SEARCH: Robust TCP Slow Start Performance over Satellite Networks

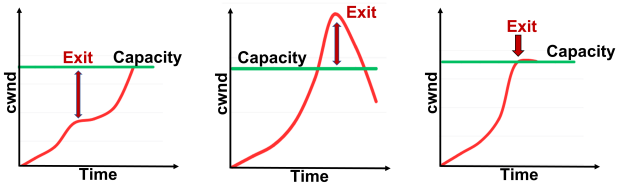
Maryam Ataei Kachooei
Worcester Polytechnic Institute
Worcester, MA, USA
mataeikachooei@wpi.edu

Jae Chung, Feng Li, Benjamin Peters
Viasat
Marlboro, MA, USA
{jaewon.chung, feng.li, benjamin.peters}@viasat.com

Mark Claypool
Worcester Polytechnic Institute
Worcester, MA, USA
claypool@cs.wpi.edu

***Abstract*—TCP slow start begins at a conservative bitrate but quickly ramps up to the available bandwidth. Unfortunately, current TCP implementations can either: 1) exit from slow start prematurely, which is especially detrimental to utilization on satellite links, or 2) exit from slow start too late, causing unnecessary packet loss. We propose a novel technique to exit slow start while avoiding both premature and belated exits. We evaluate our approach over commercial satellite links – long, fat networks that pose challenges to determining the right slow start exit time. Preliminary results show a high success rate for picking appropriate exit points over satellite links, with potentially being applicable to other types of networks, more generally.**

***Index Terms*—Satellite, Round-Trip Time, Congestion**



(a) Too early. (b) Too late. (c) At chokepoint.

Fig. 1: TCP slow start exit points.

I. Introduction

The Transmission Control Protocol (TCP) is a cornerstone of the Internet's communication infrastructure, providing reliable and efficient data transfer between hosts. However, TCP's performance can be severely degraded in high bandwidth-delay product (BDP) networks, such as those over satellite links. Geostationary Earth Orbit (GEO) and Low Earth Orbit (LEO) are satellite-based communication technologies widely used when and where wired communication can be a challenge (e.g., remote areas or during natural disasters). However, the high altitudes of GEO satellites introduce a significant latency in signal transmission – about 300 milliseconds (ms) one-way – which can negatively impact real-time applications [1]. LEO satellite links have lower latency – about 40 ms one-way – but move relative to the Earth's surface, making link capacities and round-trip times (RTTs) vary with time and with weather [2], [3].

The slow start phase in TCP is designed to ramp up to the available bandwidth quickly, doubling the congestion window each RTT until link capacity is reached, whereupon TCP exits the slow start phase. Unfortunately, the default Linux TCP slow start implementation – TCP Cubic with Hystart [4] – has been shown to be harmful over LEO and GEO links, causing premature exit from slow start [5] depicted in Figure 1a. However, without HyStart, TCP exits too late, causing excessive packet loss depicted in Figure 1b. The optimal exit point occurs when the congestion window size meets the link capacity (the chokepoint), depicted in Figure 1c, allowing for full link utilization while avoiding congestion packet loss. However, determining this optimal chokepoint exit is difficult in satellite networks due to their variable RTTs and capacities over time.

Exiting slow start early is especially detrimental in satellite networks due to how long it takes to ramp up the congestion window size to meet the link capacity. While exiting too late is preferred over exiting too early, the ideal is to exit slow start before encountering packet loss. Several techniques have been explored in an effort to detect the optimal slow start exit point, but have not been effective for satellite networks. For instance, bandwidth estimation through packet-pairs [5], [6] has potential but is not robust enough in noisy environments where estimates are either too high or too low. Hystart++ [7] uses increases in RTT to find an exit point, but the variations in satellite network RTTs make any RTT-based exit approach problematic.

We propose an alternative approach based on TCP's delivery of one window of packets each RTT. Figure 2 depicts the idea with RTTs depicted on the x-axes and bytes sent/delivered on the y-axes. All the bytes sent at time t_1 on the left graph are acknowledged as delivered at time t_2 on the right graph. This continues until sending capacity is surpassed at time t_4 whereupon the delivered bytes remain the same at time t_5. Thus, detecting when bytes delivered are significantly lower than bytes sent one round-trip earlier signals that the congestion window limit has been reached and slow start can safely exit. While fairly simple as a concept, the challenge is an implementation that is robust in the presence of the variable RTTs inherent in satellite links. In addition, the implementation must be scalable: a) server-side only for incremental deployment, and b) minimal per-flow resources.

