

TCP-SuperCharger : A New Approach to High-Throughput Satellite Data Transfer

By Jack Yiu-Bun Lee, Peter Tak-Shing Yum and Wing-San Wan

*Department of Information Engineering
The Chinese University of Hong Kong, Hong Kong*

It is well-known that TCP does not perform well in modern satellite-based communications due to the channel's very large bandwidth-delay product. The conventional solution is to make use of TCP's Large Window Scale (LWS) extension as defined in RFC1323 to increase TCP's windows size. However this can only be done if either the operating system or the application can be modified to explicitly make use of TCP's LWS extension during connection setup. This work proposes a novel TCP-SuperCharger (TCP-SC) to enable TCP to fully utilize the underlying network bandwidth without modification to the applications or the operating system. TCP-SC is implemented as a gateway along the path between the sender and the receiver. By estimating the receiver's processing capacity, TCP-SC allows the sender to transmit more data than the amount of receiver buffer available. As modern computers are very fast compared to network speed, this prevents the receiver's buffer limit from restricting the throughput achievable by TCP. Experimental results show that TCP-SC can enable existing TCP with default buffer size (64 KB) to achieve 98% link utilization over a satellite link such as the WINDS communications satellite with 1,000 ms RTT and 24 Mbps bandwidth.

Key Words: TCP, Bandwidth Delay Product, Satellite Communication

Nomenclature

D	: Instantaneous RTT between the gateway and the receiver
q_i	: Length of i -th packet
f_i	: Time at which the packet i is forwarded by the gateway to the receiver
t_i	: Time at which the ACK i arrived at the gateway
a_i	: Advertised window of the i -th acknowledgement
$B(t)$: Predicted receiver buffer availability at time t

1. Introduction

It is well-known that TCP does not perform well in modern satellite-based communications due to the channel's very large bandwidth-delay product (BDP). For example, the WINDS communications satellite launched recently by JAXA has bandwidth starting from 24 Mbps all the way up to 155 Mbps. Even with the lowest bandwidth of 24 Mbps, a round-trip-delay (RTT) of 500ms will lead to a bandwidth-delay product of 1.5 MB – a value far exceeding TCP's maximum advertised window size of 64 KB. In this case TCP's flow control mechanism will limit the throughput to no more than 1 Mbps – less than 5% of the satellite's link-layer bandwidth capacity.

The conventional solution to the above problem is to make use of TCP's Large Window Scale (LWS) extension as defined in RFC1323. This extension allows TCP to negotiate during connection setup a multiplier to apply to the window size so that a window size larger than 64 KB can be used. However this approach relies on two assumptions. First, either the Operating System or the application needs to be modified to explicitly make use of TCP's LWS extension. Second, there

must be a way for the application to request the use of LWS during connection setup.

While these two assumptions can be easily satisfied in the laboratory where custom network applications and operating systems can be developed to exploit TCP's LWS extension, they will likely prevent the vast amount of network applications already available in the Internet to benefit from TCP's LWS extension.

We analyzed the TCP advertised window size of a number of popular network applications on the Microsoft Windows platform, including web browsers (Internet Explorer, Firefox, Google Chrome), FTP clients (Windows' built-in FTP client), and Outlook Express, and were surprised to find that not only they do not make use of TCP's LWS option, their default advertised window size is merely 17 KB (depending on the connection type), which will certainly lead to severe underutilization of high-speed satellite links.

To tackle this problem we propose a novel TCP-SuperCharger (TCP-SC) to enable TCP to fully utilize the underlying network bandwidth, but without the need to modify the network applications running on both ends of the communication session nor even requires LWS support from the operating system.

The principle of TCP-SC is based on the observation that although the advertised window cannot be adjusted without modification to the network application, it is possible to send more data than the amount specified by the advertised window size. For example, Fig. 1 plots the advertised window size of a browser downloading a file from a web server. It is clear that the application's advertised window size is small (12 TCP segments totaling 17 KB) so the achievable throughput is far lower than is otherwise possible (only 256 Kbps on a 100 Mbps link).

