# Improving QUIC Slow Start Behavior in Wireless Networks with SEARCH

Amber Cronin*, Maryam Ataei Kachooei*, Jae Chung†, Feng Li†, Benjamin Peters†, Mark Claypool*
*Worcester Polytechnic Institute (Worcester, MA, USA)
†Viasat (Marlboro, MA, USA)
*{acronin, mataeikachooei}@wpi.edu, †{jaewon.chung, feng.li, benjamin.peters}@viasat.com, *claypool@cs.wpi.edu

***Abstract*—QUIC is increasingly being deployed on the Internet as an alternative to TCP. However, QUIC over satellite links faces particular challenges as high and variable round-trip times (RTTs) make it difficult to determine and then reach link capacity. Standard slow start algorithms to detect link capacity can perform poorly over satellite links, often exiting slow start too early and limiting throughput or exiting too late and causing unnecessary packet loss. The Slow start Exit At Right CHokepoint (SEARCH) algorithm aims to exit slow start after reaching link capacity but before incurring packet loss by tracking delivery rates and exiting when rates have not increased by the expected amount. SEARCH has shown benefits over traditional slow start for TCP connections but has yet to be implemented and evaluated in QUIC. This paper presents the design and implementation of SEARCH in an open-source QUIC library, with the code publicly available as a contribution. Evaluation of SEARCH over a geostationary satellite link show SEARCH successfully exits slow start before loss in the majority of cases, improving goodput compared to the baseline.**

***Index Terms*—QUIC, SEARCH, Satellite, Congestion Control**

## I. Introduction

QUIC is a relatively new transport-layer protocol, developed by Google since 2012 and standardized by IETF in 2021 [1]. QUIC implements transport control in userspace and uses UDP packets for communication between hosts. Further, QUIC encrypts all communications on the link as a fundamental part of the protocol, performing this setup in the first round-trip as compared to the 3-4 round-trips needed by TCP/TLS. QUIC utilizes multiple streams over a single connection to reduce head-of-line blocking after loss, improving performance for use in applications that use HTTP/3. QUIC has gained popularity since its public release, carrying nearly 30% of web traffic as of 2023 [2]. Studies of QUIC implementations over a wide range of network conditions are important for assessing and verifying behavior of the protocol.

Satellites in Geostationary Equatorial Orbit (GEO) have high baseline round-trip times (RTTs) of about 600 milliseconds (ms) and large bandwidths in the hundreds of megabits per second (Mb/s). These long, fat links have bandwidth-delay products (BDP) in the megabytes, compared to kilobytes for a typical low/medium latency terrestrial connection. TCP has traditionally faced challenges saturating high BDP connections due to the nature of TCP probing mechanisms [3]. High latencies impact the growth of the congestion window and high bandwidths take multiple RTTs to reach – issues which have been shown to carry over to QUIC [4] since it shares many similarities with TCP in its implementation of congestion control. QUIC implementations include IETF-defined Reno as well as a mix of CUBIC, BBR, and other less-common congestion control algorithms, mirroring the ACK-driven loss detection and congestion control implementations of TCP. Many endpoint-focused techniques for improving performance employed by TCP remain relevant for analysis with QUIC.

A typical solution for increasing the performance of TCP in satellite network environments is to implement a performance enhancing proxy (PEP) using middleboxes. These break a TCP connection into two or more parts over the terrestrial and orbital sections – known as Split TCP [5] – which allows the terrestrial hosts to see low-latency responses while the orbital transport can be implemented with a high-RTT optimized congestion control algorithm such as TCP Hybla [6]. This strategy relies on TCP's lack of encryption of headers, enabling middleboxes to split TCP connections without exposing the details of the proxy to the endpoints. Since QUIC is encrypted, this solution is no longer feasible as endpoints are cryptographically verified during the handshake, forcing each endpoint's own congestion controller to respond to the characteristics of the link without the possibility of middlebox intervention.

The slow start phase of congestion control implements the initial bandwidth-seeking phase of the connection when the congestion controllers typically double the amount of data being pushed over the link every RTT until a packet loss occurs. Over GEO links, the initial 150 MB of application data delivered over the connection takes place during slow start. On these links, when packet losses do occur, the high RTT and the amount of data in-flight (sent, but not acknowledged) impacts the recovery time. Large queues on satellite ground stations contribute to bufferbloat [7], which exacerbates the impact of packet loss by not dropping packets immediately when link capacity is reached, increasing the penalty of slow start overshoot.