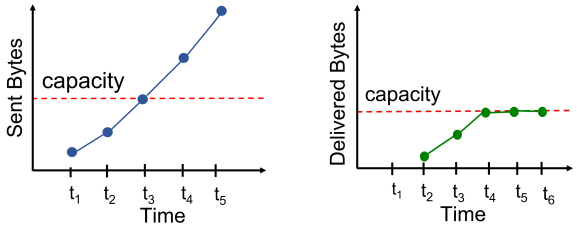


Fig. 2: Theoretical transmission: sent and delivered bytes.

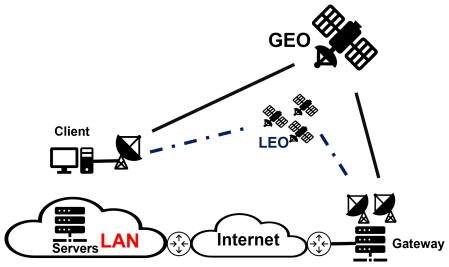


Fig. 3: GEO and LEO satellite measurement testbed.

We propose a new algorithm that monitors the sent and delivered bytes over a large time window to account for variation in RTTs, using server-side acknowledgments to compute delivery rates and estimates to reduce per-flow state. The delivered byte history is shifted back one RTT and the difference between sent and delivered bytes is normalized to account for bitrate increases. The slow start phase is then exited when the difference exceeds a threshold. Preliminary evaluation over GEO and LEO links shows the promise of our approach, determining the ideal slow start exit point most of the time over these networks, some of the most challenging in terms of capacity and RTT variance.

II. Related Work

Round-trip Time: Ye et al. [8] propose Personalized FAST TCP that uses link buffers to judge slow start exit times, being resilient to variations in bandwidth and leading to faster convergence. While promising, the evaluation is based on simulations rather than real network deployments. Balasubramanian et al. [7] propose HyStart++ to address the premature slow start exiting in HyStart. HyStart++ uses increases in RTT to find an exit point and a mitigation to prevent jitter from causing premature slow start exit. While HyStart++ has widespread deployment in Microsoft Windows, the variations in satellite network RTTs make such RTT-based approaches problematic.

Bandwidth Estimate: Lübben [9] proposes forecasting TCP's congestion control rate based on an available bandwidth estimate using a neural network to pick a slow start exit point. The author demonstrates earlier slow start exit times and better TCP fairness. However, the proposed approach relies on accurate measurements of network parameters, challenging in networks with high variability, such as satellite networks. Guo et al. [10] proposed a stateful S-Cubic which estimates bandwidth based on a previous flow, setting the initial congestion window and applying pacing. Performance via simulation shows S-Cubic has higher efficiency and lower queuing delays than Cubic. However, the dependency on historical data is ineffective where such information is not readily accessible or when the network conditions have significantly changed. Kachooei et al. [5] introduce the BEST algorithm that estimates bandwidth (and slow start exit) using packet-pair measurements. However, BEST struggles where the estimated bandwidth and RTTs fluctuate significantly, resulting in measurements that are either too low or too high. Jasim et al. [6] propose approximating TCP congestion window thresholds in high latency connections with packet-pair bandwidth estimates. Their experiments find their approach improves TCP performance, but likely also struggles over satellite networks.

Satellite-Specific: Utsumi et al. [11] develop analytical models for TCP Hybla to improve performance over satellite networks. Their steady-state throughput and latency models yield more throughput improvements over emulated satellite links. These approaches would benefit from further analysis beyond emulation to evaluate its effectiveness and limitations in real-world network environments.

III. Methodology

We used a testbed – shown in Figure 3 – with two different bottleneck links to evaluate the performance of TCP slow start over satellite networks: one link uses a Viasat GEO satellite while the other uses a Starlink LEO satellite. The client is configured with the satellite link as the "last mile" connection downloading from a server, mimicking a typical configuration where the only connection to the Internet is via a satellite. When using the Viasat GEO satellite link, the client connects through a Ka-band outdoor antenna. The Viasat gateway has a queue that can grow up to 36 MBytes and uses Active Queue Management to randomly drop 25% of incoming packets at 18 MBytes. The Viasat performance-enhancing proxy (PEP) was disabled for all experiments to pertain to cases where encryption or cost prevent PEP use. The Viasat link provides a maximum data rate of about 150 Mb/s with a minimum RTT of about 600 ms. When using the Starlink LEO satellite link, the client connects through an electronic phased array outdoor antenna. Starlink provides a peak downlink data rate of about 100 Mb/s with a minimum RTT of about 40 ms.

