Effectiveness of TCP SACK, TCP HACK and TCP Trunk over Satellite Links

Lillykutty Jacob, K.N. Srijith, Huang Duo, A.L. Ananda
Centre for Internet Research
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543

Abstract - **This paper reports the study on the performance enhancements of two extensions to the standard TCP implementation - Selective Acknowledgement (SACK) and Header Checksum (HACK) - over satellite links that are characterized by high latency and high bit error rate. We also examine the effectiveness of TCP Trunk, an edge-to-edge aggregation and congestion control mechanism, over the satellite link. Our study on the effect of varying the TCP window size over long latency link for New Reno, SACK, HACK and TCP Trunk implementations show that increasing window size does improve the performance, but only up to a certain value of the window size, and a further increase actually reduces the performance. Other interesting observations from our experimental study are: SACK enabled TCP Trunk across the satellite link edge routers can improve the throughput regardless of the end host TCP implementation; disabling the link layer CRC and instead implementing the HACK extension to the TCP (and of course HACK+ SACK) can improve the throughput further.**

I. INTRODUCTION

In the recent few years, there have been several experimental studies on the efficacy of using standard TCP/IP protocol suite over satellite channels. Allman and Kruse [1] present several references to those studies as well as the main results. There are several IETF standardized mechanisms [2] as well as a number of other possible TCP mitigations [3] that may allow TCP to better utilize the available bandwidth provided by networks containing satellite links. In this paper, we focus on analyzing the efficiency of three mechanisms, Selective Acknowledgment (SACK) [4], Header Checksum (HACK) [5] and TCP Trunk [6,7]. We conducted our experiments over two network test beds, one using link layer emulator and the other using a Geosynchronous Earth Orbit (GEO) satellite link.

Rest of the paper is organized as follows. Section 2 gives a brief description of the various problems unique to satellite environment. Section 3 outlines some background information on the different extensions to TCP that we study in this paper. In Section 4, the experimental setup is presented. The results and discussions are presented in Section 5. Section 6 concludes the paper.

II. PROBLEMS WITH TCP/IP OVER SATELLITE LINKS

Satellite links have a number of characteristics that may degrade the performance of TCP over it. The important ones among them are discussed below.

A. *Long RTT*

Satellite links have an average RTT of around 500ms. TCP uses the slow start mechanism to probe the network at the start of a connection. Time spent in slow start stage is directly proportional to the round trip time (RTT) and for a satellite link, it means that TCP stays in slow start mode for a longer time than in the case of a small RTT link. This drastically decreases the throughput of short duration TCP connection. Furthermore, when packets are lost, TCP enters the congestion control phase, and due to higher RTT, remains in this phase for a longer time, thus reducing the throughput of both short and long duration TCP connections.

B. *Large Bandwidth-Delay Product*

Bandwidth-Delay product is a measure of the amount of data in flight on a link at any point of time. Because of the high RTT of the satellite, the bandwidth-delay product tends to be very large. In windowed protocols like TCP, the value of the bandwidth-delay product is very critical. To fully utilize the link, the window size of the connection should be equal to the bandwidth-delay product.

In TCP, without the window scaling option, the largest window size allowed is 64KB. For a GEO satellite the maximum throughput achievable is:

$$throughput_{\max(satellite)} = \frac{advertised\ window}{round\ trip\ time} \approx \frac{64KB}{500ms} \approx 1024Kbps$$

Thus, a TCP connection, which might be using a satellite link of 2Mbps bandwidth, ends up achieving a maximum throughput of just over 1Mbps. In addition, many TCP stack implementations use advertised window sizes much less than 64KB by default. To remedy this, RFC 1323 [8] specifies that TCP may use a maximum window size of up to 2^{30}, by using the window scaling option.

C. *Transmission Errors*

One of the biggest drawbacks of TCP is that TCP cannot distinguish between packet loss due to link error and that due to link congestion. Any segment lost is always considered as caused by congestion and TCP performs congestion avoidance with the receipt of three duplicate ACKs or slow start in the case of timeout. As mentioned before, because of the long RTT value, once entered, TCP on satellite links will take a longer time to return to the throughput level that the connection was enjoying just before entering the congestion control phase. Thus errors on a satellite link have more

0-7803-7400-2/02/$17.00 (C) 2002 IEEE

deleterious effect on the performance of TCP rather than over low latency links.

