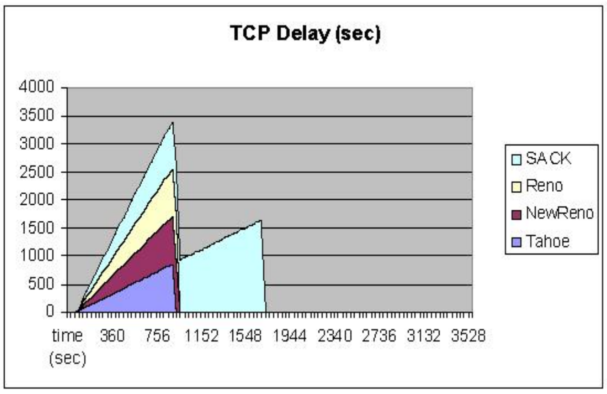

Figure 2 TCP Delay (sec)

Figure 3 shows a comparison of the congestion window sizes. The performance of Reno and NewReno is very close to Tahoe. As the errors keep on increasing, as in the case of wireless links, the performance of Reno and SACK deteriorate as compared to Tahoe. In simulations for multi segment drops, it was found that Tahoe gave the best performance. Since the scenario includes wireless links, there will be many segment drops due to unreliable media as well as handovers.

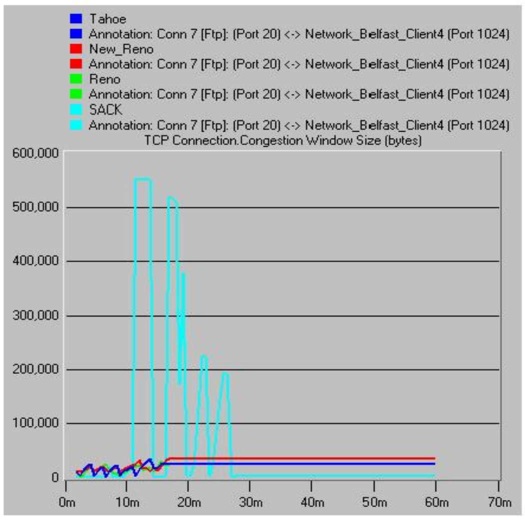

Figure 3 TCP Connection Congestion Window Size (Server of Network Bradford and Client4: Network Belfast)

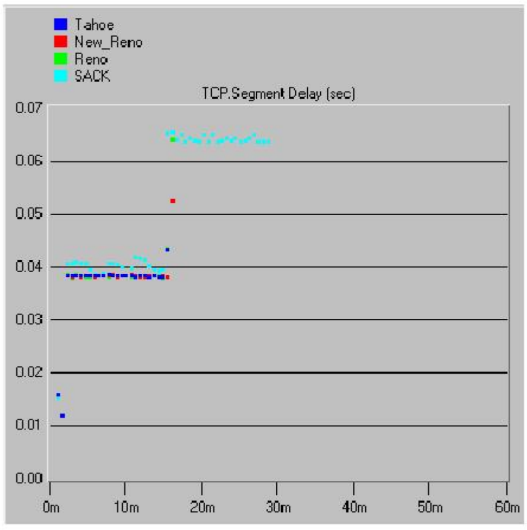

Figure4 TCP Segment Delay (Server of Network Bradford)

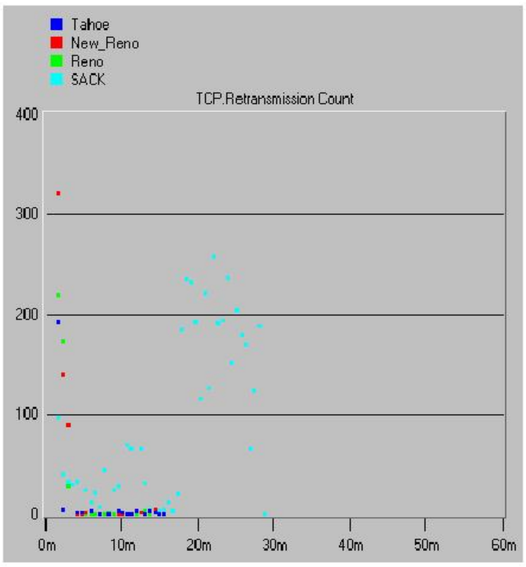

Figure 5 TCP Retransmission Count (Server of Network Bradford)

As wireless links can be very unreliable, they will generate higher segment loss than due to network congestion in wired networks. Therefore, retransmitting lost segments without delay will make for faster recovery without waiting for the congestion avoidance measures. Tahoe as compared to other types has less segment delay and retransmission count as shown in Figures 4 and 5.

5. CONCLUSIONS

This paper has described OPNET implementation of TCP/IP over satellite networks and discussed how to improve TCP performance under these circumstances. As we know that TCP performs poorly over channels with long delay paths and high bit error rates. This is because of the loss of data segments which, reduces TCP's ability to fully utilize channel capacity and also increases the response time over a long delay path. From the results of simulations, it is clear that due to many segments lost in the wireless communication, TCP Tahoe gives better performance as compared to the other TCP standards. Therefore, Tahoe is recommended for the best and high performance of TCP over satellites.

6. REFERENCES

[1] M. Kojo, D. Astuti, L. Daniel, A. Nyrhinen, and K. Raatikainen, "Enhancing TCP Performance Over Satellite Networks - A TCP/IP-Friendly Link-Layer Approach", *Technical Report C-2004-50*, University of Helsinki, Department of Computer Science, 2004.

[2] M. Allman, H. Balakrishnan, and S. Floyd. "Enhancing TCP's Loss Recovery Using Limited Transmit". *RFC 3042*, Internet Society, 2001.

[3] V. Jacobson, "Congestion avoidance and control", in *Proceedings of the ACM SIG-COMM*, pp. 214-329, August 1988.

[4] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control", *RFC 2581*, IETF, April 1999.

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

[6] S. Floyd and T. Henderson, "The NewReno modification of TCP's fast recovery algorithm", *RFC 2582*, April 1999.

[7] J. Hoe, "Improving the start-up time behavior of a congestion control scheme for TCP", *in Proceedings of the ACM-SIGCOMM*, pp. 270-280, August 1996.

[8] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP selective acknowledgment options", *RFC 2018*, IETF, October 1996.

[9] OPNET Technologies Inc., Washington DC, USA. OPNET Modeling Manual. URL:http://www.opnet.com/