The client and server run Linux kernel version 5.10.79 is instrumented to record sent bytes, delivered bytes, RTTs, and congestion window sizes. For each experimental run, iperf3 downloads from the server to the client using TCP Cubic – the default TCP congestion control algorithm in Linux with HyStart disabled (it is typically on by default in Linux).

Figure 4a shows the congestion window size (cwnd) versus time during a download via the GEO satellite link. The cwnd growth occurs steadily, doubling each RTT during slow start. This is in contrast to the RTTs during this same time, shown in Figure 4b, which vary considerably even though the downlink is not at capacity. The LEO satellite link also has RTT

fluctuations before reaching capacity limits (not shown due to space limits). These RTT variations pose a challenge to our approach since comparing bytes sent to bytes delivered occurred with different RTTs.

To overcome baseline RTT variance, we use a sliding window of data that is much larger than an RTT, thus smoothing over the observed variance. To avoid storing per-packet information, we aggregate sent and delivered bytes over a much smaller time period – we call this a "bin" – and then aggregate multiple bins to get an approximation for the full window. Since data sent in one RTT is acknowledged as delivered in the next RTT, the delivered bytes are shifted back in time by the current RTT before comparison.

We dub our algorithm *Slow start Exit At Right CHokepoint* (SEARCH). Pseudocode for SEARCH is shown in Algorithm 1, initialized with the parameters at the top for each TCP flow that is established. The window size is 3.5 × the minimum RTT to smooth out variation in link latency. Ten bins are used to approximate the sent and delivered rates over the window size, each bin of equal duration. The threshold (THRESH) is the limit (0.25) in the difference in the normalized sent minus delivered bytes above which slow start exits.

Each acknowledgment, the sent and delivered bytes are updated and stored in an array implemented as a circular queue. When the time (**now**) has passed the bin boundary, the bin index is incremented. Also every acknowledgment, the difference between the total bytes sent for the previous RTT and the total bytes delivered for the current RTT is computed and normalized. If this normalized difference is greater than the threshold (THRESH), ssthresh is set to the current cwnd which causes slow start to exit.

For overhead, SEARCH runs each time an acknowledgment is received, but only with a constant, and limited, number of operations ($O(1)$). The additional memory needed by SEARCH is likewise small, with only two arrays of 10 integers each and a few local variables. The effectiveness of SEARCH relies on adequate queue capacity at the bottleneck, and SEARCH can fail with spurious (e.g., non-congestion) packet loss – but then so does traditional slow start.

Algorithm 1 SEARCH: Slow start Exit At Right CHokepoint.

Parameters:
WINDOW_SIZE = RTT_MIN × 3.5
NUM_BINS = 10
BIN_TIME = WINDOW_SIZE / NUM_BINS
THRESH = 0.25

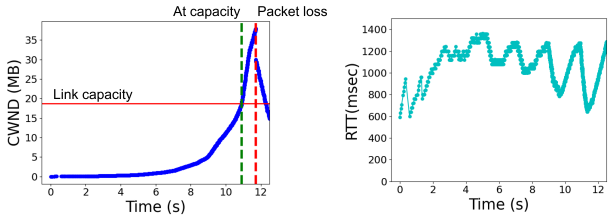
Initialization:
sent[0] = delv[0] = 0
bin_index = 0
bin_end = **now** + BIN_TIME

Each Acknowledgement:
if (**now** > bin_end) **then** // Update bin boundary
  bin_end += BIN_TIME
  bin_index += 1 mod NUM_BINS
  sent[bin_index] = delv[bin_index] = 0
end if
sent[bin_index] += bytes_sent
delv[bin_index] += bytes_delivered
total_sent = $\sum_{prev}$ sent[] // Sum up to (now - RTT)
total_delv = $\sum_{now}$ delv[] // Sum up to now
normalized_diff = (total_sent - total_delv) / total_sent
if (normalized_diff > THRESH) **then**
  set `ssthresh` to `cwnd` // Exit slow start
end if