III. Background Work

New Reno is the present default TCP implementation in most systems. It, as the name suggests, is an improvement to Reno, wherein the need for the sender to wait for a retransmit timer when multiple packets are lost from a window is eliminated. However, even with the New Reno implementation, when multiple packets are lost from a window, TCP may end up either retransmitting packets that might have already been successfully received, or retransmitting at most one dropped packet per round-trip [9]. In the remaining part of this section we provide a brief outline of the extensions and mechanisms of TCP, the effectiveness of which we study in this paper.

A. TCP SACK

To overcome the limitations of TCP New Reno, a selective acknowledgment (SACK) mechanism was defined in RFC 2018 [4]. With TCP SACK, the data receiver can inform the sender about all the segments that have arrived successfully, allowing the sender to retransmit only the segments that have actually been lost. In [10], Floyd addressed various issues related to behavior and performance of TCP SACK. In another paper [9], Floyd and Fall analyzed the performance of TCP SACK with respect to Tahoe and New Reno using simulations, and found that SACK was able to provide better performance. In [11], Allman et al. discuss the performance of TCP SACK over satellite links. Our experiments on TCP SACK extend these studies to investigate the joint impact of long latency, corruption, congestion, window size and file size on TCP performance.

B. TCP Trunk

TCP Trunks were first suggested and studied by Kung and Wang in [6]. A TCP Trunk is a TCP circuit between two edge routers that carries IP packets over the network. It is an aggregate traffic stream whose data packets are transported at a rate dynamically determined by TCP's congestion control mechanism. In our experiments with TCP Trunk, we deploy a TCP Trunk between the two gateways across the satellite link.

C. TCP HACK

TCP HACK [5] is an extension proposed for the TCP protocol to improve its performance over lossy links. In HACK, by adding two more option fields to TCP (Header Checksum option and Header Checksum ACK option), a connection is able to recover uncorrupted header of TCP packets with corrupted data and determine that packet corruption and not congestion has taken place along the link. TCP can then respond accordingly and avoid going into congestion control phase.

IV. Experimental Setup

To analyze the performance of SACK, HACK and TCP Trunk, we conducted a variety of experiments using both the link emulator as well as the GEO satellite link. We used the link emulator because of the limited testing time allowed with the satellite link and also because of the flexibility. We used Intel Pentium machines running Linux 2.2.14-5.0 kernel for our experiments. Iperf [12] was used to generate TCP and UDP traffic. Tcpdump [13] was used to observe the TCP packets and tcptrace [14] was used to analyze the output from tcpdump. The experimental testbed using the link emulator is show in Figure 1 and the testbed using the GEO satellite link is show in Figure 2. The error/delay box in Figure 1 was used to corrupt and delay packets in the network to simulate lossy and long latency environments, respectively.

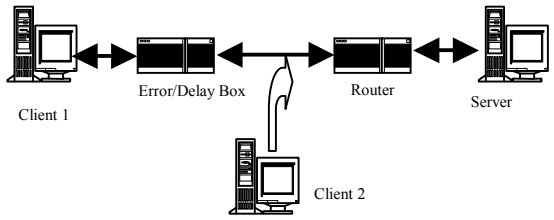

Fig. 1. Experimental testbed 1

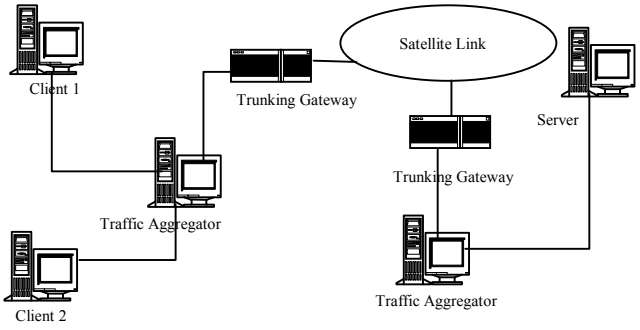

Fig. 2. Experimental testbed 2

V. Results

A. SACK

We compare the performance of TCP SACK under various conditions of corruption and congestion. The experiments were conducted using both the link emulator and the GEO satellite link.

a. Link Emulator – No Corruption

Our first set of experiments tested the performance of TCP New Reno and SACK over the long latency link in a no corruption environment. An RTT of 510 ms was introduced by the error/delay box to simulate the long latency of the satellite link. We sent files of different sizes (100KB, 1MB and 10MB) from Client 1 to Server (Figure 1). The TCP window size was also varied from 32KB to 1024KB. At the server, the goodput obtained was measured.

