Fixing TCP Slow Start for Slow Fat Links

Maryam Ataei Kachooei[+], Pinhan Zhao[+], Feng Li[*], Jae Chung[*], and Mark Claypool[+]
[+] Worcester Polytechnic Institute, Worcester, MA, USA
{mataeikachooei, pzhao2, claypool}@wpi.edu
[*] Viasat, Marlboro, MA, USA
{jaewon.chung, feng.li}@viasat.com

Abstract

TCP slow start is designed to begin at a conservative bitrate, but quickly ramp up to the available bandwidth. Unfortunately, current default Linux TCP socket buffer sizes impede slow start bitrates on large bandwidth-delay product (BDP) links. However, even with our recommended socket buffer sizes in place, traditional slow start does not work well on large BDP links such as satellites, often overshooting and causing significant packet loss. Conversely, TCP HyStart (on by default in Linux), intended to avoid overshooting during slow start, can exit from slow start prematurely which is especially detrimental to utilization on large BDP links. This paper proposes adjustments to TCP slow start that find a safe point to enter congestion avoidance without overshooting, while also avoiding premature exiting that degrades link utilization on large BDP links. We evaluate the proposed slow start algorithm over a commercial geostationary satellite link and our preliminary results indicate that our proposed slow start adjustments improve start-up performance, outperforming the measured alternatives.

Keywords

Satellite networks, TCP, Slow Start, ACK-pair time, Round-Trip Time, Bandwidth estimate

Introduction

Satellite networks are a crucial part of modern networks, covering large areas of the earth without the need for supporting towers and wires. This allows communication without other infrastructure, particularly important during emergency rescue and when other networks are disrupted. Low Earth Orbit (LEO) satellite networks use many small satellites that orbit the Earth at relatively low altitudes, covering 3 and 12 percent of Earth's surface at altitudes of 400 and 2000 km, respectively [1]. LEO satellites are sensitive to weather conditions and have asymmetric bitrates upward and downward, with delays and capacities at 20-40ms and 100 Mb/s [10]. GEO satellite networks use a single satellite at a mucher high altitude (36,000 km), which correspondingly has much higher delays of about 600 ms round-trip [5].

The high latency of the satellite network can impact TCP bitrates and degrade the performance of the network [4, 3]. During slow start, TCP increases the congestion window (`cwnd`) by one for each Acknowledgement (ACK) packet it receives, so the cwnd size approximately doubles for each round-trip time. Upon packet loss, TCP stops doubling the `cwnd` by existing slow start and entering congestion avoidance. TCP buffer sizes can also limit throughputs – TCP sending rates each round-trip time are limited by the smallest of the sender buffer size (`wmem`), receiver buffer size (`rmem`), and congestion window (`cwnd`). When the sender buffer size or the receiver buffer size is smaller than the congestion window, TCP is unable to fully utilize the network, especially likely for network links with both a high round-trip time and high capacity, i.e., slow, fat links such as for satellite networks. Unfortunately, the Linux defaults for sender and receiver buffer sizes are too small to fully utilize GEO satellite links. Specifically, to utilized a 150 Mb/s GEO link, the sender and receiver buffer sizes need to be about 11.25 MBytes, whereas Linux has maximum default buffer sizes of only 4.2 MBytes.

To overcome buffers that limit throughputs on slow, fat links, we recommended maximum Linux sender and receiver buffer sizes to 26 MBytes [11].[1] Figure 1 depicts the impact of this proposed change. The x-axis is the link capacity and the y-axis is the round-trip time, both shown in logscale. The dashed line nearest the origin shows the limits of network utilization for the Linux defaults. In other words, network connections between this dashed line can be fully utilized while network connections beyond this cannot. While typical LEO and 4G links are within this area, GEO and 5G links are not. The second dashed line depicts the effects of our recommended buffer sizes, which increases the set of networks that can be fully utilized to more than encompass GEO and 5G links. For the rest of our paper, we assume Linux settings with our recommended defaults.

As mentioned earlier, TCP is initially in the slow start phase and increments the value of `cwnd` by one each time an ACK packet is received until it reaches the slow start threshold (`ssthresh`) or packet loss occurs, whichever comes first. Since packets lost by overshooting a link's capacity is detrimental to the network, especially for slow, fat links, TCP should instead exit upon reaching the `ssthresh` limit. With this inteny, Hystart [7] was designed to exit TCP slow start before losing packets due to congestion and is enabled by de-

[1] Changing the maximum `rmem` from 6291456 to 26214400 and `wmem` from 4194304 to 26214400.