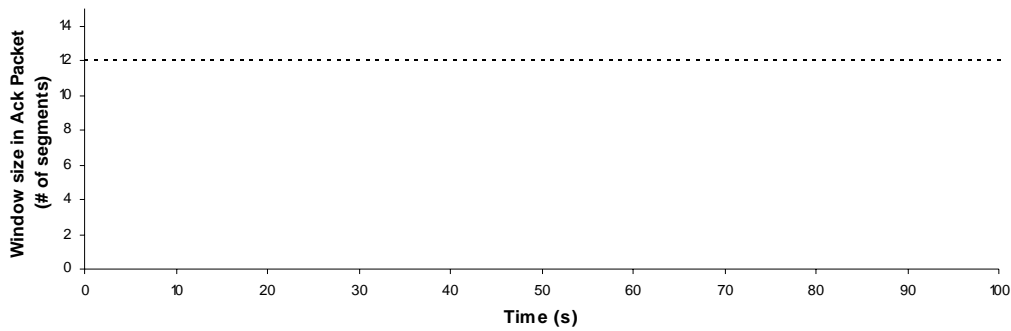


Fig. 1 Receiver’s advertised window size remains at maximum throughout the experiment session

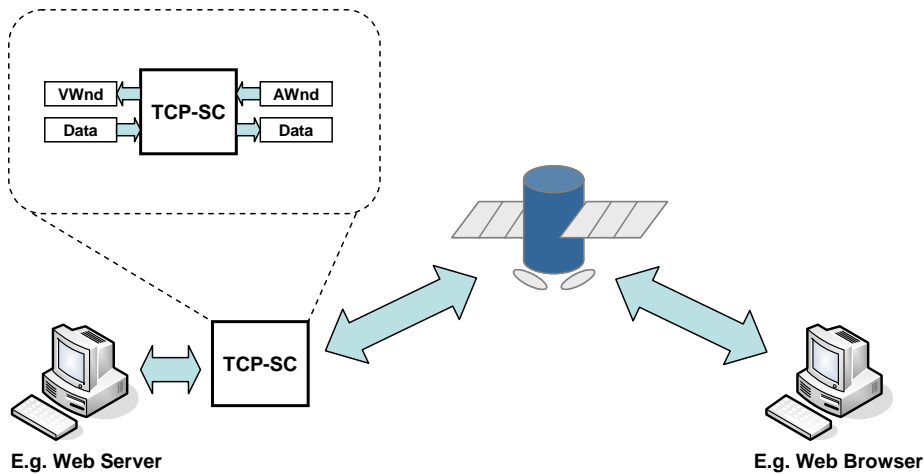


Fig. 2 A TCP connection modulated by a TCP-SC gateway

Now a more subtle observation is that the window size advertised by the receiver, although small, is consistently at the upper limit. In other words the receiver has no trouble processing all incoming TCP packets so that the receiving buffer is consistently near empty.

This observation opens up an entirely new interpretation of TCP’s advertised window size – it is an *indication* of the receiver’s processing capability instead of the absolute amount of buffer space available at the receiver. Consequently this implies that the sender is no longer constrained by the absolute value of the receiver’s advertised window size and instead can transmit more data than the absolute value of the advertised window. In other words, this new approach enables the system to increase the window size without modifying either end of the communicating applications.

The TCP-SC gateway in Fig. 2 is designed to carry out this new function. Specifically, when it receives an acknowledgement (ACK) packet from the receiver which carries an updated advertised window size (AWnd), it will compute a new *virtual* advertised window size (VWnd) that takes into consideration of the receiver’s processing capability and buffer availability to enable it to forward more data than AWnd would allow so that a higher throughput can be achieved. Experimental results show that TCP-SC can allow existing TCP-based network applications to fully utilized the

network bandwidth capacity even using the default window size of 64 KB running over a satellite link such as the WINDS communications satellite with 24 Mbps bandwidth and RTT = 1,000 ms.