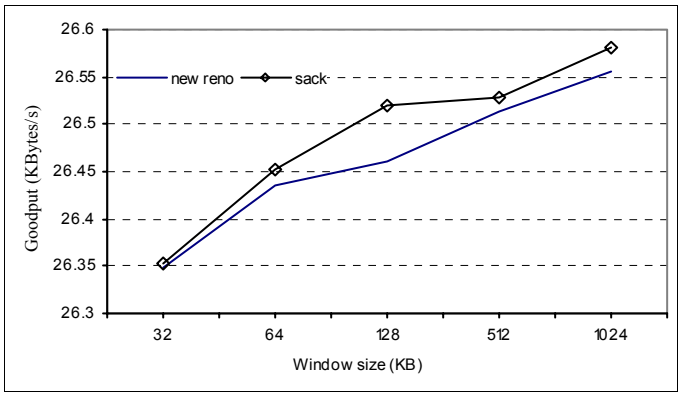

Fig. 3. Goodput for 100KB file transfer for different window sizes - no corruption

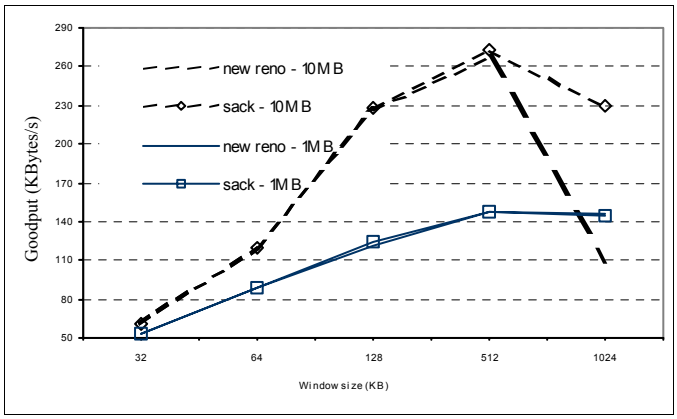

Fig. 4. Goodput for 1MB and 10MB file transfers for different window sizes - no corruption

It can be seen from Figures 3 and 4 that when the window size is increased, the goodput generally increases, except for the case with 10MB file and window size of 1024KB. The result is consistent with earlier published results and endorses the TCP window scaling option in RFC 1323 [8]. It is interesting to see that the goodput for 100KB and 1MB file transfers are smaller than that for the 10 MB file transfer for all window sizes. This is because in both the cases, there is not enough data to actually take advantage of the large TCP window size and fill the link.

The performances of New Reno and SACK are comparable except for very large window size. The comparable performances of New Reno and SACK are expected because of no loss scenario. However, for a large window size of 1024KB, for the 10MB file transfer (Figure 4), the goodput decreases in both cases, but more in the New Reno case. This can be explained as follows. For a link with RTT of 510 ms and bandwidth of 10Mbps, the bandwidth-delay product is 652.8KB. When the window size is larger than this value (1024KB in this case), congestion sets in and the throughput falls. As SACK is able to handle dropped packets by using the selective ACK feature, it fares better compared to New Reno. Thus we see that blindly increasing window size to a very large value will actually adversely affect the throughput, even if SACK is used.

b. Link Emulator – Corruption

In the next set of experiments, we introduced packet errors into the link using the delay/error box. The packet corruption rate was set to 1% and the RTT remained 510ms. Note that 1% packet error rate corresponds to around 0.9×10^{-6} bit error rate (BER), for a segment size of 1460 bytes. File transfers of 1MB and 10MB were carried out with varying window sizes. Figure 5 shows the resulting throughputs obtained.