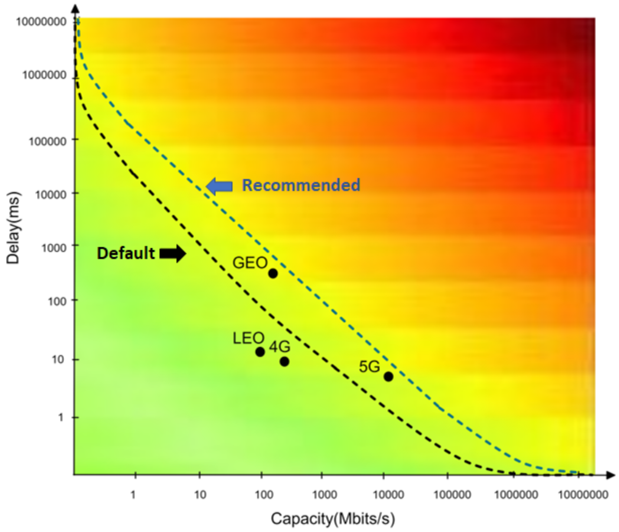


Figure 1: Bandwidth-Delay Product limitations for the default and recommended maximum Linux buffer sizes.

fault on Linux. However, our earlier work demonstrated that HyStart over GEO links causes TCP to prematurely exit slow start, result in underutilization [11].

Our paper investigates the reasons for premature exit of slow start over satellite networks, both for LEO links and GEO links. We then propose a new algorithm to exit slow start at a better point using estimates of the link capacity. This new algorithm, called Bandwidth Estimate Slow starT (BEST), uses timing for ACK pairs to estimate the bandwidth each round-trip time and sets the `ssthresh` accordingly. We measure the performance of the BEST algorithm over actual satellite networks and show improvements over slow start without HyStart, which overshoots, and slow start with HyStart which exits slow start too early.

The remainder of this paper is organized as follows: Related Work summarizes work related to our paper, Methodology describes our testbed and proposed BEST algorithm, Results evaluates our algorithm's performance, and Conclusion summarizes our findings and suggests possible future work.

Related Work

This section reviews improvements to TCP slow start and TCP performance over satellite networks related to our work.

TCP slow start

Guo et al. [6] proposed a stateful-TCP approach where the path bandwidth is estimated from a previous flow and recorded in a hash table along with the minimum round-trip time and destination IP address. Then, this information is used by the subsequent flow such that the initial `cwnd` is set according to the path bandwidth estimated from the previous flow with pacing applied to outgoing packets in the first round-trip time to smooth out the initial transmission. They evaluated the performance of their approach (S-Cubic) applied to Cubic through emulated and actual Internet experiments. This found the average efficiencies of S-Cubic are 96.3%, 98.8%, and 99.3% for 10 ms, 50 ms, and 100 ms round-trip times, respectively, with lower queuing delays than Cubic.

Arghavani et al. [2] introduced the StopEG mechanism to find the best point for stopping the exponential growth of `cwnd` during the slow start phase. This mechanism calculated that the inflight packet numbers would not exceed 56.8% when the bottleneck was not saturated. The results of the ns-3 implementation of StopEG in BBR showed lower delay and higher throughput than the original BBR.

Zhang et al. [13] introduced an algorithm that improves the slow start exit time by using ImTCP to measure the available bandwidth. If there was enough bandwidth, this algorithm increased `cwnd` aggressively, but a sudden rise in `cwnd` could consume all available capacity and cause congestion for other traffic.

Our work extends the above by finding the best slow start exit point, using a bandwidth estimate and the round-trip time to set the exit point, comparing performance to Hystart on and Hystart off.

TCP performance over satellite networks

Liu et al. [8] show the impact of using a Performance Enhancing Proxy (PEP) in a commercial satellite Internet network. They compared the effect of the PEP for Cubic, BBR, PCC, and Hybla protocols. The result showed the performance improvement for all protocols with a PEP, with the most benefits for Cubic start-up, a 3x improvement.