The rest of the paper is organized as follows: Section 2 reviews previous related works and compare them with TCP-SC; Section 3 reviews TCP’s flow control mechanism and discuss limitations of TCP’s LWS extension; Section 4 presents details of the proposed TCP-SC gateway; Section 5 reports the performance of standard TCP in sending data over the WINDS communication satellite; Section 6 reports the performance of TCP when accelerated by a TCP-SC gateway; and Section 7 concludes the paper.

2. Related works

Much research had been done to improving the performance of TCP in large BDP networks. We can classify the existing works into three categories: modifying both sender and receiver; modifying the sender only; and modifying the receiver only.

2.1. Sender-receiver-based approaches

We first consider approaches where both the sender and the receiver are modified. Jacobson *et al.* proposed in RFC 1323¹⁾ the Large Window Scale (LWS) extension to TCP which is

currently the most widely supported solution. It works by scaling the AWnd by a constant factor throughout the connection. With the maximum LWS factor 14, the maximum AWnd can be increased up to 1 GB $((2^{16}-1)*2^{14}\approx 2^{30})$. We will discuss the strengths and weaknesses of LWS in more detail in Section 3.

Alternatively, the application can be modified to initiate multiple TCP connections in parallel^{2,3)} to increase throughput by aggregating multiple TCP connections. This approach effectively multiplies the AWnd and CWnd by the number of TCP flows and so can mitigate the AWnd limitation. However, aggregating multiple TCP connections will also allow the application to gain unfair amount of bandwidth from competing TCP flows. Hacker *et al.*⁴⁾ solved this problem by deferring CWnd increase until multiple ACKs are received so as to compensate for the larger window size.

2.2. Sender-based approaches

Apart from AWnd limit, the congestion window maintained by the sender may also limit TCP's throughput in large BDP networks. Specifically, the growth of the CWnd is triggered by the arrival of ACKs. Thus in a long delay path it may take a longer time for the CWnd to grow to sufficiently large value so that the link bandwidth can be fully utilized.

To tackle this problem Allman *et al.* proposed in RFC3390⁵⁾ to initialize the CWnd to a larger value (as opposed to one TCP segment) so that it can grow more quickly in large delay networks to ramp up TCP's throughput. Since then much effort had been out into developing more sophisticated congestion control algorithms such as CUBIC⁶⁾, BIC⁷⁾, FAST⁸⁾, H-TCP⁹⁾ to further improve TCP's throughput performance.

These solutions tackled the limitation of CWnd growth and thus are complementary to our work, which tackles AWnd-imposed throughput limit.

2.3. Receiver-based approaches

At the receiving end, Fisk and Feng¹⁰⁾ proposed dynamic right-sizing of the AWnd by estimating the CWnd at the receiver and then dynamically adapt the receiver buffer size, i.e., the AWnd, to twice the size of the estimated CWnd. This ensures that when the sender's CWnd doubles (e.g., after receiving an ACK) the AWnd will not become the bottleneck.

More recent operating systems such as Linux 2.4 and Microsoft Windows Vista¹¹⁾ also implemented receiver buffer size auto-tuning by estimating the BDP from the data consumption rate. In comparison, TCP-SC does not require any modification to the receiver application or require support from the operating system, and so can be more readily deployed by an ISP or a satellite operator to accelerate all TCP traffics.

3. TCP flow control revisited

TCP's built-in flow control mechanism is designed to prevent fast sender from overflowing slow receiver. It works by reporting the receiver's buffer availability, i.e., the advertised window, back to the sender via a 16-bit field inside the TCP header so that the sender can prevent sending more data than the receiver's buffer can store.

