POSTER: Implementation of TCP SEARCH in FreeBSD and Evaluation on a Satellite Network

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

Samuel Ollari
Worcester Polytechnic Institute
Worcester, MA, USA
saollari@wpi.edu

Benjamin Skarnes
Worcester Polytechnic Institute
Worcester, MA, USA
bcskarnes@wpi.edu

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

Amber Cronin
Akamai
Cambridge, MA, USA
acronin@wpi.edu

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

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

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

***Abstract*—TCP's slow start phase is particularly inefficient over most wireless networks, especially high-latency, high-bandwidth paths such as satellite networks, often exiting too early or too late (after packet loss). To address this, the Slow start Exit At CHokepoint (SEARCH) algorithm is designed to improve exit decisions during slow start by analyzing delivery trends across sliding RTT-based windows. This paper presents a first implementation of SEARCH in the FreeBSD kernel using FreeBSD's modular congestion control framework. We evaluate our implementation on a testbed with an actual GEO satellite link with ˜600 ms RTT and 150 Mb/s capacity. Preliminary results show that SEARCH exits slow start more effectively than HyStart and HyStart++, achieving higher throughput and better utilization.**

***Index Terms*—SEARCH, FreeBSD, slow start, satellite networks**

I. Introduction

Transmission Control Protocol (TCP) remains the dominant transport-layer protocol for reliable data transfer across the Internet. A critical component of TCP's performance is its slow start mechanism, which controls how a connection ramps up its sending rate at the beginning of a transfer. While effective on many networks, slow start performs poorly on most wireless networks [1]. Slow starts ineffectiveness is exacerbated on high bandwidth-delay product networks such as over satellite links, often exiting too early – thus underutilizing link capacity – or exiting too late – thus increasing congestion and packet loss [2].

To address this challenge, a slow start algorithm called Slow start Exit At CHokepoint (SEARCH) [3] was developed. SEARCH improves slow start performance by monitoring delivered bytes over fixed duration bins, comparing recent delivery rates to delivery rates one round-trip time (RTT) ago, and exiting based on a normalized delivery rate difference. By tracking delivery rate behaviors instead of relying solely on loss or delay, SEARCH can detect when the congestion point is reached for more timely exits from slow start.

SEARCH has been implemented and evaluated in both Linux[1] and QUIC[2] and shown to be effective in various wireless network environments. In this work-in-progress, we present an implementation of SEARCH in FreeBSD, a widely-used operating system in data centers, embedded systems, and commercial platforms such as Netflix's CDN infrastructure [4].

After validating our FreeBSD implementation, we evaluate it using a real-world testbed with a GEO satellite link. Preliminary results indicate that SEARCH exits slow start more effectively than HyStart and HyStart++ [5], achieving higher throughput and avoiding premature exits. Future work may see SEARCH integration into FreeBSD's congestion control ecosystem and additional evaluation across heterogeneous networks.

II. Methodology

Our goal was to implement the SEARCH algorithm in a congestion control (CC) module in FreeBSD, while preserving the behavior and design decisions from the Linux implementation. This required adapting the algorithm to fit FreeBSD's CC framework and interfacing with its TCP internals.

A. SEARCH Algorithm Overview

During slow start, traditional congestion control algorithms approximately double the number of packets in flight every RTT, which causes the sending rate to exceed the bottleneck link capacity. The SEARCH algorithm tracks this behavior by monitoring byte delivery (via acknowledgments) growth to determine the right exit point.