Liu et al. [9] introduced a Markov chain model for a TCP connection with two transition probabilities for `cwnd` in slow start and congestion avoidance phases. They use their model to analyze the performance of TCP New Reno over a satellite. The results showed throughput improvements, but could not adapt to satellite networks with a high BDP.

Utsumi et al. [12] proposed two new analytical models for TCP Hybla in satellite IP networks: a model for steady-state throughput and a latency model. They evaluated their model's accuracy with simulated and emulated satellite links. The results showed an improvement in the performance, with significant improvements in throughput over TCP Reno for loss rates above 0.0001%.

Our work extends the above by providing performance analysis over a commercial satellite Internet network.

Methodology

We used two testbeds to evaluate network performance of TCP slow start - one with a GEO satellite link and another with a LEO satellite link.

Testbed

GEO Testbed Figure 2 shows the testbed with a GEO satellite link provided via a Viasat modem for our client and our server, which resides on our campus. The server connects to the University LAN via Gb/s Ethernet and the campus network itself is connected to the Internet via several 10 Gb/s links.

The client is a Linux PC with an Intel i7-1065G7 CPU @ 1.30GHz and 32 GB RAM. The server has an Intel Ken

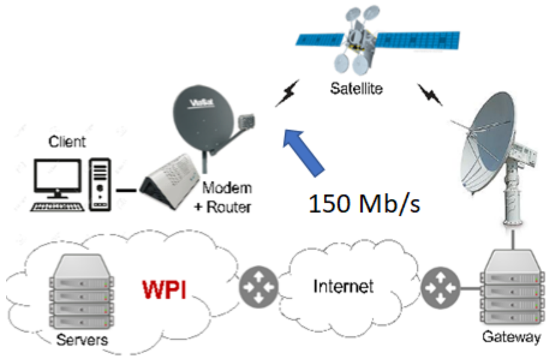


Figure 2: Geo Satellite Measurement Testbed

E312xx CPU @ 2.5 GHz and 32 GB RAM. The server and client both run Ubuntu 18.04.4 LTS, Linux kernel version 5.10.79, and Wireshark captures all packet header data on the server and the client.

The terminal communicates through a Ka-band outdoor antenna (RF amplifier, up/down converter, reflector and feed) through the Viasat 2 satellite[2] to the larger Ka-band gateway antenna. The terminal supports adaptive coding and modulation using 16-APK, 8 PSK, and QPSK (forward) at 10 to 52 MSym/s and 8PSK, QPSK and BPSK (return) at 0.625 to 20 MSym/s.

The Viasat gateway performs per-client queue management, where the queue can grow up to 36 MBytes, allowing a maximum queuing delay of about 2 seconds at the peak data rate. Queue lengths are controlled at the gateway by Active Queue Management (AQM) that randomly drops 25% of incoming packets when the queue is over a half of the limit (i.e., 18 MBytes).

The performance enhancing proxy (PEP) that Viasat deploys by default is disabled for all experiments in order to assess congestion control performance independent of the PEP implementation and to represent cases where a PEP could not be used (e.g., for encrypted flows).

The maximum data rate provided is about 150 Mb/s with a minimum round-trip time of about 600 ms.

LEO Testbed Figure 3 shows the testbed with a LEO satellite, configured similarly to the GEO testbed except that the client connects via a LEO link provided by Starlink instead of a GEO link. The LEO link has a peak down-link data rate of 100 Mb/s, but is sensitive to the weather and with a down-link bitrate greater than the up-link bitrate.

Download During the experiments, the default Linux TCP congestion control algorithm (Cubic) is used, and the performance of the BEST algorithm is compared to both HyStart on and HyStart off by performing multiple bulk downloads. For downloads, the servers and client are configured to use iperf3. The client initiates a connection to one server via iperf and downloads an object. After the download is complete, the client pauses for 1 minute, repeating the download 10 times.