To address this issue, we implement a new slow start algorithm called Slow start Exit At Right CHokepoint (SEARCH) [8] in QUIC to limit the overshoot and exit the slow start phase and enter the congestion avoidance phase before loss occurs. While SEARCH has been implemented and evaluated in TCP with good results, it has yet to be shown to be effective for QUIC. However, since SEARCH is general-

979-8-3503-5209-2/24/$31.00 ©2024 IEEE

purpose and does not rely on TCP or QUIC-specific behaviors, it is a good candidate to improve the performance of QUIC in a satellite network environment. This paper describes our SEARCH implementation in an open-source QUIC library, with further modification to SEARCH's exit from slow start for analysis. We contribute the code to the open-source library as well as evaluate the implementation's performance compared to baseline over a commercial GEO satellite link.

The rest of this paper is organized as follows: Section II discusses prior work related to this paper; Section III describes the implementation of SEARCH in QUIC and our experimental design; Section IV analyzes and summarizes the experimental results; Section V mentions some limitations of the work as well as possible future work; and Section VI summarizes our conclusions.

## II. Related Work

Endres et al. [4] evaluate the performance of multiple QUIC implementations over a simulated GEO satellite link and present a tool for further comparisons. Jaeger et al. [9] perform a similar evaluation on alternative high bitrate links, measuring endpoint to endpoint speed and efficiency with a variety of metrics.

Martin and Khademi [10] compare the performance of the BBRv1, BBRv2, and CUBIC congestion control algorithms in QUIC over a simulated GEO satellite link, and show that while BBR (both v1 and v2) perform substantially better than CUBIC, in lossy satellite environments each fails to meet fairness standards.

Custura et al. [11] study the impact of QUIC's acknowledgement frequency on throughput, including over emulated satellite links. Further, Volodina and Rathgeb [12] implement an algorithm to tune the acknowledgment frequency over time based on link characteristics, and show performance improvements over constant acknowledgment frequencies, including over emulated satellite links.

Liu et al. [13] propose Jump Start, a replacement slow start mechanism that bypasses the exponential growth phase by pacing an initial burst of data after the completion of the handshake. Jump Start calculates the speed and size of the initial transmission based on the initial handshake RTT, data to be sent, and advertised window size from the receiver. Jump Start has been added to some QUIC implementations, and Meta has demonstrated improvements to transfer times with its use [14], though their case study does not assess the algorithm's overall performance.

Caini and Firrincieli [6] propose TCP Hybla, a replacement for both slow start and congestion avoidance phases of the congestion control algorithm. Hybla modifies the CWND growth in each phase by a scale factor derived from the link RTT so that a Hybla connection can ramp up to the link capacity in about the same time as in a comparable low-RTT network, though it suffers the same chokepoint overshoot problem observed in standard slow start.

Ha and Rhee [15] propose HyStart as an addition to the standard exponential doubling of the TCP slow start algorithm and is implemented and enabled in the Linux kernel by default. TCP HyStart may perform well on terrestrial links, but can mistake the variation in RTT seen on satellite networks as congestion and exit slow start too early [16]. Balasubramanian, Huang, and Olson [17] propose HyStart++, a modification to the original HyStart algorithm that uses an RTT delay heuristic with a modification to reduce the impact of jitter causing premature exits from slow start.

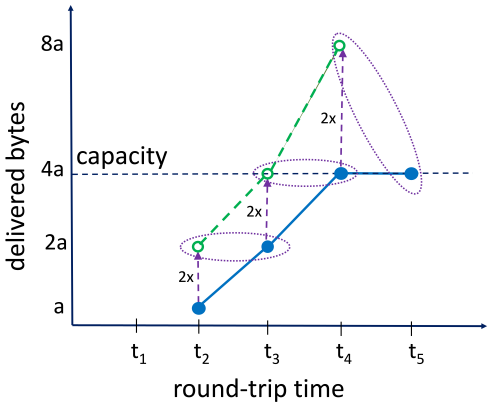


Fig. 1: Bytes marked as delivered in one RTT (solid) vs. bytes expected to be delivered in the next RTT (dashed), displaying deviation at link capacity marking the chokepoint.

Ataei Kachooei et al. [8] propose SEARCH, a modification to the slow start mechanism that tracks delivery rates with a sliding window to detect deviation between sending and delivery rates before packet loss occurs. SEARCH's benefits to performance have been shown via experiments with packet traces over GEO, Low Earth Orbit (LEO), and 4G LTE links for TCP. We implement the SEARCH algorithm in an open-source QUIC implementation.