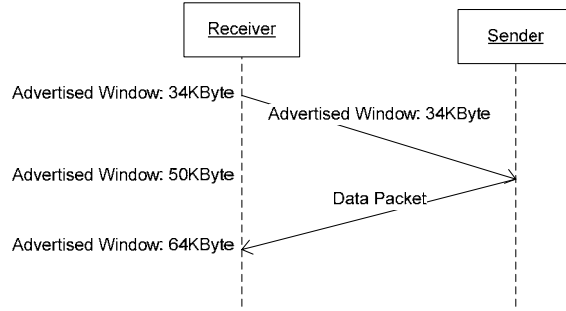


Fig. 3 Real versus delayed advertised window size

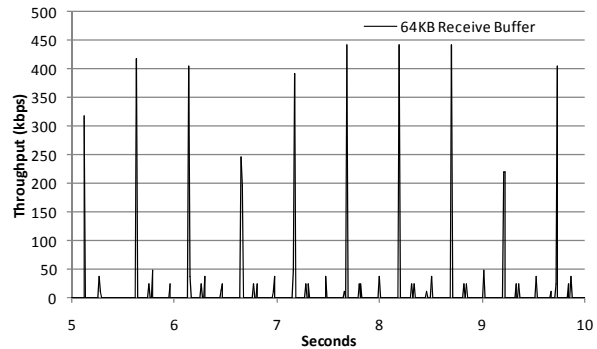


Fig. 4 TCP transmits data in a stop-and-go manner due to AWnd limitation (link bandwidth is 24Mbps)

Over the years computer processing power have grown tremendously such that even today's modest computers can easily keep up with the arriving stream of data up to hundreds of Mbps data rate. Thus an arrived packet will quickly be retrieved by the application from the receiver buffer, and in most cases this can be completed even before the next packet arrival. As a result, the reported AWnd simply stays at the maximum receiver buffer size as illustrated in Fig. 1. In such cases TCP's flow control mechanism is not activated at all as it is not needed.

However TCP's flow control mechanism can become the performance bottleneck in networks with large BDP. Consider the scenario in Fig. 3 where a sender is connected to a receiver over a large BDP link (100 Mbps, 250 ms one-way delay, BDP = 50 Mb). Ignoring processing time, when the sender receives an ACK, the reported advertised window size is in fact the value 250 ms ago (i.e., 34 KB). During this time the receiver application could have retrieved additional data from the receiver buffer, thereby freeing up more buffer space (i.e., 64 KB).

Due to the delayed AWnd the sender cannot send more than the reported AWnd and thus cannot make use of the extra buffer space available at the receiver. In cases where the BDP is larger than the maximum AWnd, the sender will operate in a stop-and-go manner as illustrated in Fig. 4, resulting in severe underutilization of the network channel.

The conventional solution – TCP’s Large Window Scale extension, was designed specifically to address this limitation by allowing the two hosts of a TCP connection to negotiate a constant multiplying factor to be applied to the AWnd value during connection setup so that AWnd larger than 64 KB can be used. Obviously this requires both communicating hosts to support *and* activate the use of LWS *prior* to connection setup is carried out. Unfortunately although most modern operating systems support LWS, there is no standard way for the application to activate the use of LWS. Some operating systems activate LWS by default (e.g., Windows Vista), some requires manual configuration (e.g., Windows XP), and yet others require the application to configure the socket to use >64 KB buffer to activate it.

A second limitation of LWS is that the AWnd value is still interpreted as the amount of buffer physically available at the receiver (and sender as well), thus for networks with very large BDP the resultant buffer requirement can be very large as well (e.g., with 655 Mbps over 1,000 ms RTT a minimum buffer of over 80 MB will be needed). This could become a problem for applications that make use of large number of sockets (e.g., server applications) or for embedded systems with limited physical memory.

If we reconsider Fig. 1 then we can expect that such a large buffer at the receiver will end up mostly unused as the received data will be retrieved out of the TCP buffer quickly by the application. In the next section we present a novel TCP-SC gateway which (a) can enable TCP to fully utilize a large-BDP link; (b) does not require support of LWS extension on the receiving end of TCP; (c) does not require modification to the operating system or the network application; and (d) does not require the allocation of large buffers at the receiver.

