Improving TCP Slow Start Performance in Wireless Networks with SEARCH

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

Jae Chung
Viasat
Marlboro, MA, USA
jaewon.chung@viasat.com

Feng Li
Viasat
Marlboro, MA, USA
feng.li@viasat.com

Benjamin Peters
Viasat
Marlboro, MA, USA
benjamin.peters@viasat.com

Joshua Chung
Lexington Christian Academy
Lexington, MA, USA
joshuachung0906@gmail.com

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

The initial TCP slow start phase seeks to ramp up data transmission rates quickly to meet available capacity but also to exit the slow start phase before causing undue congestion. Unfortunately, the typical default TCP implementation often exits slow start too early, before capacity has been reached, causing underutilization, particularly detrimental to networks with large capacities and high delays. This study introduces a novel enhancement to TCP slow start – Slow start Exit At Right CHokepoint (SEARCH) – where the link capacity is inferred at the server based on bytes delivered compared to the expected bytes delivered, smoothed to account for link latency variation and normalized to accommodate link capacities. Empirical evaluation over geosynchronous satellite links, low-orbit satellite links, and 4G LTE links shows our approach is a substantial improvement over default TCP implementations by not exiting slow start too early, but better than traditional TCP, too, by exiting slow start before encountering packet loss.

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

I. Introduction

The Transmission Control Protocol (TCP) ensures reliable and orderly data transfer across the Internet by being able to operate over network links that span orders of magnitudes in latency and capacity. In high bandwidth-delay product (BDP) networks, such as those with a satellite link, latency and throughput vary with distance and the dynamic nature of the satellite orbits. Geosynchronous (GEO) satellites have round-trip latencies of about 600 milliseconds (ms) due to their high altitudes, while LEO satellites have round-trip latencies of about 30 ms but with latencies and capacities sensitive to atmospheric conditions and satellite locations [1]. TCP faces challenges when operating over such networks in that the high round-trip times (RTTs) and network variability can make determining link capacity difficult.

The TCP slow start mechanism is an adaptive algorithm designed to start cautiously yet rapidly increase to the available link capacity, approximately doubling the congestion window (cwnd) each RTT. Unfortunately, default implementations of TCP slow start, such as TCP Cubic with HyStart in Linux, often result in a premature exit from the slow start phase, or, if HyStart is disabled, excessive packet loss when they overshoot the link capacity. Exiting slow start too early curtails TCP's ability to capitalize on the full bandwidth potential, a setback that is particularly pronounced in satellite networks where the time to grow the congestion window is substantial. Conversely, a delayed exit overshoots the link's capacity, inducing congestion and packet loss, particularly problematic for links with large (bloated) bottleneck queues.

Thus, a primary goal for TCP connections over wireless links should be to maintain slow start congestion window growth for as long as safely possible without triggering an early transition to congestion avoidance. By doing so, the network can better achieve its full bandwidth potential, crucial for maintaining high throughput in satellite links. A secondary goal is to prevent packet loss by avoiding a late exit from slow start, causing unnecessary congestion. While Active Queue Management (AQM) techniques can assist in managing congestion by marking or dropping packets as signals to transition from slow start to congestion avoidance [2], AQM mechanisms are not universally present across all links so, finding a solution that works independently of an AQM is needed.

Strategies to determine the exit point from TCP slow start include bandwidth estimation, but this often falters in links with high variability resulting in capacity estimates that can swing wildly between too high and too low [3], [4]. TCP with HyStart [5] considers packet timing to find a slow start exit point before packet loss but is destructive for satellite networks by exiting slow start too early. HyStart++ [6], which leverages incremental changes in RTT to signal an exit, suffers similarly. TCP with Bottleneck Bandwidth and Round-trip propagation time (BBR) [7] doubles data rates during startup until the rates no longer increase by more than 25% for three RTTs. While this ramps up to capacity quickly, for many network connections, this can mean BBR exits from the slow start too late.

Our approach is to monitor the disparity between the delivered bytes in an RTT compared to what is expected based on bytes delivered in the previous RTT. Large difference between delivered bytes and expected delivered bytes is a reliable indicator that slow start is at the network chokepoint and should exit. We call our approach the *Slow start Exit At Right CHokepoint* (SEARCH) algorithm. SEARCH is grounded in the principle that during slow start, the congestion window expands by one for each acknowledgment (ACK) received, prompting the transmission of two packets and effectively doubling the delivery rate each RTT. However, when the network surpasses capacity, the delivery rate does not double as expected, signaling that slow start should exit. Specifically, the current delivered bytes should be twice the delivered bytes one RTT ago. Since capacities increase over time, SEARCH normalizes the difference based on the current delivery rate and since link latencies can vary independently of data rates (especially for wireless links), SEARCH smooths the measured delivery rates over several RTTs.

This paper provides a detailed description of SEARCH, including justification for the windowed smoothing and delivery-difference thresholds. It also details the sensitivity analysis of SEARCH algorithm parameters to provide the rationale behind the heuristic settings. Lastly, experiments with hundreds of TCP downloads over GEO, LEO, and 4G LTE links provide for an empirical comparison of SEARCH versus default TCP implementations (with and without TCP HyStart enabled).

The rest of this paper is organized as follows: Section II reviews related work, Section III describes our SEARCH approach, Section IV tunes SEARCH algorithm parameters, Section V describes experiments and analysis evaluating performance, Section VI discusses implications of our findings, Section VII mentions some limitations of the work, and Section VIII summarizes our conclusions and possible future work.

II. Related Work

Jasim et al. [4] propose a technique for approximating the TCP congestion window thresholds for high latency connections by using a packet-pair technique to estimate bandwidth. The authors perform experiments using a variety of network configurations and find that their approach can improve the performance of TCP when latencies are high. Additionally, they compare their approach to other existing methods and find it outperforms them in terms of both efficiency and accuracy.

Ye et al. [8] propose a new algorithm, Personalized FAST TCP, which improves the performance of the FAST TCP congestion control for personalized healthcare systems. The algorithm uses the number of remaining link buffers to judge exit times for slow start, designed to help the queuing delay ratio stays within the expected range and not change with variations in bandwidth or other parameters, leading to faster convergence of the system. They also propose a method for dynamically adjusting the gain parameters of the controller based on local information obtained from each connection source, allowing the system to adjust to changes in network operation states within the range of relevant protocol parameters to maintain stability. With this method, they find the FAST TCP system achieves a small queuing delay and quickly converges to the equilibrium point.