## III. Methodology

During slow start, typical congestion control algorithms operate by approximately doubling the number of packets in flight every RTT, causing the sending rate to surpass the delivery rate that is limited by the link capacity. At its core, the SEARCH algorithm uses this idea by tracking bytes delivered over the course of the connection, and comparing the current delivery rate to the expected delivery rate from the previous RTT. Upon receiving acknowledgements, the server can verify that delivery rates match sending rates for the previous RTT until they are limited by the link capacity, as shown in Figure 1. Delivery rates for the current and previous period are computed based on a sliding window, set to a size of 3.5x the initial RTT, to reduce the impact of jitter. When these rates deviate by a measurable amount, SEARCH exits the slow start and moves to congestion avoidance, typically before packet loss occurs. SEARCH is shown in Algorithm 1 with more details and theoretical evaluation provided in Ataei Kachooei et al. [8].

One significant modification we make to the SEARCH algorithm as it was previously published is as follows. When exiting slow start at the chokepoint, SEARCH is delayed in its prediction by almost exactly two RTTs, and the algorithm has knowledge of the exact amount of deviation between the predicted delivered bytes and the actual delivered bytes. Thus, the number of bytes excess added to the congestion window (CWND) since it bypassed the BDP of the link can be computed and reduced from the CWND. This modification is added in lines 25-27 of Algorithm 1. We perform tests with and without this modification added, and compare the values of the CWND at exit with the SEARCH CWND reduction (removing the two-RTT window byte count) and the CUBIC CWND reduction.

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

1: **Parameters:**
2: WINDOW_SIZE = Initial_RTT $\times$ 3.5
3: W = 10
4: EXTRA_BINS = 15
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 $\geq$ W) and (curr - prev) $\leq$ 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 = $\dfrac{2 \cdot \text{prev\_delv} - \text{curr\_delv}}{2 \cdot \text{prev\_delv}}$
23: **if** (norm_diff $\geq$ THRESH) **then**
24: // Exit slow start
25: back = $\dfrac{\text{Initial\_RTT} \cdot 2}{\text{BIN\_DURATION}}$
26: over = $\sum_{\text{curr - back}}^{\text{curr}}$ bin[i mod NUM_BINS]
27: set `ssthresh` and `cwnd` to `(cwnd - over)`
28: **end if**
29: **end if**
30: **end if**
31: bin[curr mod NUM_BINS] += bytes_delivered

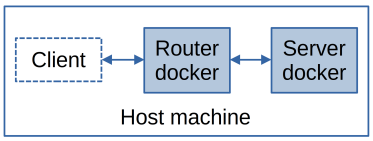


Fig. 2: Docker testbed used for verifying implementation behavior and performance.

To test SEARCH in QUIC, we select the Quicly project[1] as our QUIC implementation, developed by Fastly for the H2O webserver. Quicly is actively deployed and has been utilized for previous QUIC research. We select the Qperf project[2] as a wrapper for performing iperf3-like behavior with Quicly, and fork both projects to implement necessary features to support our testing.[3,4]

Rather than duplicating a single congestion control algorithm and adding our slow start modification into it, we refactored the Quicly code to support modular slow start implementations. We extend the congestion controller structure definition with a new field, which points to a slow start function with an identical signature as the `on_acknowledgement()` function. A configuration setting in the startup context allows dynamic selection of the slow start algorithm, and each congestion control implementation may choose to utilize the slow start function pointer, allowing non loss-based controllers to be implemented as normal. This method reduces the time required to implement alternative slow start algorithms for existing congestion controllers, and decouples the slow start phase from the congestion avoidance phase in software to better match the behavior of most congestion controllers.

By default, Quicly initializes the QUIC maximum window parameter to 16 MB. The maximum window parameter places a cap on the number of bytes permitted to be in flight on the link at the connection and stream level. This preempts the congestion controller, and a MAX_DATA frame must be transmitted from receiver to sender to grow the window and enable more data to be sent on the link. Quicly's implementation to determine when MAX_DATA frames are sent is not BDP-aware, and grows the maximum window based on percent usage of the current window. For large BDP networks, such as for a GEO link, this has an unintended consequence of limiting the CWND growth and stifling the sending rate. In our experiments, we set the maximum window parameter to 256 MB to allow the congestion controller to control the sending rate without limitations being imposed on link utilization.