IV. Evaluation

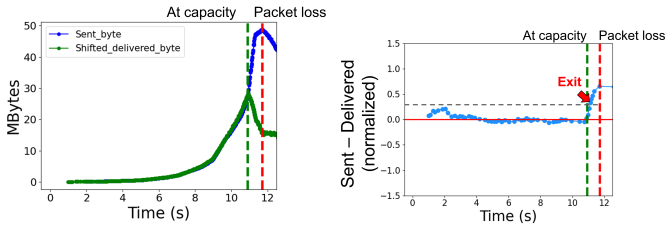
We do preliminary evaluation of SEARCH over both GEO and LEO satellite links using our testbed. The GEO satellite network's sent and shifted (by one RTT) delivered bytes are shown in Figure 5a. The y-axis is the sent/delivered MBytes and the x-axis is the time in seconds. The red vertical line indicates the first packet loss (normally when slow start exits), and the green line is when the congestion window is large enough to reach the link capacity. Together, between the green dashed line and the red dashed line is when slow start should exit. From the figure, the delivered bytes closely track the sent bytes until the link capacity is reached. Subsequently, the sent bytes continue to increase, but the delivered bytes do not. The normalized difference in sent and delivered bytes is shown in Figure 5b. From the figure, the normalized difference crosses the 0.25 threshold between the green and red vertical lines, indicating an appropriate slow start exit point – i.e., after the congestion window has grown to reach link capacity but before it causes packet loss.

To verify the accuracy of the approximation used by SEARCH (full window approximated by 10 equal-length bins), we compared the normalization graph of the best case 2000 ms measurement with that of the approximation. Figure 6a displays both normalized graphs, where the blue line represents the original data, and the orange line represents



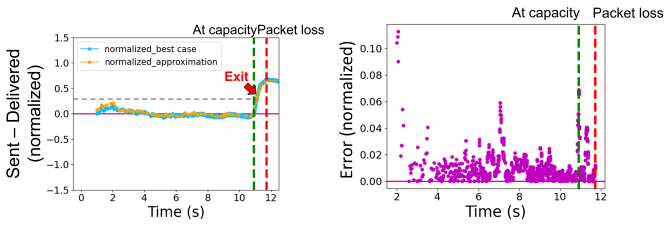
(a) Congestion window. (b) Round-trip time.

Fig. 4: GEO satellite network.



(a) Sent and delivered bytes. (b) Normalized difference in (a).

Fig. 5: GEO satellite network.



(a) Normalized best case and approximated difference.

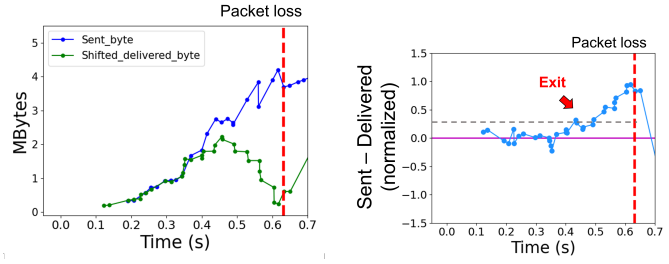
(b) Error in best case and approximated from (a).

Fig. 6: Best case and approximated - GEO satellite network.

the approximation. The two lines closely align, indicating the approximation matches well. Figure 6b shows the error between the best case and the approximated values. All the error values are relatively small, with an average error of about 2 percent.

We did the same evaluation over the Starlink satellite network, where the smaller RTTs mean the window size is only 200 milliseconds. Figure 7a depicts the approximation results, with data and axes as for Figure 5. The normalized byte difference also crosses the 0.25 threshold before the red vertical line, indicating SEARCH is successful in finding an appropriate slow start exit point. The approximation error rates (not shown due to space) average a low 4 percent.