Gál er al. [9] introduce BIC (Binary Increase Congestion Control) and Hybla as solutions to improve TCP's performance in high-latency networks. Hybla modifies the slow start and congestion avoidance phases of NewReno to reduce the dependency on RTT, achieving RTT fairness but with amplified reactions in higher RTT flows. BIC aims to approximate the optimal congestion window size using a binary search approach, though may exhibit RTT fairness issues comparable to Reno's. Unlike these loss-based algorithms, delay-based algorithms preemptively adjust to network conditions.

Kachooei et al. [3] present the BEST algorithm, a bandwidth estimation technique based on packet-pair measurements for determining the slow start exit point. While showing promise, BEST encounters challenges in environments with high variability in estimated bandwidth and RTT, leading to underestimation or overestimation of the available bandwidth.

Li et al. [10] present Fast Bandwidth Estimation (FBE), a solution designed to optimize the slow start phase in high-bandwidth networks like WiFi 6 and 5G. Recognizing the issue of link underutilization due to the slow ramp-up of congestion windows, FBE utilizes the initial ACK feedback to estimate link capacity without sending extra probe packets. By refining ACK intervals to reflect true link rates and dynamically adjusting the congestion window based on driver queue feedback, FBE attempts to address the inaccuracies in bandwidth estimation that are especially prevalent in variable wireless links. Comparative experiments show FBE outperforms traditional slow start methods, cutting down convergence time and improving short flow completion times against well-known algorithms such as CUBIC and BBR.

Kasoro er al. [11] propose ABCSS, a method that combines the strengths of Appropriate Byte Counting (ABC) and the Slow Start (SS) algorithm. This approach is designed to increase the congestion window more effectively than the traditional slow start, aiming to address the invariant congestion window during initial round trips that leads to TCP burst issues and potential buffer overflows.

While the above approaches are all alternatives, and in some sense improvements, to traditional slow start, unlike our approach they do not focus on the fundamental problem of avoiding an early exit to slow start before the link capacity is reached while still exiting before inducing unnecessary congestion.

III. Methodology

Our primary objective is to improve TCP performance by exiting slow start in a timely manner – after the capacity point is reached but before inducing packet loss. We introduce an additional algorithm to slow start that identifies the congestion point by comparing the delivered bytes to the expected delivered bytes, with the understanding that delivered bytes should approximately double each RTT until capacity is reached. The

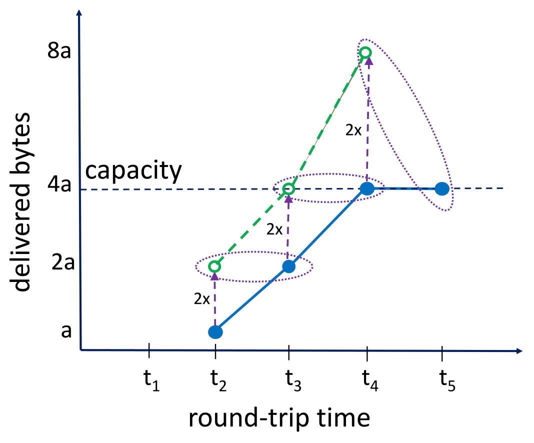


Fig. 1: TCP acknowledged delivery of data over time.

algorithm also examines delivered bytes smoothed over several RTTs in order to be resilient to underlying link variance, and normalizes delivered bytes to adapt to a range of link capacities.

During TCP slow start, each acknowledgment received increases the congestion window by one, triggering transmission of two packets and effectively doubling the window size – and consequently the sent bytes – each RTT. Thus, in uncongested network conditions, the number of bytes acknowledged as delivered in a given RTT is about twice the number of bytes delivered in the previous RTT.

Figure 1 illustrates this concept, with the number of delivered bytes shown on the y-axis according to the RTTs on the x-axis. Here, the delivered bytes (the blue line) can be compared with twice the number of bytes delivered one RTT earlier (the green line). If the number of bytes delivered during the previous RTT is equal to twice the number of bytes delivered during the current RTT – such as happens at t_3 where the delivered bytes have doubled since t_2, and then again at t_4 where the delivered bytes have doubled since t_3 – then the link capacity has not been reached. This observation aligns with the behavior of TCP slow start, where the congestion window is increased by one for each delivered byte, resulting in a doubling of the sent bytes each RTT. However, at time t_5 the number of bytes delivered is not double the bytes delivered since time t_4, indicating the congestion point has been reached.

This concept – that delivery rates should double each RTT until capacity is reached — is core to our *Slow start Exit At Right CHokepoint* (SEARCH) algorithm. Our initial version of SEARCH [12] (SEARCH 1.0), recorded the sent and delivered bytes separately for comparison, whereas here in SEARCH 2.0 we refine the algorithm to use only the delivered bytes thus reducing memory use by about 50%. This reduction is significant for TCP servers which must store slow start state per-flow so reducing a flow's memory footprint is desirable. In SEARCH 2.0, when the bytes delivered one RTT prior is half the bytes delivered now, the bitrate is not yet at capacity, whereas when the bytes delivered prior are less than half the bytes delivered now, the link capacity has been reached and TCP exits slow start.

One challenge in monitoring delivered data across multiple RTTs is latency variability for some links. An unstable latency in the absence of congestion – common in some wireless links – can cause RTTs to differ over time even when the network is not at capacity. This variability complicates the direct comparison of data because a lower latency can lead to false positives in that the delivered bytes one RTT ago seem too low, thus appearing to be at the link capacity when it is not. For instance, Figure 2 shows the RTT times for a TCP connection over a GEO satellite link. For this connection, capacity is not reached until after 12 seconds, yet the RTTs vary considerably due to the scheduling of TCP acknowledgments on the uplink. This type of variability makes accurate comparison of delivered amounts difficult.

To counteract link latency variability, SEARCH tracks delivered data over several RTTs in a sliding window providing a more stable basis for comparison. Since tracking individual packet delivery times is prohibitive in terms of memory use, the data within the sliding window is segmented into bins representing small, fixed time periods. The window then slides over bin-by-bin, rather than sliding every ACK packet, reducing both the computational load (SEARCH only triggers at the bin boundary) and memory requirements.