To verify the implementation of SEARCH in QUIC, we create a emulated test network consisting of two Docker containers – a router and a server – on two local subnets, shown in Figure 2. The router is configured with Linux traffic control (`tc`) to implement a 150 Mbit/s link with a 300 ms one-way delay and a 36 MB queue on each outgoing

[1]https://github.com/h2o/quicly
[2]https://github.com/rbruenig/qperf
[3]https://github.com/AmberCronin/quicly
[4]https://github.com/AmberCronin/qperf

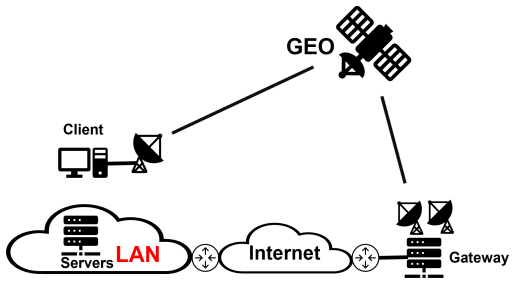


Fig. 3: GEO satellite testbed used for performance evaluation.

virtual interface to the host machine and the server container. Corresponding virtual interfaces are added to the host machine and the server container to communicate through the router container. Finally, the router is configured to route all traffic between the subnets through the virtual interfaces to emulate the satellite link. Tests using this configuration were performed on a Thinkpad T14 AMD Ryzen 7 PRO 4750U with 32 GB DDR4 RAM running Fedora 35, Docker version 20.10.17. Repeated runs were performed over this emulated link to verify the Quicly configuration and show that the QUIC flows with and without SEARCH enabled behaved as expected.

After our Quicly implementation with SEARCH was validated, we performed tests over a real satellite link to assess the SEARCH algorithm. Our satellite testbed consists of a server and a client connected over a commercial GEO satellite link, shown in Figure 3. The server connects to our University LAN via a Gb/s Ethernet. The campus network is connected to the Internet via several 10 Gb/s links, all throttled to 1 Gb/s. To address issues of network operators dropping high-bandwidth UDP traffic, the outgoing link from the server was limited to 200 Mb/s, and the network interface set to a maximum transmission unit (MTU) of 1360 bytes.

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[5] 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 performance-enhancing proxy is not available for non-TCP flows (i.e., QUIC flows).

To test the behavior and performance of the SEARCH algorithm, we record time taken to download a number of bytes for a measure of goodput with and without SEARCH enabled. This is accomplished by performing a 200 MB download,

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

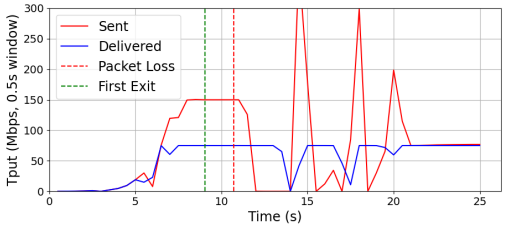


Fig. 4: Bytes sent/bytes delivered in the Docker testbed with SEARCH exit point and first packet loss marked.

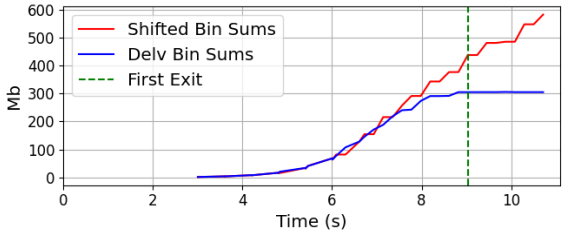


Fig. 5: Delivered and estimated delivered sums in the Docker testbed with SEARCH exit point marked, ends at packet loss.

and logging the time taken to deliver 1 MB chunks. Runs were capped at 60 seconds and alternated between SEARCH enabled and standard slow start, with a 60 second sleep period before starting the next run. Three sets of runs are analyzed: 1) 20 hours, from a Sunday night to a Monday afternoon; 2) 24 hours, from a Monday night to a Tuesday night; and 3) 24 hours, from a Wednesday night to a Thursday night.

