Improving TCP Slow Start Performance in Wireless Networks with SEARCH

Maryam Ataei Kachooei

A Dissertation
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Computer Science

December 2024

APPROVED:

Professor Mark Claypool
Advisor
Worcester Polytechnic Institute

Professor Shubbhi Taneja
Reader
Worcester Polytechnic Institute

Abstract

Transmission Control Protocol slow start is designed to gradually ramp up the congestion window, aiming to match the available network capacity while minimizing congestion. However, this mechanism often struggles in high bandwidth-delay product networks, leading to premature exits from slow start or excessive packet loss. This thesis introduces a novel algorithm, Slow start Exit At Right CHokepoint (SEARCH), which aims to address these challenges. SEARCH accurately determines the point at which slow start should exit by analyzing delivered bytes in comparison to expected rates over multiple round-trip times. By smoothing delivery rates over several round-trip times and normalizing bandwidth estimates, SEARCH adapts to varying conditions, preventing early exits while maintaining high throughput and avoiding unnecessary congestion.

The thesis details the design and implementation of SEARCH across multiple high bandwidth-delay product network types, including satellite (GEO, LEO), and 4G LTE networks, compared to traditional TCP (CUBIC without HyStart) and default TCP (CUBIC with HyStart). Extensive experiments show that SEARCH consistently exits slow start at the right capacity point, improving throughput and reducing packet loss across diverse network environments.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor Mark Claypool, for his invaluable guidance, unwavering support, and insightful feedback throughout my research journey. His mentorship has been instrumental in shaping my academic and professional growth.

I extend my sincere appreciation to Dr. Jae Chung and Dr. Feng Li from Viasat for their continuous support, technical insights, and thoughtful feedback, which have significantly enhanced the quality of my research. Their expertise and valuable discussions have played a crucial role in the development of this work.

Additionally, I would like to thank my thesis reader, Professor Shubbhi Taneja, for her time and constructive feedback. Her review and suggestions have helped improve the clarity and depth of this thesis.

Finally, I am deeply thankful to my family, friends, and colleagues for their encouragement and support throughout this journey. Their patience and motivation have been invaluable in helping me complete this work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In today's interconnected world, the efficient transmission of data across networks is fundamental. The Transmission Control Protocol (TCP), is an essential part of Internet communication and employs various mechanisms to regulate data flow and prevent network congestion. Congestion control plays a critical role in modulating the data transfer rate, with the primary goal of fully utilizing the bottleneck link and preventing congestion collapse. Full utilization of a network link occurs when the in-flight data equals the path's bandwidth-delay product (BDP). The BDP is calculated by multiplying the maximum bandwidth of the bottleneck link with the path's round-trip time (RTT), excluding queue delay.

TCP congestion control has two phases: slow start and congestion avoid-

ance. In the slow start phase, the congestion window (cwnd) increases rapidly to probe the available network capacity. This phase aims to quickly identify the bandwidth, doubling the cwnd with each RTT until signs of congestion, such as packet loss or excessive delay, are detected. Once congestion signals are received, the algorithm transitions to the congestion avoidance phase. In this phase, the cwnd increases gradually, typically by one segment per RTT, to cautiously explore additional available bandwidth while minimizing the risk of congestion. HyStart [15] is designed to improve this transition between slow start and congestion avoidance. It aims to avoid overshooting the network capacity, which can lead to unnecessary packet loss. By employing various heuristics, HyStart tries to detect the point to shift from slow start to congestion avoidance. HyStart is enabled by default in Linux.

However, TCP's performance can suffer significantly in high BDP networks. The slow start algorithm faces difficulties in such environments due to the extended time required to ramp up the cwnd to meet link capacity [29]. Default implementations, such as TCP CUBIC with HyStart in Linux, experience premature exits from the slow start phase. When HyStart is disabled, excessive packet loss can occur due to overshooting link capacity.

Exiting slow start prematurely limits TCP's ability to utilize the full bandwidth potential, a significant drawback in high BDP networks where rebuilding the cwnd can be slow. Conversely, a delayed exit can overshoot link capacity, inducing congestion and packet loss, which is especially problematic for links with large, bloated bottleneck queues.