The SEARCH algorithm is shown in Algorithm 1. In Lines 1-7, SEARCH initializes with predefined parameters for window size, number of bins, bin duration, and threshold. The window size (WINDOW_SIZE) is set to 3.5 times the initial RTT to smooth out variation in link latency. Ten bins are used to approximate the delivered rates over the window size, with an additional 10 (EXTRA BINS) bins stored (NUM_BINS) in order to allow comparison of the current delivered bytes to the previously delivered bytes one RTT earlier. The W constant is used as the actual window size for computing delivered bytes. The bin duration (BIN_DURATION) is calculated by dividing the window size by the number of bins. The threshold (THRESH), set at 0.35, is the upper bound of the permissible difference between the previously delivered bytes and the current delivered bytes (normalized) above which slow start exits.

After one-time initialization in Lines 8-11, the algorithm

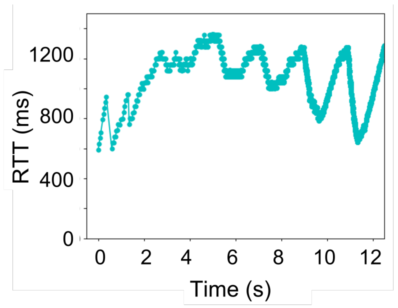


Fig. 2: RTTs for a GEO satellite link.

operates upon the arrival of each acknowledgment. On line 13, when the current time (**now**) has passed the end time of the last bin, it updates the bin boundary (Lines 14-16). Line 17 computes the index (prev) for the previous bins. Line 18 checks if there has been at least a window's-worth of data previously delivered – if so, SEARCH can begin checking to see if slow start should exit in Lines 20-26. Line 20 computes the bytes delivered for the current window by summing up the corresponding bins, and Line 21 does the same but for the window one RTT ago. Line 22 computes the normalized difference in the bytes delivered one RTT ago and the bytes delivered currently. Line 23 compares this difference to the threshold (THRESH) and, if it is greater, exits slow start by setting `ssthresh` to `cwnd`. Lastly, Line 29 adds the currently delivered bytes (in the acknowledgment packet) to the tally in the latest bin.

Note, if the value of prev (computed on Line 17) falls within a bin (i.e., between two bin boundaries), the computation on Line 21 uses an appropriate proportion of the bin at prev-W (i.e., bin[prev-W]) and prev (i.e., bin[prev]) in computing prev_delv. This adjustment is not shown in the Algorithm 1 to keep it readable.

IV. SEARCH Parameters

This section provides the rationale for SEARCH parameters: window size (Section IV-A), threshold values (Section IV-B), and number of bins (Section IV-C).

A. Window Size

The SEARCH window smooths over baseline RTT fluctuations in a connection. The window size must be large enough to encapsulate meaningful link variation, yet small in order to allow SEARCH to respond near when slow start reaches link capacity. In order to determine an appropriate window size, we analyzed RTT variation over time for GEO, LEO, and 4G LTE links for TCP during slow start.

Figure 3a shows the RTT observed over time averaged over 77 samples from a GEO network during TCP slow start preceding congestion. The SEARCH window size should be large enough to capture the observed periodic oscillations in the RTT values. In order to determine the oscillation period, we use a Fast Fourier Transform (FFT) [13] to convert the RTT values from the time domain to the frequency domain. The results are depicted in Figure 3b. The x-axis is the frequency (in Hz) and the y-axis is the magnitude of the signal. The primary peak is at 0.5 Hz, meaning there is a large, periodic cycle that occurs about every 2 seconds. Given the minimum RTT for a GEO connection of about 600 ms, this means the cycle occurs about every $2/0.6 \approx 3.33$ RTTs. Thus, a window size of about 3.5 times the minimum RTT should smooth out the latency variation for this type of link.

Figure 3c and Figure 3d show similar analysis for a LEO satellite link, again using RTT values for 77 TCP transfers during slow start prior to congestion. While the RTT periodicity for the LEO link is not as pronounced as in the GEO link, the FFT still has a dominant peak at 10 Hz, so a period of about 0.1 seconds. With LEO's minimum RTT of about 30 ms, the period is $0.1/0.03 \approx 3.33$ RTTs. Thus, a window size of about 3.5 times the minimum RTT should smooth out the latency variation for this type of link, too.

Algorithm 1
SEARCH 2.0: Slow start Exit At Right CHokepoint.

```
 1: Parameters:
 2: WINDOW_SIZE = Initial_RTT × 3.5
 3: W = 10
 4: EXTRA_BINS = 10
 5: NUM_BINS = W + EXTRA_BINS
 6: BIN_DURATION = WINDOW_SIZE / W
 7: THRESH = 0.35

 8: Initialization:
 9: bin[NUM_BINS]
10: curr = 0
11: bin_end = now + BIN_DURATION

12: Each acknowledgement:
13: if (now > bin_end) then
14:     bin_end += BIN_DURATION
15:     curr += 1
16:     bin[curr mod NUM_BINS] = 0

17:     prev = curr - (RTT / BIN_DURATION)
18:     if (prev ≥ W) and (curr - prev) ≤ EXTRA_BINS then
19:         // Check if SEARCH should exit
20:         curr_delv = $\sum_{\text{curr-W}}^{\text{curr}}$ bin[i mod NUM_BINS]
21:         prev_delv = $\sum_{\text{prev-W}}^{\text{prev}}$ bin[i mod NUM_BINS]
22:         norm_diff = $\frac{2 \cdot \text{prev\_delv} - \text{curr\_delv}}{2 \cdot \text{prev\_delv}}$
23:         if (norm_diff ≥ THRESH) then
24:             // Exit slow start
25:             set ssthresh to cwnd
26:         end if
27:     end if
28: end if
29: bin[curr mod NUM_BINS] += bytes_delivered
```

Figure 3e and Figure 3f show analysis of a 4G LTE network for RTT values collected across 55 TCP transfers during slow start prior to congestion. Similarly to the LEO link, the LTE network does not have a strong RTT periodicity. It has a dominant peak at 6 Hz, with a period of about 0.17 seconds. With the minimum RTT of the LTE network about 60 ms, this means a window size of at least $0.17/0.06 \approx 2.8$ times the minimum RTT is needed. A SEARCH default of 3.5 times the minimum RTT exceeds this, so should smooth out the variance for this type of link as well.