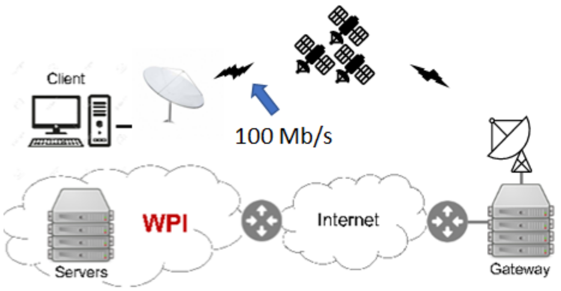


Figure 3: LEO Satellite Measurement Testbed

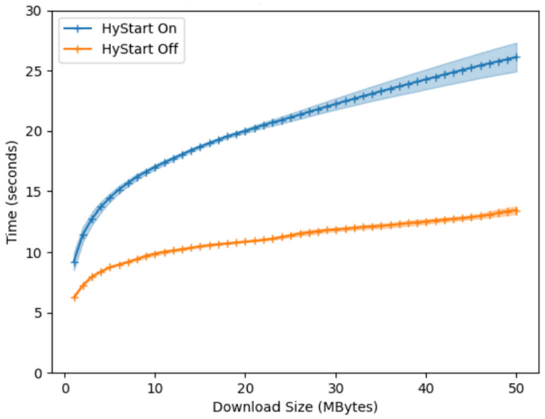


Figure 4: Time to Download Objects of Different Sizes

HyStart for Satellite Network

As previously mentioned, HyStart exits slow start before throughput has ramped up to meet the available capacity of the satellite link, which harms throughput since it then takes a long time for the TCP congestion windows to grow sufficiently large.

In order to compare the performance of the GEO satellite with HyStart on and off, the GEO testbed is used to download objects with different sizes 10 times. Figure 4 depicts the download time for the the first N bytes on the x-axis and time on the y-axis. From the graph, 1 MB downloads take 50% longer with HyStart enabled. As the size of the object to download increases up to 50 MBytes, the difference between the download time for HyStart on and HyStart off increases, taking up to 2x longer for HyStart on.

To understand why HyStart does not work well for GEO satellite networks, several 30-second downloads of the same object were performed. Figure 5 shows the average throughput and RTT for these tests. As the magnification in the first part of the RTT graph shows, the RTT goes increases and then decreases several times. TCP with HyStart considers this RTT change as a saturation of the link and exits slow start. But as is evident by the throughput, the link is not saturated despite the early increases in RTT and thus HyStart

[2] https://en.wikipedia.org/wiki/ViaSat-2

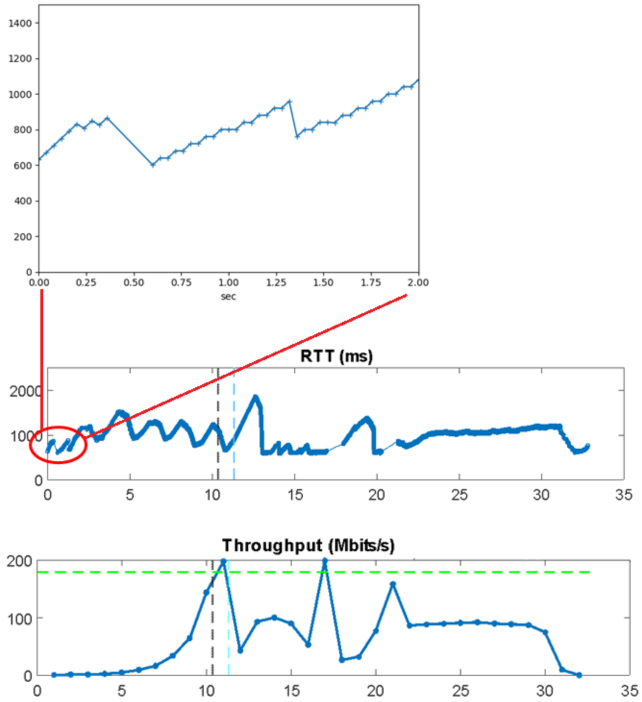


Figure 5: RTT and throughput for GEO satellite network with HyStart on

causes a premature exit from slow start.

We also evaluated HyStart performance for LEO networks using our testbed to download an object several times within 10 minutes. Figure 6 depicts throughput and RTT graphs for these experiments. As shown in this figure, the RTT varies from 20 to 60 ms, and the throughput also fluctuates, sometimes increasing to 350 Mb/s and sometimes dropping to 200 Mb/s independently of the RTT. The lack of correlation in RTT and throughput illustrates the challenges in exiting slow start based on RTT alone, as HyStart does.

Figure 7 depicts the slow start exit time for the LEO network downloads with HyStart on and HyStart off. The purple plus points show the slow start exit times when HyStart is on, and the green square points show the exit times when HyStart is off. With HyStart on, TCP exits slow start much earlier, and the congestion window is capped at small values. With HyStart off, TCP exits slow start later with larger congestion windows. Figure 8 shows the corresponding cumulative distribution function (CDF) of throughout and total retransmissions. From the graph, more packets are lost when HyStart is off, which suggests slow start is overshooting the ideal congestion window size.