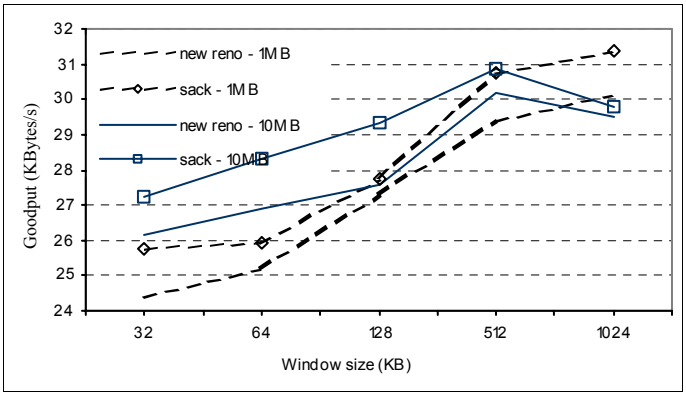

Fig. 5. Goodput for 1MB and 10 MB file transfers for different window sizes - 1% corruption

The above graphs show that SACK provides better performance compared to New Reno when link experiences corruption. As with the previous case of no corruption, the 10MB file transfer goodput decreases when window size is increased beyond 652.8KB because of the presence of congestion in addition to corruption. SACK is able to handle this situation better and provides a better goodput. The percentage reduction in goodput from no corruption to corruption case is more with 10MB file transfer because of the ten times larger number of packets loss events and consequent reduction in the TCP congestion window during the transfer.

We repeated the above experiment with packet corruption rates of 0.5% and 2%. Figure 6 summarizes the results observed for the 10MB file transfer.

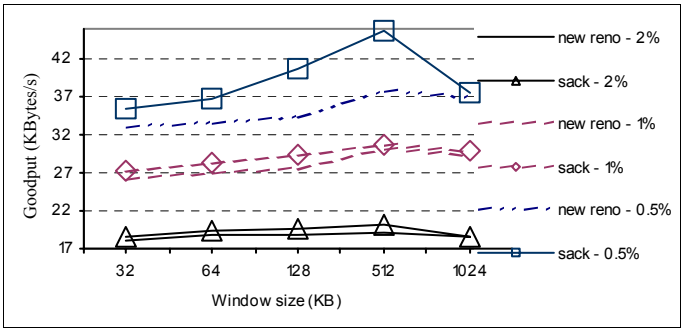

Fig. 6. Goodput for a 10MB file transfer for different window sizes and corruption rates

SACK performs better than New Reno for all corruption levels. However, we see that the margin of improvement is

greatest for lower corruption rates. As corruption rate increases, the difference in throughput between SACK and New Reno decreases, especially for very large window sizes. For the largest window size we used, 1024KB, the difference between the goodput (in bytes) for different corruption rates is as follows:

Corruption	SACK	New Reno	Difference
0.5 %	37055	37653	598
1.0 %	29509	29798	289
2.0 %	18463	18684	221

Table 1. Goodput for 10MB file transfer using 1024KB window

We see that in the presence of congestion (1024KB window), when the corruption rate is more, the improvement that SACK provides decreases. This might be because of the fact that the option field limits SACK to informing the sender about the receipt of a maximum of 4 blocks of data (i.e., 3 or 4 blocks of lost data). When both congestion and corruption are present, instances of more than 4 blocks of data getting lost increases. It might also be because of the fact that, due to high packet loss, the receiver receives no data packets and hence no ACKs are sent which can carry the SACK information back to the sender. In these cases, even SACK is not able to respond properly and times out more frequently. This causes a decrease in the goodput.

c. Link Emulator – Congestion and Corruption

In the experiments so far, we noticed onset of congestion when the window size was larger than the delay-bandwidth product. In the next experiment we introduced congestion due to non-responsive UDP flow. We used a setup where only TCP data was corrupted. The corruption rate was set to 0% (no corruption, only congestion) and 1%. The window size was fixed at 512KB so that we could take full advantage of a larger window without running the risk of creating additional link congestion. The file size used was 10MB. The results for 1% corruption are shown in Figure 7. Similar behavior was observed for 0% corruption, but of course with higher goodput values.

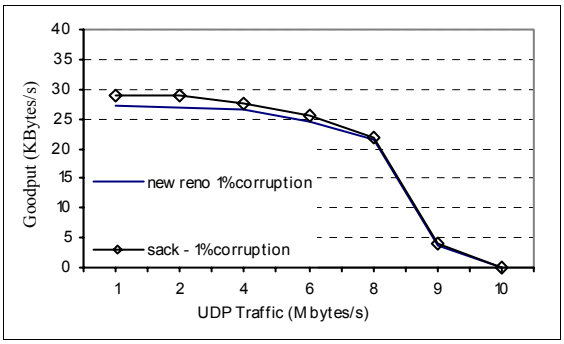

Fig. 7. Goodput for 10MB file transfer – 1% corruption and background UDP traffic