B. Threshold

The threshold determines when the difference between the bytes delivered currently and the bytes delivered during the

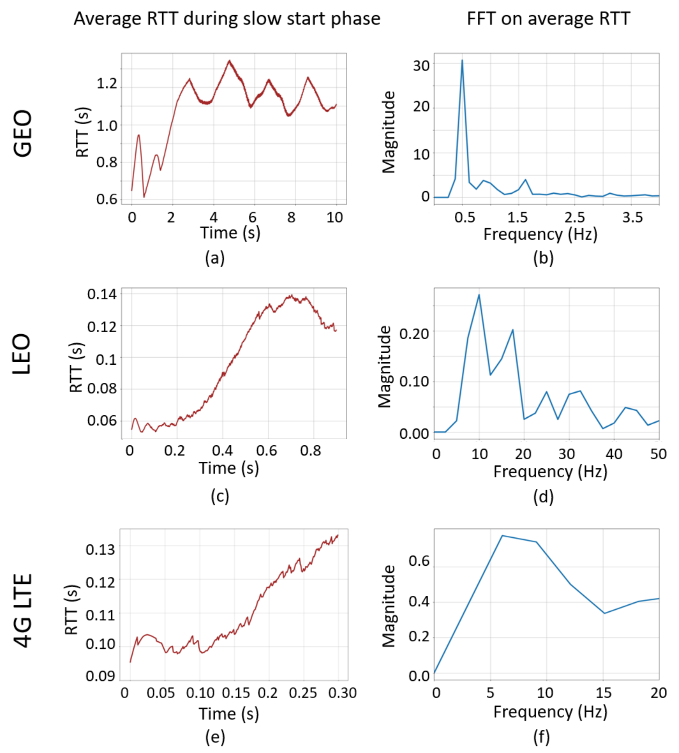


Fig. 3: Average RTT and FFT over GEO, LEO, and 4G LTE.

previous RTT is great enough to exit the slow start phase. A small threshold is desirable to exit slow start close to the 'at capacity' point, but the threshold must be large enough not to trigger an exit from slow start prematurely due to noise in the measurements.

During slow start, the congestion window doubles each RTT. In ideal conditions and with an initial cwnd of 1, this results in a sequence of delivered bytes that follows a doubling pattern $(1, 2, 4, 8, 16, \ldots)$. Once the link capacity is reached, the delivered bytes each RTT cannot increase despite cwnd growth.

For example, consider a window that is $4x$ the size of the RTT. After 5 RTTs, the current delivered window comprises $2, 4, 8, 16$, while the previous delivered window is $1, 2, 4, 8$. The current delivered bytes is 30, exactly double the bytes delivered in the previous window. Thus, SEARCH would compute the normalized difference (Line 22 in Algorithm 1) as zero.

Once the cwnd ramps up to meet full link capacity, the delivered bytes plateau. Continuing the example, if the link capacity is reached when cwnd is 16, the delivered bytes growth would be $1, 2, 4, 8, 16, 16$. The current delivered window is $4 + 8 + 16 + 16 = 44$, while the previously delivered window is $2+4+8+16 = 30$. Here, the normalized difference between $2\times$ the previously delivered window and the current window is about $(60-44)/60 = 0.27$. After 5 more RTTs, the previous delivered and current delivered bytes would both be $16 + 16 + 16 + 16 = 64$ and the normalized difference would be $(128 - 64)/64 = 0.5$.

To generalize this relationship, Figure 4 shows the congestion window as a function of time. The blue curve has an exponential growth (doubling) during slow start and the window. The shaded region labeled 'x' is the window in the initial growth period. At window 'z', capacity has been reached and sustained long enough that the window is full of 'at capacity' values, as in the above example. At window 'y', the left side of the window has exponential growth while the right side has plateaued at capacity.

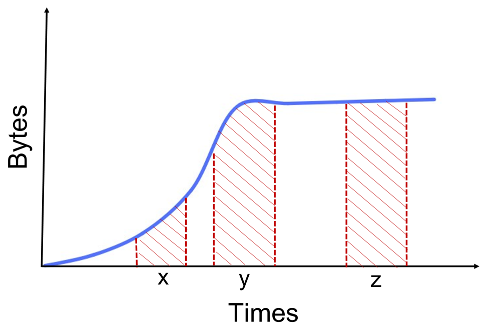


Fig. 4: Window position based on congestion point: 'x' is before capacity and window growth doubles each RTT, 'y' is at capacity where the left side of the window has doubled growth and the right side has plateaued, and 'z' has plateaued for the whole window.

The theoretical underpinnings of this behavior can be quantified by integrating the area under the congestion window curve. During exponential growth (e.g., Figure 4 window 'x'), the area in a window from RTT round a to RTT round b is:

$$\int_a^b 2^x\,dx = \left[\frac{2^x}{\ln(2)}\right]_a^b = \frac{2^b}{\ln(2)} - \frac{2^a}{\ln(2)}$$

Once at capacity is reached (e.g., Figure 4 window 'z'), say by RTT round c, the area in a window from RTT round d to RTT round e is:

$$\int_d^e 2^c\,dx = [2^c \cdot x]_d^e = 2^c \cdot e - 2^c \cdot d = 2^c \cdot (e - d)$$

If the window encompasses both exponential growth and constant, at capacity rate (e.g., Figure 4 window 'y'), the area is computed in pieces, exponential up to the 'at capacity point' and then constant to the end of the window.

If r is the current round measured in RTTs, c is the round where capacity has been reached, and w is the size of the window in RTTs, then if capacity is reached after the current round (e.g., Figure 4 window 'x'), $r < c$:

$$\overbrace{2^x, 2^{x+1}, \ldots, 2^{r-1}, \underbrace{2^r}_{r}}^{w} \quad \underbrace{\ldots}_{\text{c after window}}$$

So:

$$\begin{aligned} \text{prev\_delv} &= \left[\frac{2^x}{\ln(2)}\right]_{r-1-w}^{r-1} \\ \text{curr\_delv} &= \left[\frac{2^x}{\ln(2)}\right]_{r-w}^{r} \end{aligned} \qquad (1)$$

If capacity has been reached before the window (e.g., Figure 4 window 'z'), $c < (r - w)$:

$$\overbrace{, 2^c, 2^c, \ldots, 2^c, \underbrace{2^c}_{r}}^{w}$$

$\uparrow$
c before window

So:

$$\begin{aligned} \text{prev\_delv} &= 2^c \cdot ((r-1) - (r-1-w)) \\ \text{curr\_delv} &= 2^c \cdot ((r) - (r-w)) \end{aligned} \tag{2}$$