BEST Algorithm

As an alternative to slow start exit points when HyStart is on (exits too early on slow, fat links) and when HyStart is off (exits too late for slow, fat links) we propose a new algorithm based on bandwidth estimation called *Bandwidth Estimated Slow starT* (BEST).

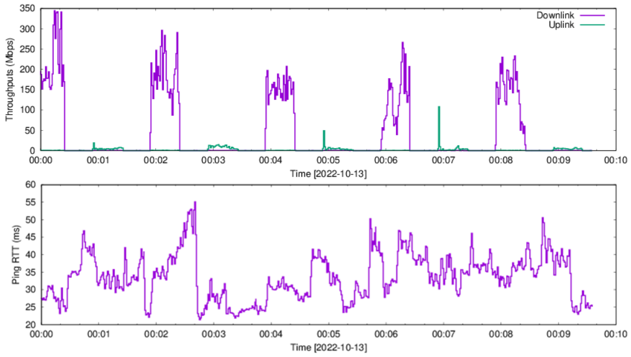


Figure 6: RTT and throughput for LEO satellite network with HyStart on

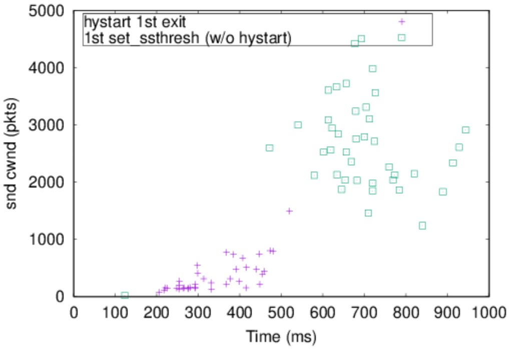


Figure 7: Congestion window size versus slow start exit time for LEO satellite network

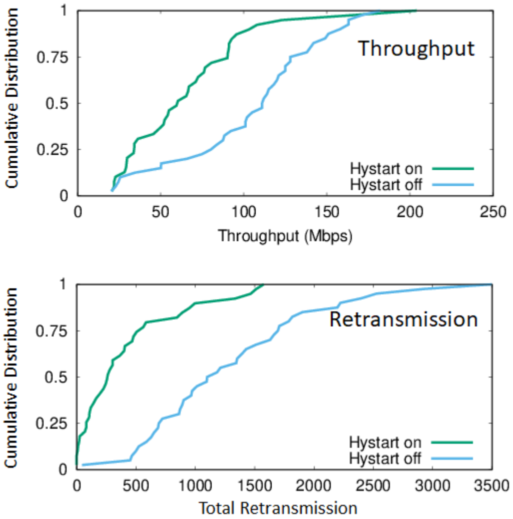


Figure 8: Cumulative distribution function of throughout and total retransmissions for LEO satellite network

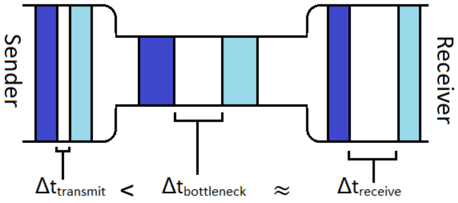

Sender
Receiver
$\Delta t_{transmit}$ < $\Delta t_{bottleneck}$ ≈ $\Delta t_{receive}$

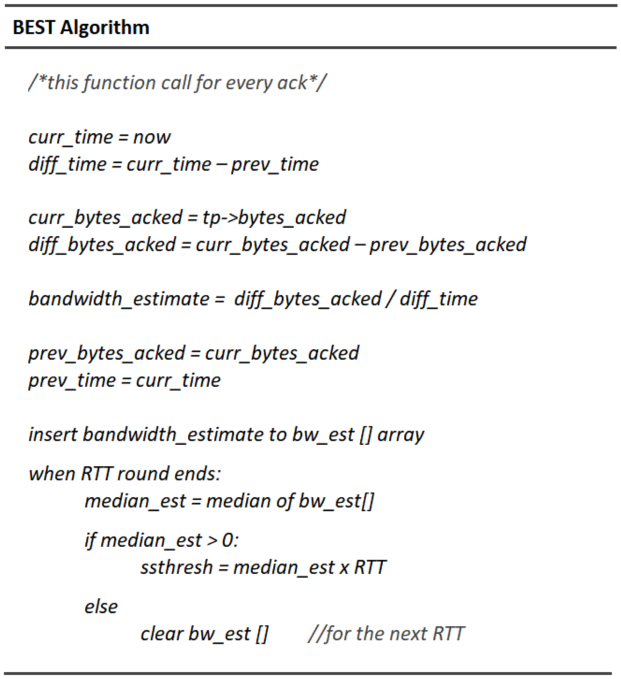
Figure 9: Illustration of packet temporal spacing through a bottleneck link