## IV. Analysis

Figure 4 depicts the behavior of SEARCH in a quiet environment using the Docker testbed. The x-axis is the time in seconds since the download starts and the y-axis the throughput computed in 500 ms intervals. The sent bytes are captured on the server and the delivered bytes captured on the client. SEARCH (which runs on the server) infers by about time 9s (shown with the left vertical dashed line) that the link capacity has been reached and exits slow start before packet loss occurs at 11 s (shown with the right vertical dashed line). Figure 5 shows the internal workings of SEARCH where the delivered bytes confirmed by the server (delv bin sums) do not grow as expected based on the delivery rate the previous RTT (shifted bin sums), and so SEARCH exits at about time 9s (shown with the vertical dashed line).

Measurements over the real satellite network found some runs reported loss in the first RTT after the connection establishment, behavior that did not happen in the Docker testbed. While were were unable to ascertain the cause of these reported losses, they occur whether or not SEARCH is enabled and so are are excluded from further analysis. We define early (non-congestion based) loss as those with packet loss before 9 seconds and are likewise excluded. Table I shows the breakdown of the dataset. The runs listed in the "Clean" column make up the dataset considered for the remainder of the evaluation.

| | First-RTT Loss | Wireless Loss | Clean | **Total** |
|---|---|---|---|---|
| Baseline | 103 | 12 | 544 | 659 |
| SEARCH | 116 | 8 | 535 | 659 |
| **Total** | 219 | 20 | 1079 | 1318 |

TABLE I: Initial dataset. The "Clean" column is used for evaluation.

| | **Total** | 50 MB | 100 MB | 150 MB | 200 MB |
|---|---|---|---|---|---|
| Baseline | 544 | 449 | 421 | 386 | 358 |
| SEARCH | 535 | 445 | 419 | 380 | 323 |
| **Total** | 1079 | 894 | 840 | 766 | 681 |

TABLE II: Number of runs delivering indicated Mbytes.

Table II lists the number of runs that succeed in delivering the specified byte counts within the time limit.

Figure 6 compares the median, 25%, and 75% time quartiles required to deliver the specified byte count for each set of runs. Prior to delivering 130 MB, SEARCH and the baseline perform nearly identically, as SEARCH does not modify the behavior of standard slow start. At approximately 135 MB, the median run without SEARCH begins deviating from the median run with SEARCH enabled since SEARCH is able to exit slow start and avoid packet loss and degraded throughput. The greatest separation occurs at 160 MB, as the median time saved by SEARCH reaches about 3s. The lower quartile of the baseline runs deviate from the lower quartile of the SEARCH runs at 175 MB, showing that continuing to double the sending rate degrades the overall goodput.

Figure 7 depicts the median time savings (i.e., baseline minus SEARCH) by enabling SEARCH, with the difference of the quartiles depicted by the shaded area. SEARCH provides benefit by reducing the download time of files over 125 MB by up to three seconds by extending the region before packet loss occurs compared to the baseline. The percentage improvement over the baseline is shown in Figure 8, and peaks at a 14% improvement for the median 160 MB download. There is no benefit to goodput before this point, and the benefit after this particular point is amortized by the congestion avoidance phase as the connection progresses.

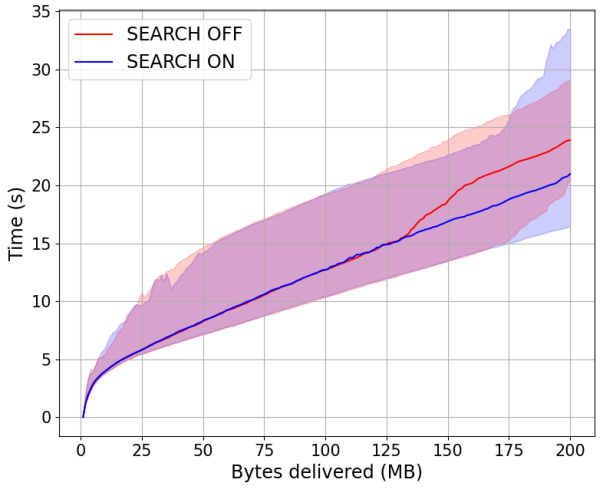


Fig. 6: Median time to download the indicated Mbytes with shading depicting the range between the 25% and 75% quartiles.