If congestion occurs within the window (e.g., Figure 4 window 'y'), $(r - w) < c < r$:

$$\overbrace{2^x, 2^{x+1}, \ldots\ldots \quad , 2^c, \underbrace{2^c}_{r}}^{w}$$

$\uparrow$
c within window

So:

$$\begin{aligned} \text{prev\_delv} &= \left[\frac{2^x}{\ln(2)}\right]_{r-1-w}^{c} + 2^c \cdot (r-1-c) \\ \text{curr\_delv} &= \left[\frac{2^x}{\ln(2)}\right]_{r-w}^{c} + 2^c \cdot (r-c) \end{aligned} \tag{3}$$

With closed-form equations for both the current delivered bytes (curr_delv) and the previously delivered bytes (prev_delv), the normalized difference can be computed based on the RTT round relative to the 'at capacity' round. Figure 5 shows this relationship. The x-axis is the RTT round relative to the 'at capacity' round. So, for example, $x = 2$ means 2 RTTs after capacity has been reached and $x = -1$ means one RTT before capacity will be reached. The y-axis is the normalized difference in current delivered bytes and previously delivered bytes (Line 22 in Algorithm 1). For illustration, the trendline shows three functions, each representing a different window size (in RTTs): 2, 3.5, and 5.

From the graph, as expected, before capacity ($x < 0$) the normalized difference is 0. When capacity is reached, the normalized difference climbs sharply during the first RTT, leveling off slightly and reaching a maximum of 0.5 when the window is full of 'at capacity' values. The initial growth in the normalized difference values once $x > 0$ is approximately the same regardless of the window size.

While SEARCH seeks to detect the 'at capacity' point as soon as possible after reaching it, it must also avoid premature exit in the case of noise on the link. The 0.35 threshold value chosen does this and can be detected with 2 RTTs of reaching capacity.

C. Number of Bins

Dividing the delivered byte window into bins reduces the server's memory load by aggregating data into manageable segments instead of tracking each packet. This approach simplifies data handling and minimizes the frequency of window updates, enhancing server efficiency. However, more bins provide more fidelity to actual delivered byte totals and allow SEARCH to make decisions (i.e., compute if it should exit slow start) more often, but require more memory for each flow. The sensitivity analysis conducted here aims to identify the impact of the number of bins used by SEARCH and the ability to exit slow start in a timely fashion.

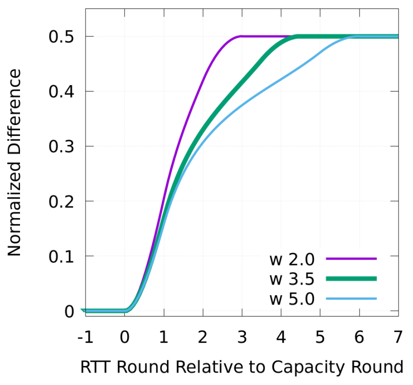


Fig. 5: Normalized difference versus RTT round. The normalized difference must exceed the SEARCH threshold for slow start to exit.

Using a window size of 3.5× the minimum RTT and a threshold of 0.35, we vary the number of bins from 5 to 40 and observe the impact on SEARCH's performance over 77 GEO TCP downloads, 77 LEO TCP downloads, and 55 4G LTE downloads. Figure 6 depicts the results, with one graph for each network type: GEO, LEO, and 4G LTE. The x-axis for each graph is the number of bins and the y-axis is a percentage. For each bin value, the percentage of times SEARCH exited slow start too early (before capacity was reached), too late (packet drops were encountered), or at the chokepoint is computed and plotted as the y-value.

For the GEO link (Figure 6a), the trendlines are all flat which indicates SEARCH performance is relatively stable no matter what bin size is chosen. For the LEO link (Figure 6b) and 4G LTE link (Figure 6c), the trendlines are still mostly flat, albeit the LEO link showing a slight upward trend early and at the chokepoint with number of bins. While ideally, SEARCH would maximize the at chokepoint values, it also seeks to avoid early slow start exits since these are especially detrimental to high BDP links. For all graphs, a bin size of 10 minimizes early exits while having an at chokepoint percentage that is close to the maximum.

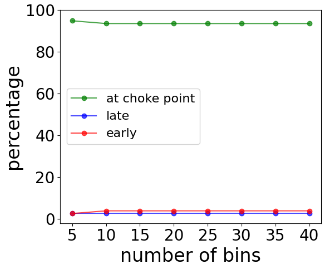
V. Evaluation

This section describes our measurement testbed (Section V-A), a heuristic to determine the 'at capacity' time (Section V-B), and evaluation results (Section V-C).

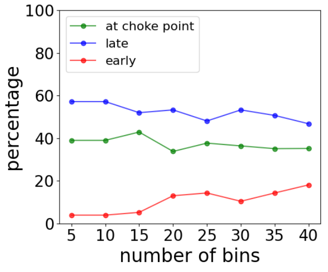
A. Measurement Testbed

We set up a measurement testbed to evaluate SEARCH which mirrors common Internet user scenarios wherein the wireless link is the "last mile" of connectivity to the client which is downloading from an Internet-connected server.

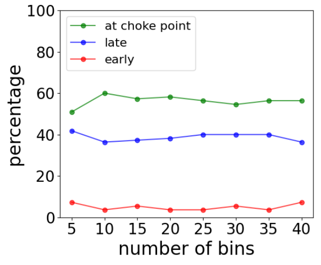
Our testbed includes GEO and LEO satellite networks, depicted in Figure 7. The client is a PC with an Intel i7-5820K



(a) GEO



(b) LEO



(c) 4G LTE

Fig. 6: Bin sensitivity analysis.

CPU @ 3.30GHz and 32 GB RAM running Ubuntu-20.04 and Linux kernel version 5.4.0. The server is a PC with an Intel i5-8500 CPU @ 3.00GHz and 8 GB RAM running Mint-20.3 and Linux kernel version 5.10.79. The server connects to our University LAN via Gb/s Ethernet. The campus network is connected to the Internet via several 10 Gb/s links, all throttled to 1 Gb/s.