4. TCP-SC Gateway

The proposed TCP-SC gateway locates between the sender and the receiver as shown in Fig. 2. All IP packets are routed through the TCP-SC gateway before forwarding to the satellite for uplink transmission. Thus in this case the path between the gateway and the receiver is the large BDP path. The TCP-SC gateway filters out TCP packets for processing and simply forwards all other packets to maintain compatibility with other traffics (e.g., UDP-based protocols).

The principle of TCP-SC is to estimate the receiver’s buffer availability at the time the to-be-forwarded TCP segment arrives at the receiver. If the estimated buffer availability is sufficiently large to accommodate the TCP segment then it will be forwarded to the receiver immediately. Otherwise the gateway will defer transmission to a later time until when sufficient receiver buffer is predicted to become available.

To predict the receiver buffer availability the TCP-SC gateway will need three inputs, namely the receiver’s buffer availability at a previous time instant; the time it takes for the forwarded TCP segment to arrive at the receiver; and the rate at which the receiver application processes data received inside the TCP buffer (i.e., remove received data from the TCP buffer). The first input can be determined from the AWnd in the last ACK received. The remaining two inputs

will need to be estimated and will be presented in Section 4.1 and 4.2.

In the following we assume (a) the gateway always has data to forward; (b) network delays remain constant; (c) network delays of the link between the gateway and the receiver are symmetric in the forward and the reverse direction; and (d) the receiver generates an ACK immediately upon the arrival of a TCP segment, i.e., zero processing delay.

4.1. Gateway-receiver RTT estimation

Let D be the RTT between the gateway and the receiver. Obviously the RTT is not known *a priori* and thus will need to be estimated from measurements. Let f_i be the time packet i was forwarded by the gateway to the receiver, and let t_i be the time at which the corresponding ACK arrived at the gateway. Then the RTT D can be computed from:

$$D = (t_i - f_i) \quad (1)$$

as we assume symmetric network delays.

To smooth out random fluctuations in the estimated RTT, the gateway will apply exponentially weighted moving averaging to the measured values:

$$RTT^i = \alpha \times RTT + (1 - \alpha) \times D \quad (2)$$

where the weight $\alpha=0.9$ follow the ones used in TCP’s internal RTT estimator¹³.

4.2. Receiver processing rate estimation

The receiver’s processing rate can be estimated from comparing the receiver’s buffer availabilities between two time instants. Obviously the time at which ACKs arrive at the gateway will be good candidates for these two time instants as every ACK packet contains the receiver buffer availability inside the AWnd field.

Let t_i and a_i be the arrival time and the AWnd value of ACK packet i . Let f_j and q_i be the time TCP packet j was forwarded by the gateway and the segment size of the packet i respectively. Then for some positive integer k , the processing rate, denoted by R , can be computed from

$$R = \frac{(a_{i+k} - a_i) + \sum_{\forall j|f_j \in (t_i, t_{i+k}]} q_j}{t_{i+k} - t_i} \quad (3)$$

where the first term in the numerator is the difference in the receiver’s buffer availability and the second summation term is the total amount of data transmitted during the time interval. The parameter k controls the width of the estimation interval (in number of ACK packets) and can be adjusted to tradeoff between accuracy and timeliness of rate estimation. Exponentially weighted moving averaging similar to (2) is applied to R to smooth out random fluctuations.

4.3. Gateway transmission scheduling

Whenever a TCP segment is received from the sender, the gateway must schedule it for transmission to the receiver such that it will not cause buffer overflow at the receiver. The tricky part is that the AWnd reported in the last ACK is delayed information - it was the receiver’s buffer availability 0.5D s ago. During the time the ACK travelled to the gateway additional TCP segments may have arrived at the receiver, and the receiver application may have processed more data from the receiver buffer.