SEARCH divides time into fixed-duration bins derived from the initial RTT and records the number of bytes acknowledged (ACKed) within each bin. It maintains a sliding window of 10 bins spanning 3.5 RTTs. For each new bin, the total delivered bytes in the current window ($curr\_window$) are compared

[1]https://github.com/Project-Faster/tcp_ss_search
[2]https://github.com/Project-Faster/quicly/tree/generic-slowstart

to the bytes delivered one RTT earlier ($prev\_window$). The normalized difference ($norm$) is computed as:

$$norm = \frac{2 \cdot prev\_window - curr\_window}{2 \cdot prev\_window}$$

When the $norm$ exceeds a pre-defined threshold (0.35), indicating that throughput growth has flattened, SEARCH exits the slow start phase. The parameters used in SEARCH – window size, threshold, and bin counts – have been derived in prior work.[3]

B. Implementation of SEARCH in FreeBSD

We implemented SEARCH as a modular congestion control algorithm in FreeBSD 14, selected for its API stability. Unlike the Linux version, which integrates SEARCH into TCP Cubic, our FreeBSD implementation is based on TCP NewReno. This choice aligns with Netflix's production deployment of NewReno and may facilitate future testing using their CDN infrastructure.

Our implementation extends FreeBSD's `cc_newreno` module. Since SEARCH is updated upon receiving each ACK, its logic is inserted into the `ack_received` callback. The algorithm initializes bin-related state during connection setup and finalizes configuration upon receiving the first ACK when an RTT estimate becomes available (the initial RTT). Delivered bytes are tracked in fixed-length bins, with the bin duration derived from the initial RTT. A circular buffer maintains a history of bins spanning 3.5 RTTs, along with additional bins to support shifting back by one RTT for comparison (25 bins in total). For each ACK, the number of newly acknowledged bytes is recorded in the current bin, and a normalized difference is computed when a full comparison window becomes available. Exiting from slow start is triggered when the normalized difference exceeds a defined threshold, consistent with the SEARCH heuristic.

In our FreeBSD implementation, RTT estimation is performed using the `t_srtt` field from the TCP control block. While alternative sources such as `h_ertt` were evaluated, `t_srtt` was preferred for its higher resolution and consistent behavior under high-RTT satellite conditions.

The SEARCH FreeBSD module is configured at runtime using `sysctl`, and the implementation is compatible with FreeBSD's congestion control lifecycle. This should enable interoperability with user-level applications and simplify deployment for further testing.

III. Evaluation

SEARCH experiments are conducted over a real GEO satellite link, as illustrated in Figure 1. The client is a Linux machine (Ubuntu 20.04, kernel 5.4) connected via Gigabit Ethernet to a Viasat-2 satellite terminal. The server is a virtual machine running FreeBSD 14, hosted on a Debian-based system (kernel 6.1) using KVM virtualization and connected to the university network over Gigabit Ethernet.

[3]https://github.com/Project-Faster/tcp_ss_search

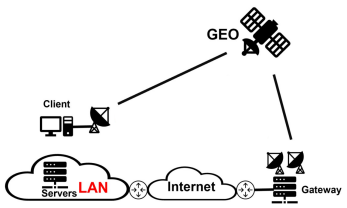

Fig. 1: GEO satellite measurement testbed.

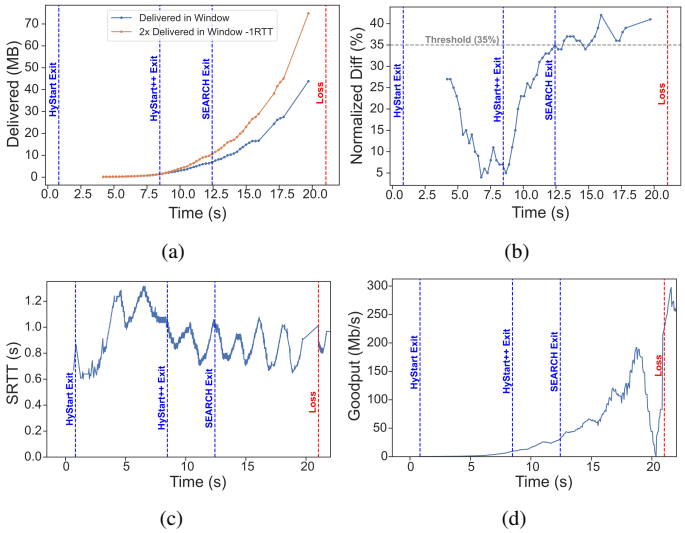

Fig. 2: Per-flow behavior over the GEO satellite link during slow start: (a) Bytes delivered, (b) Normalized difference, (c) Smoothed RTT, and (d) Goodput.