The client connects to a Viasat GEO satellite terminal (with a dish and modem) via a Gb/s Ethernet connection. The client's downstream Viasat service plan provides a peak data rate of 144 Mb/s. The terminal communicates through a Ka-band outdoor antenna (RF amplifier, up/down converter, reflector, and feed) through the Viasat 2 satellite[1] 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 for each client can grow up to 36 MBytes. Queue lengths are controlled at the gateway by Active Queue Management (AQM) that randomly drops 25% of incoming packets when the queue is over half of the limit (i.e., 18 MBytes). The GEO performance-enhancing proxy (PEP) is deliberately deactivated to simulate conditions where encryption or other constraints preclude PEP usage.

The client connects to the LEO satellite through an electronic phased array outdoor antenna. The LEO network delivers a peak downlink rate of approximately 100 Mb/s and has a minimum RTT of about 30 ms.

Our 4G LTE testbed has a server with an Intel Pentium Silver N6005 CPU and 16 GB of RAM running Ubuntu version 22.04.1 and Linux kernel version 6.2.0. The client is an

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

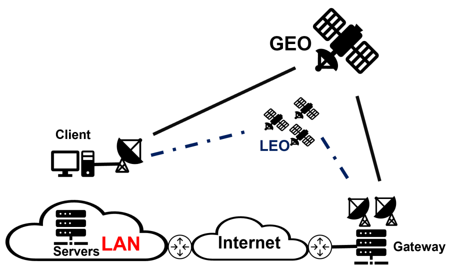


Fig. 7: GEO and LEO satellite measurement testbed.

Intel Celeron 2957U CPU and 4 GB of RAM running Ubuntu-22.04.1 and Linux kernel version 6.2.0. The client connects to the Internet via a 4G LTE "hotspot" connection over an iPhone XS through the Verizon network.

The clients and server are instrumented to log delivered bytes, RTTs, and congestion window sizes. The standard test protocol uses TCP Cubic – Linux's default TCP congestion control algorithm – with HyStart disabled, unless otherwise noted.

B. *Determining when TCP has Reached Link Capacity*

SEARCH's goal is to exit slow start after the congestion window has grown large enough to reach link capacity but before inducing packet loss. In order to determine the 'at capacity' point for an unknown link, we deploy a heuristic that uses the one-way latency offset recorded via Wireshark on the client to determine when the downlink has been saturated. At the start of data transmission, the latency offset is low, indicating the link before congestion. As the transmission continues, the latency offset increases once the downlink is saturated – i.e., the capacity limit has been reached and subsequent congestion window growth results in queuing thus increasing latency. Using this idea, our heuristic for determining the 'at capacity point' from a previously-gathered Wireshark trace is:

1) Determine when the latency offset first surpasses twice the minimum RTT. Reaching this level suggests that the link capacity has been reached sometime in the past.
2) Trace backward from this time point to find when the latency offset falls below one-half the minimum RTT. This is used as the 'at capacity' point.

Figure 8 shows an example for the latency offset values recorded at the client from Wireshark for a TCP download from a server over a GEO satellite link with a minimum RTT of 600 ms. The x-axis is the time in seconds and the y-axis is the latency offset in milliseconds with the blue curved line the latency offset values. For the first 10 seconds, the latency offsets are low, well under one-half the minimum RTT. After time 10 s, the latency offset values rise fairly continuously until past time 25 s. The time of the first packet loss is determined via 3 duplicated ACKs in the Wireshark trace at about time 17 s. The latency offset first surpasses 2$\times$ the minimum RTT (i.e., 1200 ms) at about time 12 s, and the earlier latency offsets are examined sequentially in reverse from here (i.e., prior to time 12) to determine where the offset drops below one-half the minimum RTT (i.e., 300 ms). In this example, this happens at about 10.5 s. This is used as the 'at capacity' point for this transfer.

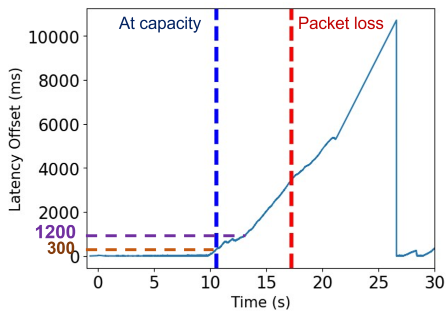


Fig. 8: Determining the 'at capacity' point using the latency offset.

C. Results

SEARCH parameters are configured based on analysis in Section IV: window size 3.5× the initial RTT, 10 bins, and threshold 0.35. The 'at capacity' point during a download is determined using the heuristics described in Section V-B and requires two thresholds: twice the minimum RTT and half the minimum RTT. Table I displays these thresholds for GEO, LEO, and 4G LTE networks.

TABLE I: Thresholds for finding the 'at capacity' point.

Network	First (ms)	Second (ms)
GEO	1200	300
LEO	60	15
4G LTE	116	28

Figure 9 shows a representative example of SEARCH performance for a single download on each link: GEO, LEO, and 4G LTE. For all graphs, the x-axis is the time (in seconds) since the download started.

The left graphs show the data transferred in Mbytes. The green line shows the current delivered bytes and the blue line shows twice the previous delivered bytes. The vertical dashed red lines indicate the first time a packet is lost. In all three graphs, the green line (current bytes) and the blue line (previous bytes) start near each other at the beginning of the download but then diverge before packet loss, where the current delivered bytes do not match (twice) the previous delivered bytes suggesting capacity has been reached.

The right graphs show the normalized difference (Line 22 in Algorithm 1), depicted as a light blue trendline. The vertical blue dashed lines indicate when capacity is reached (based on our heuristic in Section V-B), the vertical green dashed lines indicate when SEARCH exits slow start, and the red vertical dashed lines indicate the first packet loss. For all three runs, SEARCH is effective at exiting slow start after capacity has been reached but before packet loss occurs. Comparing the three runs, the GEO link has a much clearer signal that capacity has been reached in that there is less noise (variation) in the normalized difference measurements compared to the LEO and 4G LTE links. This suggests that while a lower threshold value could benefit the GEO link, the inherent noise in the measurements for the LEO and 4G LTE links could cause a premature exit from slow start with a lower threshold.