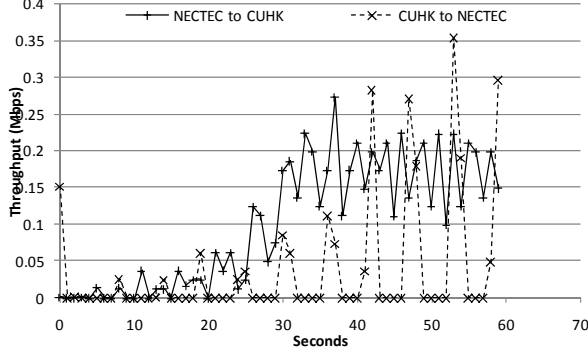


Fig. 5 TCP throughput anomaly over satellite channel

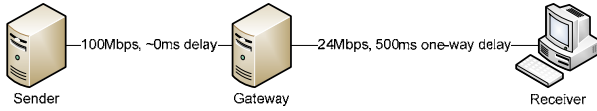


Fig. 6 Setup of the emulated experiments

To account for these two factors the gateway can compute the predicted receiver buffer availability at a future time t , denoted by $B(t)$ from:

$$B(t) = a_i - \sum_{\forall j | (f_j + 0.5D) \leq t} q_j + R(t - t_i) \quad (4)$$

where the first term is the AWnd reported by the i^{th} ACK, the second term is the predicted total amount of data arriving at the receiver from time t_i to time t , and the last term is the predicted amount of data which will be processed by time t .

Now the scheduling problem can be restated as finding the set of transmission schedule $\{f_j | j=0,1,\dots\}$ such that $B(t) \geq 0 \forall t$. In practice this can be done whenever a TCP segment, say segment i , is received from the sender. The gateway then determines f_i according to the previous constraint and then queue it for transmission. Queued TCP segments will then be processed in a FIFO manner with the head-of-line segment transmitted when the current time reaches the schedule time f_i .

5. WINDS satellite experiments

We conducted experiments over the WINDS communication satellite with the support of JAXA and NECTEC Thailand in Jan 2009. The original objectives are to characterize the communications parameters of WINDS for future emulated experiments and to evaluate the performance of TCP-SC over the satellite channel. We were successful in the first objective but failed to complete the second one due to unforeseen difficulties. We summarize below the measured WINDS satellite communications parameters and reported an anomaly we encountered.

At the beginning of the experiment, we first transmitted UDP data between the Chinese University of Hong Kong (CUHK) and NECTEC over the WINDS satellite. The channel behaved normally, achieving 40 Mbps with negligible packet loss. However when we switched to TCP the achievable throughput dropped significantly to 0.12 Mbps. With a TCP window size of 64 KB and a typical satellite link RTT of 1s we would expect a throughput of around 0.5 Mbps.

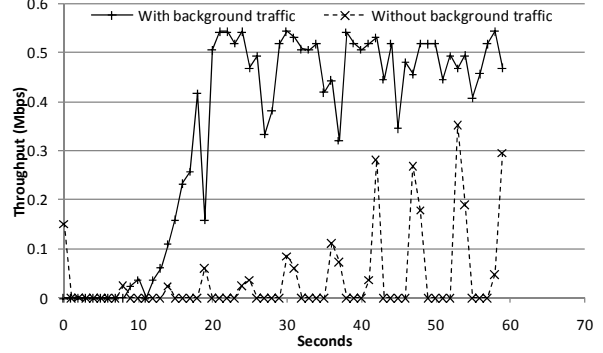


Fig. 7 TCP throughput restored to normal with background UDP traffic

Table 1. Summary of WINDS satellite measurements

Protocol	Background flow	Receive buffer size (KB)	RTT (seconds)	Throughput (Mbps)
UDP	N	64	5.78	40
TCP	N	64	5.78	0.12
TCP	N	512	5.78	0.8
TCP	N	1024	5.78	4
TCP	Y	64	1.03	0.5

Fig. 5 plots the TCP throughput versus time for both directions of data transfer in this experiment. It suggests that the TCP flow, especially in the direction from CUHK to NECTEC, periodically stopped transmitting data completely for a significant duration.

Next we measured the RTT between CUHK and NECTEC and were surprised to find that it consistently exceeded 5 seconds, which is significantly longer than expected. Based on these two measurements we conjectured that the satellite downlink could have been released automatically after certain channel idle time, after which a lengthy channel setup process is required to re-establish the downlink, thus leading to the long RTT.