BEST uses a variant of packet-pair bandwidth estimation. Figure 9 shows two packets of the same size traveling from source to destination. The wide part of the pipe represents a high bandwidth link, while the narrow part represents a low bandwidth link. Packets are sent back-to-back from the sender, but due to the queue at the bottleneck, a space (a temporal gap) appears between the packets and remains on the receiver side. Bandwidth is estimated by using the time between two packets (packet pair time) and packet sizes. Since we only have access to information on the sender, we use receiver-sent Ack packets (Ack pairs) to calculate the bandwidth.

The pseudocode of the BEST algorithm is shown in Figure 10, called each time an Ack packet arrives at the sender. The time when the previous Ack is received subtracted from the current time to get the Ack pair time (diff_time). The number of bytes acked can be obtained (diff_bytes_acked) by subtracting the previous total we know how many bytes were acked. The bandwidth estimate is then the number of bytes acked divided by the time ($bandwidth\_estimate = diff\_bytes\_acked \div diff\_time$).

Using this part of the code, we downloaded an 80 MByte object multiple times and recorded the estimated bandwidth for each run over the GEO satellite link. Figure 11 depicts the estimated bandwidth for one run, where the estimated bandwidth is seperated into 5 groups based on RTT round (only RTT rounds 1 to 5 are shown). Each graph shows the cumulative distribution function of estimated bandwidth in one RTT round and the dashed vertical line indicates the link capacity (150 Mb/s). There are only a few samples for the first RTT (the initial congestion window size is 10), most of which are low, less than 1 MB/s. This is due to scheduling of the Ack packets on the returning satellite link. These low values continue through RTT rounds 2-3. By RTT rounds 4 and 5, the return link scheduler does regular transmissions so there are enough samples to estimate the bandwidth. Given the distribution of values – many too low as noted, but many too high – we filter them by taking the median of the distribution as the bandwidth estimate for the link.

This estimate coupled with the round-trip time is used to set the slow start exit threshold (`ssthresh`).



BEST Algorithm

```
/*this function call for every ack*/

curr_time = now
diff_time = curr_time – prev_time

curr_bytes_acked = tp->bytes_acked
diff_bytes_acked = curr_bytes_acked – prev_bytes_acked

bandwidth_estimate =  diff_bytes_acked / diff_time

prev_bytes_acked = curr_bytes_acked
prev_time = curr_time

insert bandwidth_estimate to bw_est [] array

when RTT round ends:
        median_est = median of bw_est[]

        if median_est > 0:
                ssthresh = median_est x RTT

        else
                clear bw_est []        //for the next RTT
```

Figure 10: Pseudocode of BEST algorithm

The bottom half of the algorithm code in Figure 10 shows the corresponding pseudo code. Each RTT, the estimated bandwidth values are stored in an array and, when the RTT ends, the median determined. If the median is greater than 0, it is used with the RTT to set `ssthresh` to the BDP. Otherwise, the array is cleared for the next RTT round.

Result

This section analyzes the performance of TCP using BEST over a GEO satellite link. We download an 80 MByte object 10 times using our testbed and compare the performance for BEST, the original HyStart (Hystart on), and when HyStart is off. Figure 12 and Table show the download times (average with 95% confidence interval) and the number of packet retransmissions, respectively. When HyStart is on, the number of packet retransmissions is the lowest, but the premature slow start exit results in a high average download time (about 30 seconds). By turning off HyStart, the download time is reduced, but with more packet retransmissions (more than 8000) since slow start overshoots the desired exit point. BEST has the shortest download times and about half the packet losses of HyStart.