The satellite terminal communicates with the Viasat-2 satellite using a Ka-band outdoor unit and supports a peak downlink rate of 144 Mb/s. To evaluate end-to-end TCP behavior, the performance-enhancing proxy (PEP) on the GEO link is disabled. The Viasat gateway applies active queue management (AQM), triggering random packet drops once per-client queues exceed 18 MB (about 1.5x the bandwidth-delay product). This setup enables us to evaluate SEARCH under realistic, high-delay satellite network conditions, with a baseline RTT of approximately 600 ms and a bandwidth cap of 150 Mb/s.

We compare the performance of SEARCH against HyStart and HyStart++ in our testbed using our FreeBSD implementation. We conducted a 24-hour experiment, yielding 77 iperf3 download runs. The original data was collected with NewReno with the kernel generating log messages for SEARCH, and exit times based on HyStart, HyStart++, and SEARCH.

Figure 2 shows a representative example of how SEARCH, HyStart, and HyStart++ behave during a single bulk download over the Viasat GEO satellite link. For all graphs, the x-axis is the time (in seconds) since the download started. The vertical dashed red line marks the first packet loss, while the vertical blue dashed lines—labeled in the figure—mark the exit points from slow start for SEARCH, HyStart++, and HyStart.

In Figure 2a, the blue curve shows the number of bytes deliv-

ered within the current sliding window, while the orange curve shows twice the number of bytes delivered one RTT earlier. At the start, both curves track closely, indicating steady growth. As the link nears capacity, the two lines diverge, signaling that the delivery rate is no longer doubling. Figure 2b plots the normalized difference between the current and previous delivered-bytes windows. Once the value exceeds the 0.35 threshold (35%) – the horizontal dashed gray line – SEARCH exits slow start. This exit occurs just before congestion manifests (vertical dashed red line), showing that SEARCH detects the slowing delivery trend. In contrast, HyStart and HyStart++ exit much earlier, well before the network is fully utilized. Figure 2c confirms that the RTT exhibits fluctuations which act as noise for delay-based signals, which likely mislead HyStart and HyStart++ into exiting early. Figure 2d shows that throughput continues to increase after HyStart and HyStart++ exit, peaking just before the first loss event, suggesting their premature exit. SEARCH exits later than HyStart and HyStart++, avoiding loss while achieving higher throughput.

This case study illustrates that SEARCH more accurately identifies the chokepoint for slow start exit, enabling better utilization of high BDP satellite links without causing congestion.

Figure 3a shows the cumulative distribution of slow start exit times for each algorithm over all cases. HyStart exits almost immediately, with over 90% of flows exiting within the first 2 seconds. HyStart++ exits more gradually, with the majority of flows exiting between 5 and 10 seconds. SEARCH exits later, with most flows exiting between 10 and 17 seconds – closer to the onset of congestion.

Figure 3b shows the cumulative distribution of goodput measured at the point where each algorithm exits slow start for all cases. HyStart exits almost immediately, resulting in very low goodputs – nearly all flows exit below 10 Mb/s. HyStart++ performs better, with most flows exiting between 10 and 100 Mb/s, but still having unused capacity. SEARCH achieves significantly higher goodput, with many flows exiting between 30 and 150 Mb/s. In contrast, loss-based exits tend to occur at high goodputs—often above 150 Mb/s – but at the cost of triggering a congestion response.