To test this conjecture we introduced a background UDP data flow at 400 Kbps from NECTEC to CUHK in parallel to the TCP flow aimed at keeping the channel busy to prevent it from being released. With the background UDP traffic the TCP throughput increased to 0.5 Mbps as expected (see Fig. 7). Similarly with the background UDP traffic the measured RTT decreased to 1s as expected. These two experiments strongly suggest that the anomaly was caused by channel setup time although it is uncertain which component is responsible. Further investigations are warranted to isolate the anomaly and to devise a long-term solution.

Table 1 summarizes the measured parameters for the WINDS satellite channel. We adopt the set of parameters when there is UDP background traffic for emulated experiments in the next section.

6. Emulated Experiments

With the measured satellite parameters we setup a testbed to emulate the satellite link behavior as depicted in Fig. 6 to evaluate the performance of TCP-SC. The sender and the gateway are running on Linux kernel 2.6 (with TCP-CUBIC as the congestion control module) and the receiver is running on Windows XP.

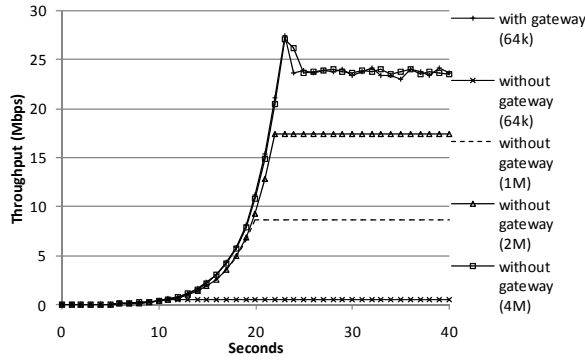


Fig. 8 Comparison of TCP throughput under various settings

Table 2. Average throughput after TCP stabilizes (from 25s to 40s)

Receive buffer size (KB)	64	64	1024	2048	4096
With gateway	Y	N	N	N	N
Throughput (Mbps)	23.67	0.53	8.69	17.39	23.72
Link utilization (%)	98.63	2.20	36.21	72.45	98.86

Network bandwidth limit is emulated using Netem¹⁴⁾ which is built-in into the Linux kernel and network delay is emulated by the gateway itself. This resulted in a BDP of 3 MB which is significantly larger than the default receiver buffer size. The TCP connection is initiated by the receiver and then the sender would keep sending data towards the receiver as fast as TCP allows. We logged TCP's throughput at the receiver.

As we developed the receiver application we can change the receiver buffer size by manually setting the socket buffer size. We conducted control experiments (i.e., with TCP-SC) for receiver buffer sizes of 64 KB, 1 MB, 2 MB, and 4 MB respectively and compare them to the case of TCP-SC with 64 KB buffer size in Fig. 8.

As expected TCP's throughput increases proportionally with the receiver buffer size. In comparison, with TCP-SC in-place TCP can achieve throughput similar to the case of 4 MB receiver buffer size. Since the BDP is only 3 MB this implies that the achieved throughput is no longer receiver-buffer-limited. In fact the link utilization reached 98.63% for TCP-SC and 98.86% for 4 MB buffer size respectively (see Table 2).

Fig. 9 plots the AWnd of all five sets of experiments. It is worth noting that regardless of the receiver buffer size setting, the buffer availability stays at the maximum buffer size setting. This suggests that the receiver application's processing rate is higher than the network bandwidth so that data arriving at the receiver are quickly retrieved and cleared from the buffer. Thus although the 4 MB setting also enabled TCP to fully utilize the network, the extra buffers allocated are in fact not used at all. By contrast, TCP-SC enables existing TCP to fully utilize network bandwidth without the need for large buffers and modifications to the receiver application.