In this congestion scenario, with both 0% and 1% packet corruption rates, SACK again gives a better goodput compared to New Reno when the combined effect of corruption and congestion is moderate. This is because, when the combined effect of corruption and congestion is high, SACK will not be able to recover fast enough and the throughput will decrease. Notice that when 10MBytes/s UDP was pumped into the network, TCP connection was killed, as expected, because the non-responsive UDP traffic was taking up the whole link bandwidth of 10Mbps.

d. Satellite Link

Here we discuss the performance of TCP New Reno and SACK using the satellite link in Figure 2. The satellite link has a bandwidth of 2Mbps and an RTT of 510ms. 1MB and 10MB files were transferred from Client 1 to Server. Window size was varied from 64KB to 128KB and finally to 256KB. Each experiment was repeated 5 times and the average value was taken for the final analysis. Table 2 presents the goodputs measured in KBytes/s.

	1MB New Reno	1MB SACK	10MB New Reno	10MB SACK
64KB	13	14	16.5	17.6
128KB	13.75	15	16.5	18.5
256KB	12.5	13	15.75	17.75

Table 2. Goodput for 1MB and 10MB file transfers for varying window size – satellite link

The results are similar to that obtained using the link emulator. The goodput increases when the window size is increased, as long as the window size is kept less than the bandwidth-delay product (2Mbps*510ms =130.56KB). When the window size is set to 256KB, we notice that the throughput decreases for both New Reno and SACK. This is consistent with the results obtained earlier while using the link emulator. The results also show that SACK performs better than New Reno for both the file sizes as well as for all the window sizes used. This too is consistent with the results of the previous experiments.

B. TCP HACK

Next we analyze the comparative performance of TCP HACK [5] and SACK, for the satellite link latency case. The link emulator was used for the experiments. We modified the device drivers of the Ethernet cards to stop them from discarding packets that failed the packet CRC checks. As a result, corrupted packets arriving at the network cards were passed up to the TCP/IP stack without being discarded. We could not perform the experiments with the real satellite link due to the inability to disable the link layer CRC.

a. Burst Error

First we compare the performance of SACK and HACK in bursty error (i.e., a few contiguous packets – as many as the burst length – in error) conditions with the window size set

high enough not to be the limiting factor. Figures 8 and 9 show the results of the various TCP schemes under 2% and 5% burst error probability.

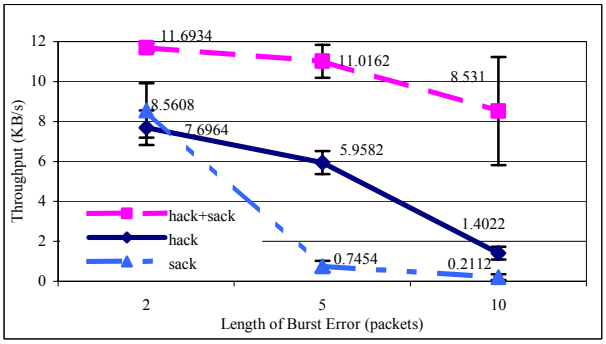

Fig 8. Throughput for 2% burst error for various burst lengths

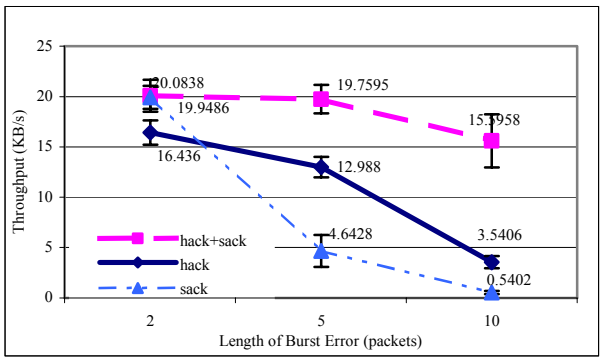

Fig 9. Throughput for 5% burst error for various burst lengths

We can see from the graphs that HACK performs substantially better than SACK. The reason is that when TCP loses too many packets in a row (within the same window), SACK is not able to respond properly. HACK performs better because it can retrieve the TCP header of the corrupted packets and use those headers to generate ACKs and thus keep the pipe flowing. We can also see that HACK is able to perform better when SACK is also activated. This is because HACK is able to make use of the out of order retransmission of SACK, when it gets the ACKs. These out of order situations are created when HACK is not able to retrieve the headers of all the packets that have been corrupted. HACK sends ACKs for those packets whose headers it is able to retrieve. This creates a gap in the receiving window. SACK then uses its mechanism to retransmit selectively the packets whose header HACK is not able to retrieve.

b. Effect of window size

The effect of window size on the performance of HACK is considered next and we compare it with the performance of SACK. Figure 10 shows the throughput of the TCP connections for an error probability of 2% and window size of 16KB, while Figure 11 shows that for a 64KB window size.