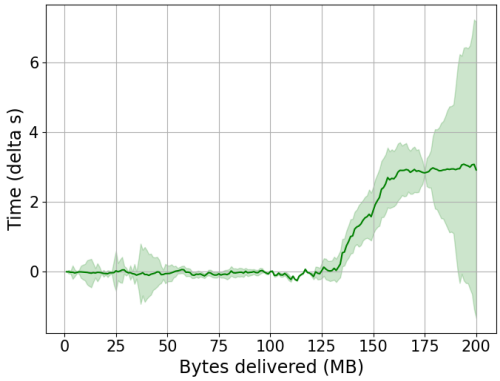


Fig. 7: Difference in median run timing, shading depicts absolute difference between quartiles.

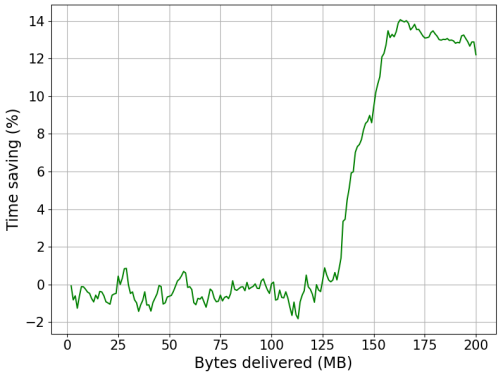


Fig. 8: Percent time savings of the median runs when SEARCH is enabled compared to the baseline performance.

Figure 9 shows the time to download 150 MB for all included runs. Fourty-five percent (45%) of runs have no significant difference in the time to download the 150 MB, an additional 40% of runs perform better with SEARCH by several seconds, and the remaining 15% (the tail of the CDF) are where the baseline performs slightly better than SEARCH. The number of runs included in this analysis are presented in Table II.

Finally, we present analysis comparing SEARCH CWND reduction to the equivalent CWND reduction if SEARCH declaring exit was treated as a packet loss, similar to exiting slow start due to explicit congestion notification (ECN) marks. When ECN marks are received, the packet containing those marks triggers a call to the congestion controller's `on_loss()` function. When using CUBIC, this reduces the congestion window by $\beta$ (CUBIC defines $\beta$=0.3). This analysis is presented in Figure 10 as a CDF. This analysis is from the third set of runs, with statistics presented in Table III. By reducing the CWND by only the amount of bytes transmitted since congestion began, SEARCH is able to maintain a higher

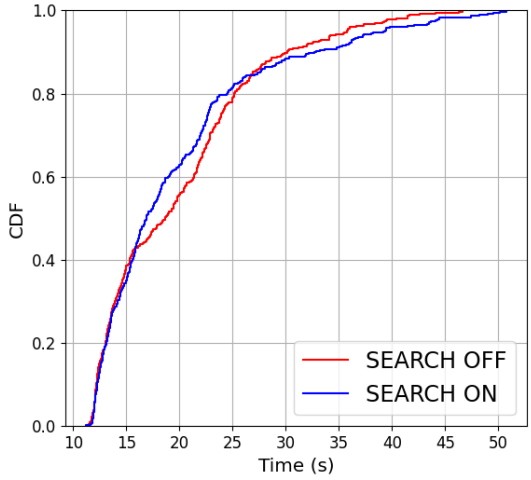


Fig. 9: CDF of time to deliver 150 MB for full dataset.

sending rate that should be closer to the link capacity than it would if the exit was declared equivalent to an ECN-triggered call to the congestion controller's loss function.

| | First-RTT Loss | Wireless Loss | Clean | **Total** |
|---|---|---|---|---|
| Baseline | 106 | 22 | 232 | 360 |
| SEARCH | 83 | 23 | 254 | 360 |
| **Total** | 189 | 45 | 486 | 720 |

TABLE III: SEARCH with CWND lowering dataset makeup.

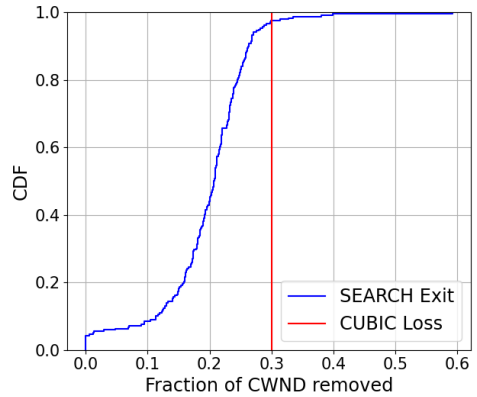


Fig. 10: Ratio of CWND reduction at SEARCH exit point to the original CWND exit (i.e., a $\beta$ multiplicative decrease).