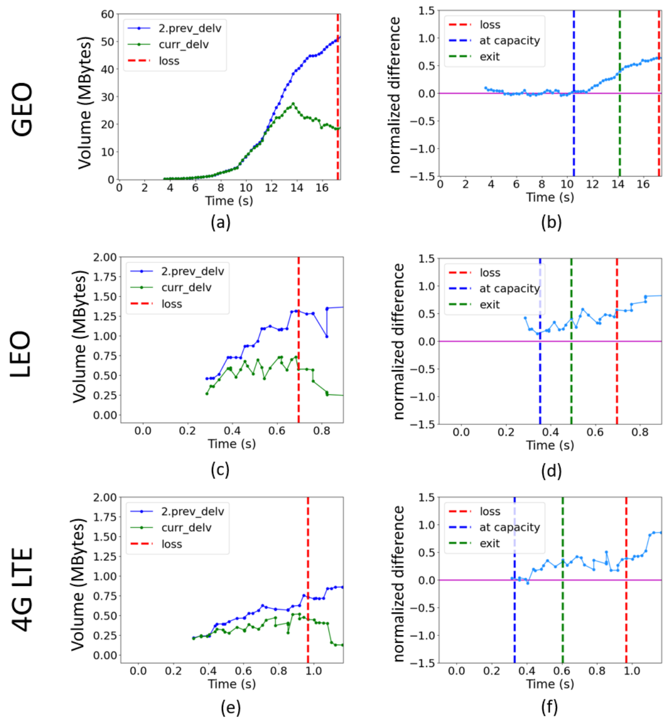


Fig. 9: Bytes delivered and normalized difference over GEO, LEO, and 4G LTE links.

To fully evaluate SEARCH, we implemented periodic downloads over a full day for both GEO and LEO satellite connections and multiple time-spaced downloads on the LTE network. Each session had an iperf3 download from the server to the client using standard TCP slow start with HyStart disabled, with the server kernel logging data to evaluate SEARCH performance. We completed 77 runs each on the GEO and LEO satellite links and 55 runs on the LTE network. The results are summarized in Table II: 'capacity' is the time for the congestion window to hit the link's capacity, 'loss' is the time for the first packet loss, and 'exit' is the time when slow start exits. All values are shown as means with standard deviations in parentheses. The 'Timely' column is the percentage of runs that did not exit prematurely, before capacity was reached.

The results from these tests suggest that SEARCH reliably determines the exit point following congestion for all link types. The average slow start exit point is after capacity is reached but before packet loss. For all links, for over 96% of the connections, SEARCH did not exit slow start too early.

In our comparative analysis presented in Table III, we compare the performance of TCP with SEARCH compared to TCP with HyStart on (the Linux default) and TCP with

TABLE II: SEARCH evaluation.

Network	Runs	Capacity (s)	Loss (s)	Exit (s)	Timely (%)
GEO	77	11.8 (2.6)	22.9 (8.7)	13.7 (1.7)	96.1
LEO	77	0.4 (0.1)	0.6 (0.1)	0.5 (0.1)	96.1
4G LTE	55	0.3 (0.2)	1.9 (1.8)	0.7 (0.6)	96.4

HyStart off (i.e., traditional TCP). 'Capacity' is the time for the congestion window to hit the link's capacity, 'Exit' is the time for first packet loss, 'Early Exit' is the percentage of runs that exit slow start before capacity is reached, 'Late Exit' is the percentage of runs that exit slow start only upon packet loss, 'Choke Point Exit' is the percentage of runs that exit after capacity is reached but before packet loss. Times are shown in seconds, averaged over all runs with the standard deviation in parentheses.

Note, we do not have 'Capacity' times for SEARCH since our implementation only logs kernel values but does not actually trigger slow start exit. However, since there are very few 'Early Exit' values for SEARCH, its 'at capacity' times would be similar to those for HyStart off. Also, for the LEO runs with HyStart on, TCP exited slow start immediately and the data ramped up so slowly that our heuristic was only able to detect the 'at capacity' time about 10% of the time. Hence, our estimates for Capacity, Early Exit, Late Exit, and Choke Point Exit are approximated.

When HyStart is off, the capacity limit is reached the quickest but TCP always overshoots the 'at capacity' point, exiting slow start only when it encounters packet loss. When HyStart is on, over GEO and 4G LTE, TCP exits slow start extremely early (always before capacity is reached) which, in turn, causes the capacity limit to be reached much later than when HyStart is off. With SEARCH, over GEO and 4G LTE, TCP rarely exits early, reaches capacity limits about as fast as TCP with HyStart off, and usually exits at the chokepoint. With SEARCH, over LEO, TCP exits after the capacity limit on average and rarely early and improves upon the late exits by about 43% versus TCP with HyStart off.

TABLE III: SEARCH, HyStart enabled, HyStart disabled over GEO, LEO, and LTE links.

Network	Slow Start Method	Capacity (s)	Exit (s)	Early Exit (%)	Late Exit(%)	Choke Point Exit (%)
GEO	SEARCH	–	13.7 (1.7)	3.9	2.6	93.5
	HyStart on	23.1 (1.9)	1.3 (0.1)	100	0	0
	HyStart off	11.8 (2.6)	22.9 (8.7)	0	100	0
LEO	SEARCH	–	0.5 (0.1)	3.9	57.1	39.0
	HyStart on	5.0	0.1 (0.1)	100	0	0
	HyStart off	0.4 (0.1)	0.6 (0.1)	0	100	0
4G LTE	SEARCH	–	0.7 (0.6)	3.6	36.4	60.0
	HyStart on	2.1 (1.0)	0.1 (0.1)	100	0	0
	HyStart off	0.3 (0.2)	1.9 (1.8)	0	100	0

VI. Discussion

Traditional TCP (without HyStart) exits slow start after capacity is reached by persisting until packets are loss. HyStart had sought to improve upon this by using packet timing to infer capacity is reached before packet loss. Unfortunately, TCP Cubic with HyStart enabled – the Linux default – always exits slow start prematurely before capacity is reached on GEO, LEO and 4G LTE wireless links. This is especially problematic for GEO links that have a high bandwidth-delay product (BDP) since any underutilization will persist and be pronounced. SEARCH shows improvements over HyStart, rarely exiting TCP slow start early over those same links, and SEARCH shows improvements over traditional TCP, too, by usually exiting slow start before packet loss occurs. This is especially promising since GEO, LEO and 4G LTE wireless links are some of the most challenging in terms of round-trip times, capacities and variances for both over time.

SEARCH is a server-side only approach meaning it requires only modifications to the TCP sender and not to the TCP receiver. This allows for incremental deployment in that only the server endpoint needs to be updated for downloads from that server to benefit from SEARCH.