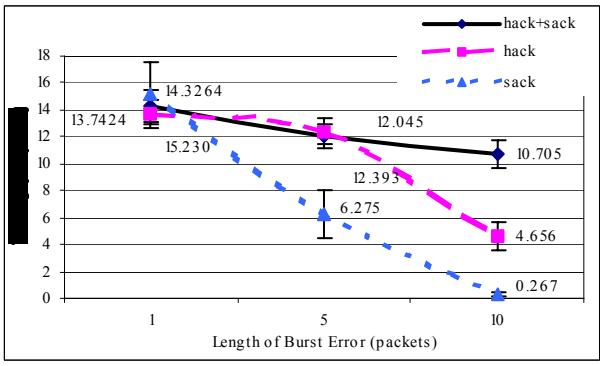

Fig 10. Throughput for 2% burst error for various burst lengths (window size of 16KB)

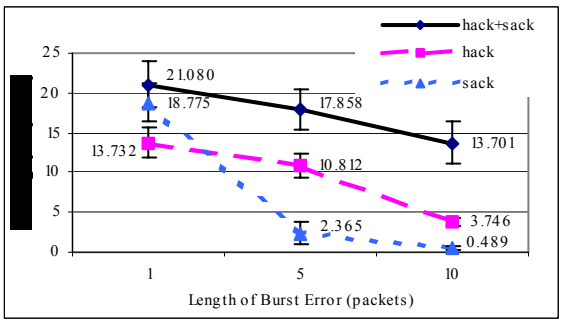

Fig 11. Throughput for 2% burst error for various burst lengths (window size of 64KB)

As expected, HACK performs better than SACK and HACK plus SACK performs still better. The reasons are the same as explained earlier.

C. TCP Trunk

We used testbed 2 (Figure 2) for evaluating the effectiveness of TCP Trunk over satellite link. The traffic aggregator and the Trunk gateway can also be integrated into a single edge device. We concentrated on 10MB file transfers and did experiments first with New Reno TCP Trunk and then with SACK enabled TCP Trunk. For this SACK enabled TCP Trunk, we also varied the window size for the TCP connection between the Trunk gateways from 64KB to 128KB and finally to 256KB. Figure 12 shows the total throughput for transfers from both the sources.

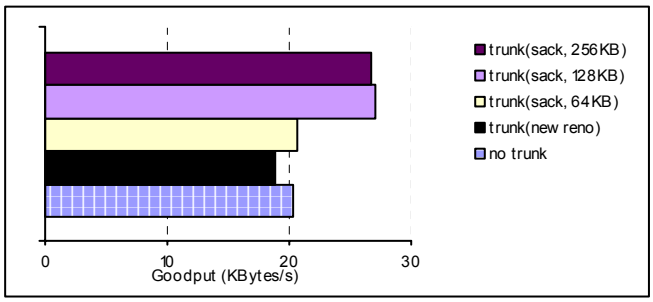

Fig. 12: Throughput observed for a 10MB file transfer for different options

We observe that TCP Trunk with New Reno gives a goodput less than what we obtain without the Trunk. The primary reason might be the presence of two separate congestion control mechanisms, one between the end hosts and the other between the Trunk gateways. That is, when packets are lost, the Trunk times out. However, if Trunk times out it is very likely that the end-to-end TCP connection also times out. This is because the Trunk is over a long latency link (i.e., satellite) and the client to the Trunk is a short latency link. Thus retransmissions may happen twice, from the end host as well as from the Trunk gateway. This will decrease the throughput. This assertion is given credibility when the actual amount of data transferred across the link is analyzed. Table 3 shows the actual amount of data transferred in both cases. "Excess" is the difference in the amount of data transferred across the link when Trunk is used and when it is not used, expressed as a percentage of the data sent without the Trunk.