Table 1: Number of retransmissions

Algorithm	Retransmissions
HyStart on	88
HyStart off	9111
BEST	4328

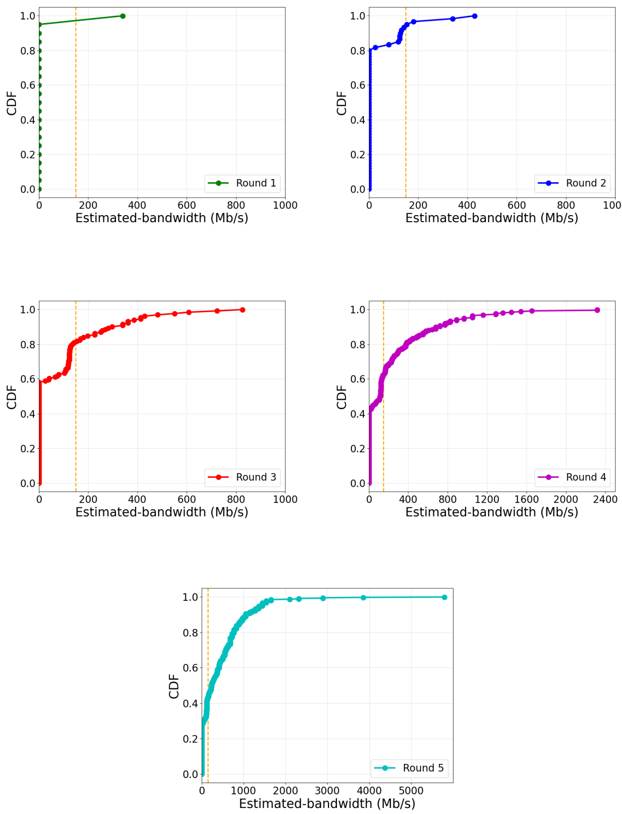


Figure 11: Estimated bandwidth for each RTT (rounds 1 to 5)

Queue occupancy at the bottleneck link is controlled at the Viasat gateway by Active Queue Management (AQM) which randomly drops 25% of incoming packets when the queue is over half of the limit (i.e., 18 MBytes). To assess the effect of different slow start exit times on the download time of an object with varying capacity sizes of the queue, we turn HyStart off and download an 80 MB object for full BDP queue (100% BDP), half BDP queue (50% BDP), and quarter BDP queue (25% BDP) with the Viasat AQM disabled (i.e., using only tail drop). Figure 13 shows the results. When TCP exits slow start too early, the download times are high for all three queue sizes. The object download time decreases as the slow start exit times increase to the lowest point near the capacity (150 Mb/s). However, when TCP exits slow start too late, the download time rises again especially for small queue sizes. We repeat the experiment with the BEST algorithm, shown with the dashed horizontal lines. The BEST results are near the optimal exit points for different queue sizes.

Conclusion

LEO and GEO satellite networks are an important part of today's Internet. However, their high latencies and large capacities (slow, fat links) present challenges for TCP flows that ramp up bitrates in proportion to round-trip times. Default TCP window settings are too small to fully utilize GEO satellite links and should be increased. Moreover, our experiments show that the default TCP HyStart algorithm exits slow start prematurely, causing unnecessary underutilization. Unfortunately, disabling HyStart causes slow start to exit too late, resulting in unnecessary packet losses. The penalty for exiting slow start too early and exiting slow start too late is exacerbated for small bottleneck queue capacities.