While SEARCH requires some additional per-flow state, the amount of memory needed is modest – about 100 extra bytes per flow. The additional processing required by SEARCH is similarly modest, with only a few extra computations each time an acknowledgement arrives (updating the byte-delivered total) and the SEARCH exit condition check only run at the bin boundary (about twice for each congestion window of packets delivered.)

The SEARCH idea of computing the slow start exit point based on delivered bytes has a solid foundation in the way TCP slow start works – approximately doubling the cwnd each RTT – but can be sensitive to timing; in fast networks, the speed of packet delivery is small compared to the TCP processing potentially adding noise to the RTT shifting. Fortunately, the smoothing window – set to 3.5 times the minimum RTT – ought to help for fast networks, too, but further testing should be done.

VII. Limitations

There are three main limitations to the work:

Evaluations until now have only considered bulk downloads, where the server looks to transfer data as fast as possible. While this is perhaps the most interesting and critical in terms of bottleneck link utilization, the slow start characteristics of flows that are application-limited may differ. TCP flows that are limited not by the congestion window but by the application sending rate are a potential challenge for SEARCH. In such cases, the SEARCH window will advance without an increase in the delivered bytes, but this is not an indicator that capacity has been reached. Future work should assess – and then address – this case.

The results in this paper are created by first: 1) running experiments with an instrumented Linux kernel that logs the delivered bytes and RTT each acknowledgment, and 2) then running a Python version of SEARCH on the log files to determine how it would perform. While this version of SEARCH should perform the same as would a kernel implementation of SEARCH, our ongoing work is to do verify exactly this – evaluate a kernel implementation version of SEARCH over the same network conditions evaluated thus far.

The evaluations thus far have been for only three types of wireless last mile links – GEO, LEO and 4G LTE – with relatively uncontrolled conditions for competing traffic. While these wireless networks are among the most challenging for slow start due to the underlying variations in RTTs and capacities, there are many more network conditions to consider including, but not limited to, other wireless links (e.g., WiFi, 5g) and traffic patterns (contending and cross traffic).

VIII. Conclusion

TCP slow start is designed to ramp up to the capacity limit of a link quickly, exiting the slow start phase and entering congestion avoidance once it is sure capacity has been reached. For traditional slow start, this means exiting slow start only when encountering packet loss which can cause unnecessary queuing, retransmissions, and recovery time. TCP with HyStart is designed to avoid overshooting a link's capacity, exiting slow start before packet loss is encountered. Unfortunately, when running on some wireless links, TCP with HyStart always exits slow start prematurely causing link capacity to be reached far later than for traditional TCP.

TCP with SEARCH looks to exit slow start after capacity is reached but before packet loss. SEARCH uses the difference between bytes delivered currently and bytes delivered during the previous RTT to detect if capacity has been reached. SEARCH calculates delivered bytes over several RTTs to smooth out RTT variance and uses bins to approximate the bytes delivered over time and reduce server-side memory use.

Evaluation of hundreds of downloads of SEARCH across GEO, LEO, and 4G LTE network links compared to TCP with HyStart and TCP without HyStart shows SEARCH almost always exits after capacity has been reached but before packet loss has occurred. This results in capacity limits being reached quickly while avoiding inefficiencies caused by lost packets.

Future work includes implementing SEARCH in the Linux kernel and in an open-source, user-level QUIC library (e.g., QUICly [14]). In addition, we intend to evaluate SEARCH over a broader range of network conditions, LAN and other wireline networks, and wireless networks where the client is mobile. Other future work could compare SEARCH to other slow start approaches, such as HyStart++ [6], deployed by default in Microsoft OS, and TCP Hybla [15]. Additional future analysis is to determine the number of extra bins SEARCH needs to adequately cover shifting back when determining previously-delivered bytes – limiting the number of extra bins is desirable but so is the ability to handle highly variable RTTs. Future work also needs to modify SEARCH to handle scenarios when there are extended periods of inactivity (e.g., the application has no data to send or when server acknowledgment processing is delayed) – in such cases, SEARCH may look to freeze the delivered byte window.

References

[1] A. Raman, M. Varvello, H. Chang, N. Sastry, and Y. Zaki, "Dissecting the Performance of Satellite Network Operators," *arXiv preprint arXiv:2310.15808*, 2023.

[2] M. M. Hamdi, S. A. Rashid, M. Ismail, M. A. Altahrawi, M. F. Mansor, and M. K. AbuFoul, "Performance Evaluation of Active Queue Management Algorithms in Large Network," in *Proceedings of the IEEE 4th International Symposium on Telecommunication Technologies (ISTT)*, 2018, pp. 1–6.

[3] 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.

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

[5] S. Ha and I. Rhee, "Taming the Elephants: New TCP Slow Start," *Computer Networks*, vol. 55, no. 9, pp. 2092–2110, 2011.

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

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based Congestion Control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[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] Z. Gál, G. Kocsis, T. Tajti, and R. Tornai, "Performance Evaluation of Massively Parallel and High Speed Connectionless vs. Connection Oriented Communication Sessions," *Advances in Engineering Software*, vol. 157, p. 103010, 2021.

[10] L. Li, Y. Chen, and Z. Li, "Small Chunks can Talk: Fast Bandwidth Estimation without Filling up the Bottleneck Link," in *Proceedings of the IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. IEEE, 2023, pp. 1–10.

[11] N. Kasoro, S. Kasereka, G. Alpha, and K. Kyamakya, "ABCSS: A Novel Approach for Increasing the TCP Congestion Window in a Network," *Procedia Computer Science*, vol. 191, pp. 437–444, 2021.

[12] M. A. Kachooei, J. Chung, F. Li, B. Peters, and M. Claypool, "Search: Robust tcp slow start performance over satellite networks," in *2023 IEEE 48th Conference on Local Computer Networks (LCN)*. IEEE, 2023, pp. 1–4.

[13] H. J. Nussbaumer and H. J. Nussbaumer, *The Fast Fourier Transform*. Springer, 1982.

[14] Various contributors, "H2O/QUICly Git Repository," https://github.com/h2o/quicly/tree/master, 2023, accessed: November 30, 2023.

[15] C. Caini and R. Firrincieli, "Tcp hybla: a tcp enhancement for heterogeneous networks," *International journal of satellite communications and networking*, vol. 22, no. 5, pp. 547–566, 2004.