File size	5MB	10MB
Data transferred without trunk (bytes)	5046336	10114232
Data transferred with trunk (bytes)	5401835	11675170
Excess	7.04%	15.43%

Table 3: Total data transferred across the link

Another reason for the poor performance of Trunk with New Reno is that the Trunk is implemented at the user-space using libpcap [15]. The process of snooping up of packets from kernel space to user space and then putting it back to the kernel space may be introducing enough delay to make a significant impact on the throughput.

However, when SACK was enabled for the Trunk TCP, the throughput observed was larger than when the Trunk was not deployed. This is because SACK can use the ACKs to trigger out of sequence retransmissions, without waiting for timeout. The Trunk alone will do these retransmissions and the end host will not retransmit because timeout has not happened yet. When the window size for the TCP Trunk was increased, the throughput was also increased.

We also see that the throughput for a 128KB SACK enabled Trunk implementation is higher than that of the 256KB SACK enabled Trunk. This is because when we increase the window size beyond the bandwidth-delay product (2Mbps x 510ms = 130.56KB), the connection may try to send more data than what the link can hold. When this happens, packets are dropped and throughput decreases. This is what we see in Figure 12. When the window size is increased but kept below the bandwidth-delay product value, we see an increase in the throughput, but when the window size is set to 256KB, which is more than the bandwidth-delay product, the performance degrades, yielding smaller throughput for the TCP connection.

VI. CONCLUSION

In this paper, we showed that TCP SACK does perform better than New Reno in long latency error and congestion prone environment. However, there is a limit to which SACK can be helpful. We showed that SACK has no effect when the level of corruption or congestion increases. Our experiments prove that HACK performs better than SACK in similar conditions and that HACK plus SACK compliment each other and perform better than each individually. We showed that TCP Trunk is not always good over a long latency link. Trunk implementation with New Reno gave lesser throughput than without the Trunk. However SACK enabled Trunk performed better compared to the scenario without Trunk. The increase of window size does increase the overall throughput of a connection. However, when the window size is set more than the bandwidth-delay product, lesser throughput is experienced, for both New Reno and SACK.

ACKNOWLEDGMENT

We would like to thank Mr. Leong Kit Hoong and his team at Satellite-Internet Competency Unit of Temasek Engineering School, Temasek Polytechnic, Singapore, for the satellite time they provided us.

REFERENCE

[1] M. Allman, H. Kruse, "A History of the Improvement of Internet Protocol Over Satellites Using ACTS", Proceedings of ACTS Conference 2000.
[2] M. Allman, D. Glover, NASA Lewis, L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", *RFC 2488*, IETF, 1999.
[3] M. Allman et.al., "Ongoing TCP Research Related to Satellites", *RFC 2760*, IETF, 2000.
[4] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, "TCP selective acknowledgment and options", *RFC 2018*, IETF, 1996.
[5] R.K. Balan, B.P. Lee, K.R.R. Kumar, L. Jacob, W.K.G. Seah, A.L. Ananda, "TCP HACK: TCP Header Checksum Option to Improve Performance over Lossy Links", *Proceedings of IEEE INFOCOMM 2001*, vol. 1, pp. 309—318.
[6] H.T. Kung, S.Y. Wang, "TCP Trunk: Design, Implementation and Performance", *Proceedings of ICNP*, 1999.
[7] B.P. Lee, R.K. Balan, L. Jacob, W.K.G. Seah, A.L. Ananda, "TCP Tunnels: Avoiding Congestion Collapse", *Proceedings of LCN 2000*, pp. 408-417.
[8] Jacobson, R. Barden, D. Borman, "TCP Extensions for High Performance", *RFC 1323*, IETF, 1992.
[9] K. Fall, S. Floyd, "Simulation-based Comparison of Tahoe, Reno and SACK TCP", *Computer Communications Review*, vol. 26, pp. 5—21, 1996.
[10] S. Floyd, "Issues of tcp with sack", *Technical Report*, LBL Networking Group, 1996.
[11] M. Allman, C. Hayes, H. Kruse, S. Ostermann, "TCP Performance over Satellite Links", *5th Int'l. Conference on Telecommunication Systems*, 1997.
[12] Distributed Application Support Team, "Iperf", http://dast.nlanr.net/Projects/Iperf
[13] Joseph Shaw, "Tcpdump", http://www.tcpdump.org
[14] S. Ostermann, "Tcptrace" http://www.tcptrace.org
[15] Joseph Shaw, "Libpcap", http://www.tcpdump.org