Figure 3c shows the trade-off between headroom and goodput at the point of slow start exit for all cases. Headroom is defined as the number of RTTs between the exit point and the first loss event. An effective algorithm should provide enough headroom to the congestion avoidance algorithm can avoid a congestion response (e.g., dropped packets, rate reduction, retransmissions), while not exiting too early, which can result in underutilization and limit goodput. From the figure, HyStart consistently exits with large headroom (often 20-50 RTTs), but with low goodputs, indicating significant underutilization. HyStart++ achieves modest goodputs but still tends to exit conservatively. In contrast, SEARCH exits with moderate headroom (typically 5–20 RTTs) and higher goodput. This indicates that SEARCH is able to probe more aggressively while still leaving a sufficient safety margin before congestion.

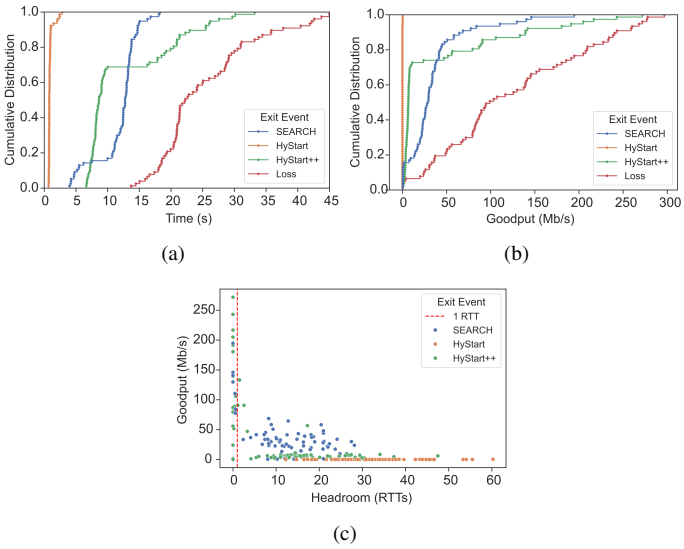

Fig. 3: (a) Slow start exit times, (b) Goodput at slow start exit, (c) Goodput at exit versus headroom (in RTTs).

IV. Conclusion

This work presents the first implementation of the SEARCH slow start algorithm in the FreeBSD kernel, extending SEARCH to BSD-based systems commonly used in data centers and CDNs. By integrating SEARCH into NewReno and validating it over an actual GEO satellite link, we demonstrate that SEARCH consistently achieves higher throughput than HyStart and HyStart++, while maintaining sufficient headroom to avoid congestion loss. SEARCH exits slow start closer to the TCP operating point, making better use of the available bandwidth without triggering congestion.

Future work includes conducting evaluations across a wider range of network environments, including LEO satellite, WiFi, 4G, and 5G networks, as well as assessing the cost of overshooting the chokepoint. We also aim to integrate SEARCH into production testbeds and contribute the module to the FreeBSD codebase for broader adoption and validation.

References

[1] P. Bruhn, M. Kuehlewind, and M. Muehleisen, "Performance and Improvements of TCP CUBIC in Low-delay Cellular Networks," *Computer Networks*, vol. 224, p. 109609, 2023.

[2] M. Arghavani, H. Zhang, D. Eyers, and A. Arghavani, "SUSS: Improving TCP Performance by Speeding Up Slow-Start," in *Proceedings of the ACM SIGCOMM Conference*, 2024, pp. 151–165.

[3] M. A. Kachooei, J. Chung, F. Li, B. Peters, J. Chung, and M. Claypool, "Improving TCP Slow Start Performance in Wireless Networks with SEARCH," in *WoWMoM Symposium*, 2024, pp. 279–288.

[4] FreeBSD Foundation, "Netflix Case Study: Maintaining the World's Fastest Content Delivery Network at Netflix on FreeBSD," Nov. 2023. [Online]. Available: https://freebsdfoundation.org/netflix-case-study/

[5] P. Balasubramanian, Y. Huang, and M. Olson, "RFC 9406: HyStart++: Modified Slow Start for TCP," 2023.