To evaluate the effectiveness of our approach, we conducted evenly-spaced downloads over a 24-hour period for GEO and then LEO satellite links. Each run does an iperf3 download using the default TCP slow start with HyStart disabled, logging values to ascertain SEARCH performance. In total, we conducted 213 runs over the GEO satellite link and 131 runs over the LEO satellite link. Table I shows the results averaged across all runs for each link: *cg* is the average time when the congestion window reaches the link capacity, defined as where the normalized difference exceeds 0.1; *ex* is the average time when TCP exits slow start via SEARCH – i.e., the normalized difference becomes greater than 0.25; *dp* is the average time when packet loss occurs with normal slow start – i.e., if SEARCH was not used; and *hr* is the "headroom", the average time between exiting slow start via SEARCH and experiencing packet loss. The experiment results indicate SEARCH successfully detects an exit point after congestion



(a) Approx. sent and delivered.

(b) Normalized difference in (a).

Fig. 7: SEARCH – LEO satellite network.

TABLE I: SEARCH evaluation.

Link	Runs	Success (%)	cg (s)	ex (s)	dp (s)	hr (s)
GEO	213	64.4	11.0	12.4	18.1	5.7
LEO	131	52.0	0.5	0.6	0.7	0.1

but before packet loss most of the time for both GEO and LEO links. For the GEO link, the headroom is much greater than the RTT while for the LEO link it is slightly larger than the RTT, suggesting TCP with SEARCH can exit slow start without incurring packet loss.

V. Conclusion

We propose SEARCH, a novel algorithm that accurately detects a good slow start exit point via the difference in bytes sent and bytes delivered. SEARCH is server-side only, and uses a large window to account for link variance and bins to accurately approximate with low overhead. Preliminary evaluation over GEO and LEO satellite networks shows SEARCH can determine a slow start exit point after the congestion window has grown to capacity but before inducing packet loss. The effectiveness of SEARCH in challenging GEO and LEO networks suggests potential to enhance TCP performance beyond only satellites.

Our current work is to finalize our Linux kernel implementation, including sensitivity analysis of our parameter settings (e.g., bin size and exit threshold). We also plan to compare our proposed algorithm with state-of-the-art approaches, e.g., HyStart++, and over a wider range of network conditions.

References

[1] M. Furqan and B. Goswami, "Satellite Communication Networks," *Handbook of Real-Time Computing; Tian, Y.-C., Levy, DC, Eds*, 2022.

[2] B. Al Homssi, A. Al-Hourani, K. Wang, P. Conder, S. Kandeepan, J. Choi, B. Allen, and B. Moores, "Next Generation Mega Satellite Networks for Access Equality: Opportunities, Challenges, and Performance," *IEEE Communications Magazine*, vol. 60, no. 4, 2022.

[3] C. H. Park, P. Austria, Y. Kim, and J.-Y. Jo, "MPTCP Performance Simulation in Multiple LEO Satellite Environment," in *IEEE Computing and Communication Workshop and Conference (CCWC)*, 2022.

[4] S. Ha and I. Rhee, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks," in *International Workshop on Protocols for Fast Long-Distance Networks*, Manchester, UK, Mar. 2008.

[5] M. A. Kachooei, Z. Pinhan, F. Li, J. Chung, and M. Claypool, "Fixing TCP Slow Start for Slow Fat Links," in *Proceedings of the 0x16 NetDev Conference*, Oct. 2022.

[6] A. M. Jasim and G. A. Abed, "An Effective Practice to Approximating TCP Congestion Window Threshold in High Latency Connectionsut-sumi2018new," *Al-Iraqia Journal for Scientific Eng. Research*, 2022.

[7] P. Balasubramanian, Y. Huang, and M. Olson, "HyStart++: Modified Slow Start for TCP," *IETF Draft draft-balasubramanian-tcpm-hystartplusplus-03*, Apr. 2020.

[8] J. Ye, B. Huang, and X. Chen, "An Improved Algorithm to Enhance the Performance of FAST TCP Congestion Control for Personalized Healthcare Systems," *Wireless Commun. and Mobile Computing*, 2021.

[9] R. Lübben, "Forecasting TCP's Rate to Speed up Slow Start," *IEEE Open Journal of the Computer Society*, vol. 3, pp. 185–194, 2022.

[10] L. Guo and J. Y. Lee, "Stateful-TCP—A New Approach to Accelerate TCP Slow-Start," *IEEE Access*, vol. 8, pp. 195 955–195 970, 2020.

[11] S. Utsumi, S. M. S. Zabir, Y. Usuki, S. Takeda, N. Shiratori, Y. Kato, and J. Kim, "A New Analytical Model of TCP Hybla for Satellite IP Networks," *Journal of Network and Computer Applications*, 2018.