## V. Limitations and Future Work

This work does not analyze the behavior of SEARCH in QUIC with competing flows. Future work could study how SEARCH in QUIC behaves when competing with SEARCH-enabled and baseline flows, as well as other congestion controllers (e.g., BBR), and ideally in the presence of TCP flows of various types. Similarly, SEARCH in QUIC could also be studied over other network links, including Ethernet and WiFi, mobile (4G/5G) networks, and LEO satellite networks.

Our Docker testbed findings indicate that several of Quicly's default settings degrade its performance when deployed over high-BDP networks. We recommend that Quicly (and QUIC as a whole) behavior as it relates to the maximum window parameter be studied to better apply a limited window size in high-BDP networks to allow congestion control to function properly.

## VI. Conclusion

This work presents an implementation of the SEARCH algorithm [8] in QUIC. We extended the Quicly open-source library to support generalized slow start modules and implemented SEARCH as one such module. Evaluation over an emulated GEO satellite link validates our implementation, illustrating how SEARCH detects the congestion point and exits slow start before packet loss occurs. Evaluation over a commercial GEO satellite link shows SEARCH can improve median download time by about 3 seconds (14%) compared to the baseline by limiting CWND growth when capacity is reached and delaying any packet loss due to congestion.

## References

[1] J. Iyengar and M. Thomson, "QUIC: A UDP-based multiplexed and secure transport," 2021, IETF RFC 9000.
[2] D. Belson and L. Pardue. Examining HTTP/3 usage one year on. [Online]. Available: https://blog.cloudflare.com/http3-usage-one-year-on
[3] S. Claypool, J. Chung, and M. Claypool, "Comparison of TCP congestion control performance over a satellite network," in *Passive and Active Measurement*, O. Hohlfeld, A. Lutu, and D. Levin, Eds. Springer International Publishing, 2021, pp. 499–512.
[4] S. Endres, J. Deutschmann, K.-S. Hielscher, and R. German, "Performance of QUIC implementations over geostationary satellite links." [Online]. Available: http://arxiv.org/abs/2202.08228
[5] J. Griner, J. Border, M. Kojo, Z. D. Shelby, and G. Montenegro, "Performance enhancing proxies intended to mitigate link-related degradations," 2001, IETF RFC 3135.
[6] 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.
[7] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet: Networks without effective AQM may again be vulnerable to congestion collapse." *ACM Queue*, vol. 9, no. 11, pp. 40–54, 2011.
[8] M. A. Kachooei, J. Chung, F. Li, B. Peters, and M. Claypool, "SEARCH: Robust TCP slow start performance over satellite networks," in *IEEE 48th Conference on Local Computer Networks (LCN)*, 2023, pp. 1–4.
[9] B. Jaeger, J. Zirngibl, M. Kempf, K. Ploch, and G. Carle, "QUIC on the highway: Evaluating performance on high-rate links," in *2023 IFIP Networking Conference (IFIP Networking)*, 2023, pp. 1–9.
[10] A. Martin and N. Khademi, "On the suitability of BBR congestion control for QUIC over GEO SATCOM networks," in *Proceedings of the Workshop on Applied Networking Research*, ser. ANRW '22. Association for Computing Machinery, 2022, pp. 1–8.
[11] A. Custura, T. Jones, R. Secchi, and G. Fairhurst, "Reducing the acknowledgement frequency in IETF QUIC," *International Journal of Satellite Communications and Networking*, vol. 41, no. 4, 2023.
[12] E. Volodina and E. Rathgeb, "Impact of ack scaling policies on QUIC performance," in *Proceedings of IEEE LCN*, Oct. 2021.
[13] D. Liu, M. Allman, S. Jin, and L. Wang, "Congestion control without a startup phase," in *Proceedings of the Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, Feb. 2010.
[14] J. Beshay, "Improving transfer times in the backbone network using QUIC jump start," @Scale. [Online]. Available: https://www.youtube.com/watch?v=E3RUgw2-k0g
[15] S. Ha and I. Rhee, "Taming the elephants: New TCP slow start," *Computer Networks*, vol. 55, no. 9, pp. 2092–2110, 2011.
[16] B. Peters, P. Zhao, J. Chung, and M. Claypool, "TCP HyStart performance over a satellite network," in *Proceedings of the 0x15 NetDev Conference*, Jul. 2021.
[17] P. Balasubramanian, Y. Huang, and M. Olson, "HyStart++: Modified slow start for TCP," 2023, IETF RFC 9406.