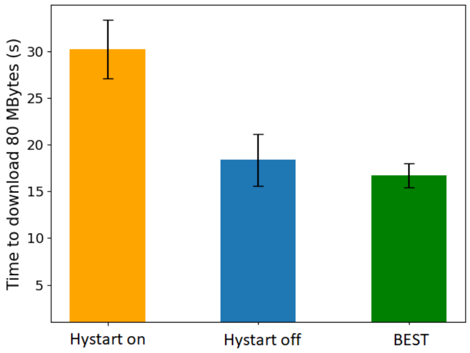


Figure 12: Time to download 80 MB for HyStart on, HyStart off, and BEST

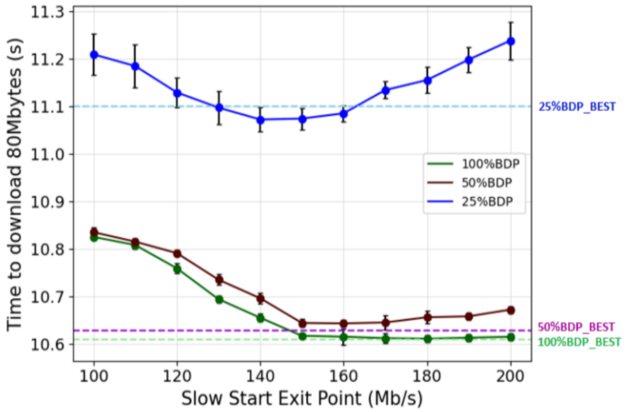


Figure 13: Performance of BEST for different queue sizes

To improve TCP performance over slow, fat links, we propose a new algorithm that uses bandwidth estimation to decide on the exit point – Bandwidth Estimate Slow starT (BEST). BEST estimates bandwidth based on Ack pair times at the sender and when the median estimate for an RTT round is above 1 MB/s, the slow start threshold (`ssthresh`) is set to the BDP based on this estimate and the RTT.

Our performance results show BEST provides shorter download times for 80 MByte objects over a GEO satellite link than HyStart on or HyStart off with more substantial improvements for smaller bottleneck queue sizes.

Future work is to evaluate BEST for LEO satellite links as well as a wide range of wired network conditions. Additional future work is to consider implementation efficiency in terms of memory needed to store bandwidth estimates each round, with heuristics to reduce this if deemed necessary.

References

[1] Al Homssi, B.; Al-Hourani, A.; Wang, K.; Conder, P.; Kandeepan, S.; Choi, J.; Allen, B.; and Moores, B. 2022. Next generation mega satellite networks for access equality: Opportunities, challenges, and performance. *IEEE Communications Magazine* 60(4):18–24.

[2] Arghavani, M.; Zhang, H.; Eyers, D.; and Arghavani, A. 2020. Stopeg: Detecting when to stop exponential growth in tcp slow-start. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, 77–87. IEEE.

[3] Claypool, S.; Chung, J.; and Claypool, M. 2021. Measurements comparing tcp cubic and tcp bbr over a satellite network. In *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 1–4. IEEE.

[4] Deutschmann, J.; Hielscher, K.-S.; and German, R. 2022. Broadband internet access via satellite: Performance measurements with different operators and applications. In *Broadband Coverage in Germany; 16th ITG-Symposium*, 1–7. VDE.

[5] Furqan, M., and Goswami, B. 2022. Satellite communication networks. *Handbook of Real-Time Computing; Tian, Y.-C., Levy, DC, Eds* 1–22.

[6] Guo, L., and Lee, J. Y. 2020. Stateful-tcp—a new approach to accelerate tcp slow-start. *IEEE Access* 8:195955–195970.

[7] Ha, S., and Rhee, I. 2011. Taming the Elephants: New TCP Slow Start. *Computer Networks* 55(9):2092–2110.

[8] Liu, M.; Liu, Y.; Ma, Z.; Porter, Z.; Chung, J.; Claypool, S.; Li, F.; Tutlis, J.; and Claypool, M. 2022. The effects of a performance enhancing proxy on tcp congestion control over a satellite network. In *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 325–331. IEEE.

[9] Liu, J.; Han, Z.; and Li, W. 2019. Performance analysis of tcp new reno over satellite dvb-rcs2 random access links. *IEEE Transactions on Wireless Communications* 19(1):435–446.

[10] Park, C. H.; Austria, P.; Kim, Y.; and Jo, J.-Y. 2022. Mptcp performance simulation in multiple leo satellite environment. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 0895–0899. IEEE.

[11] Peters, B.; Zhao, P.; Chung, J. W.; and Claypool, M. 2021. TCP HyStart Performance over a Satellite Network. In *Proceedings of the 0x15 NetDev Conference*.

[12] Utsumi, S.; Zabir, S. M. S.; Usuki, Y.; Takeda, S.; Shiratori, N.; Kato, Y.; and Kim, J. 2018. A new analytical model of tcp hybla for satellite ip networks. *Journal of Network and Computer Applications* 124:137–147.

[13] Zhang, Y.; Ansari, N.; Wu, M.; and Yu, H. 2012. Afstart: An adaptive fast tcp slow start for wide area networks. In *2012 IEEE International Conference on Communications (ICC)*, 1260–1264. IEEE.