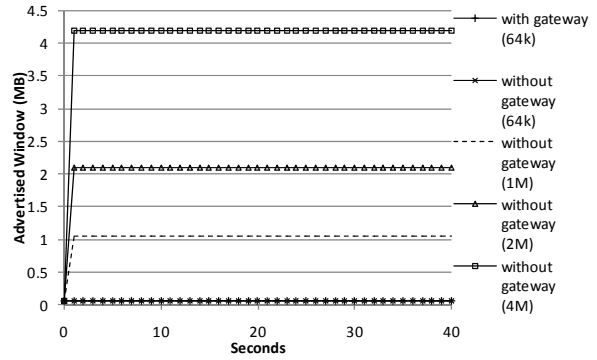


Fig. 9 Advertised window size reported in ACK packets

7. Conclusion

This work tackled the performance problem of running TCP over large BDP networks by introducing a TCP-SC gateway between the sender and the receiver. Compared to existing solutions the proposed TCP-SC gateway exhibit two desirable properties: (a) it does not require modification to the applications, TCP implementation at the hosts, or operating system; and (b) it significantly reduces the receiver buffer requirement. TCP-SC is compatible with existing TCP flows and can readily enable them to fully utilize available network bandwidth. It can be deployed as part of a satellite communications network and is completely transparent to its users.

Acknowledgement

The authors would like to express their gratitude to JAXA for providing access to the WINDS satellite; to the Association for Application Experiments of WINDS for their coordination and support of the application experiment; and to Dr. Chalermopol Charnsripinyo and his team of NECTEC, Thailand for their generous support of our application experiment. This work was funded by the Shun Hing Institute of Advanced Engineering of the Chinese University of Hong Kong under project number MMT 9/07.

References

- 1) V. Jacobson, R. Braden and D. Borman, "RFC 1323: TCP extensions for high performance," May 1992
- 2) J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan and S. Tuecke, "Applied Techniques for High Bandwidth Data Transfers Across Wide Area Networks," *Proceedings of International Conference on Computing in High Energy and Nuclear Physics*, September 2001
- 3) H. Sivakumar, S. Bailey and R. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks," *Proceedings of Super Computing*, November 2000
- 4) T. Hacker, B. Noble and B. Athey, "Improving Throughput and Maintaining Fairness using Parallel TCP," *Proceedings of IEEE Infocom 2004*, March 2004
- 5) M. Allman, S. Floyd and C. Partridge, "RFC 3390: Increasing TCP's Initial Window," October 2002
- 6) I. Rhee and L. Xu "CUBIC: A new TCP-friendly high-speed TCP

- variant," *Proceedings. PFLDNet'05*, February 2005
- 7) L. Xu, K. Harfoush and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," *In Proceedings of IEEE INFOCOM 2004*, March 2004
 - 8) C. Jin, D. X. Wei and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *In Proceedings of IEEE INFOCOM 2004*, March 2004
 - 9) R. Shorten and D. Leith, "H-TCP: TCP for High-Speed and Long-Distance Networks," *Second International Workshop on Protocols for Fast Long-Distance Networks*, February 16-17, 2004, Argonne, Illinois USA
 - 10) M. Fisk and W.-C. Feng, "Dynamic Right-Sizing in TCP," *Proceedings of the Los Alamos Computer Science Institute Symposium*, October 2001
 - 11) J. Davies, "The Cable Guy: TCP Receive Window Auto-Tuning," *TechNet Magazine*, January 2007
 - 11) L. Kalampoukas, A. Varma and K. K. Ramakrishnan, "Explicit Window Adaptation: A Method to Enhance TCP Performance," *IEEE/ACM TRANSACTIONS ON NETWORKING*, VOL. 10, NO. 3, June 2002
 - 12) M. Gerla, R. L. Cigno, S. Mascolo and W. Weng, "Generalized Window Advertising for TCP Congestion control," *European Transactions on Telecommunications*, 2002
 - 13) V. Jacobson and M. Karels, "Congestion Avoidance and Control," Available: <ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z>.
 - 14) The Linux Foundation, "Net:Netem," Available: <http://www.linuxfoundation.org/en/Net:Netem>