Thus, a primary goal for connections over wireless links is to ensure that the TCP connection maintains its growth in the congestion window 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, which is essential for maintaining high throughput. A secondary goal is to prevent packet loss by avoiding a late exit from slow start. 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 [17], 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 [20, 22]. TCP with HyStart [15] considers packet timing to find a slow start exit point before packet loss but is destructive for high BDP networks by exiting slow start too early. HyStart++ [5], which leverages incremental changes in RTT to signal an exit, suffers similarly. TCP with Bottleneck Bandwidth and Round-trip propagation time (BBR) [8] 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.

While several strategies have been proposed to improve TCP perfor-

mance, many face limitations in adapting to high BDP networks. These challenges highlight the ongoing need for innovative techniques that can effectively navigate the complexities of high BDP networks. Our approach aims to address these shortcomings by introducing a novel algorithm designed to dynamically optimize TCP performance, offering more reliable data transmission and a better user experience.

1.2 Objectives

The primary objectives of this thesis are:

1. Develop an algorithm to exit TCP slow start after reaching the network's capacity, but before inducing packet loss.

2. Design and develop techniques to accomplish objective 1 across a wide range of network conditions. This includes establishing default parameter values for handling RTT and capacity variations.

1.3 Contributions

The contributions of this thesis can be summarized as follows:

1. Identification of Factors: Identify the factors that affect TCP performance in high BDP networks; laying the groundwork for developing solutions to address these challenges.

2. SEARCH Algorithm: Introduce the Slow start Exit At Right CHokepoint (SEARCH) algorithm, an approach to improve TCP performance, particularly for high BDP networks. SEARCH monitors the disparity between delivered and expected delivered bytes, providing a reliable indicator for exiting slow start at the network chokepoint.

3. Parameters Tuning: Conduct an analysis of the SEARCH algorithm parameters, including justification for windowed smoothing and delivery-difference thresholds. Explore the sensitivity of algorithm parameters to provide insights into heuristic settings.

4. Empirical Evaluation: Validate the effectiveness of the SEARCH algorithm through experiments involving hundreds of TCP downloads over geostationary earth orbit (GEO), low earth orbit (LEO), 4G LTE, and WiFi links. Provide an empirical comparison against default and traditional TCP implementations to assess the algorithm's performance.

1.4 Thesis Outline

This thesis comprises seven chapters. Chapter 1 introduces the research, outlining its motivation, objectives, and contributions. Chapter 2 presents an overview of the foundational concepts that form the basis of this research. Chapter 3 reviews related work, highlighting limitations in current TCP implementations and setting the context for the SEARCH approach. Chapter

4 details the design and evolution of SEARCH, describing its algorithms, key parameters, and solutions to challenges like RTT variation. Chapter 5 evaluates SEARCH's performance across satellites, 4G LTE, and WiFi testbeds, comparing it with other TCP strategies. Chapter 6 summarizes the findings and impacts of the research. Chapter 7 proposes future improvements and adaptations of SEARCH.

Chapter 2

Background

This chapter provides an overview of the foundational concepts relevant to this research. It introduces TCP mechanisms and the wireless network environments used for evaluation.

2.1 TCP Overview

The Transmission Control Protocol is part of the TCP/IP protocol, which is used for reliable communication between devices in a network. TCP establishes a connection-oriented communication channel between a source and destination, where it breaks data into segments at the source and reassembles them at the destination. This protocol retransmits any segments that are lost or corrupted during transmission to maintain data integrity. TCP uses sequence numbers and acknowledgment numbers to manage the data flow,

ensuring that segments are received in the correct order and without errors.

TCP also utilizes a set of control flags, such as SYN, ACK, and FIN, to manage the connection's state, allowing it to be established, maintained, and gracefully terminated. The protocol incorporates flow control, using the window field to regulate the rate of data transmission based on network conditions. Additionally, TCP employs a checksum to verify the integrity of each segment, enabling error-free data transfer [2]. Beyond these foundational features, TCP supports various congestion control algorithms—such as Reno, CUBIC, and BBR—designed to improve performance by managing network congestion and improving the efficiency of data transmission in different network environments.

2.1.1 Slow Start Phase

As shown in figure 2.1, TCP starts in the slow start phase. The TCP slow start mechanism is an adaptive algorithm designed to start data transmission rates cautiously yet rapidly to increase the available link capacity, doubling the congestion window for each RTT [27]. Initially, TCP starts with a conservative cwnd, sending a small number of packets and monitoring their successful delivery. With each successful acknowledgment (ACK) received, the congestion window size is incrementally expanded. This results in an exponential growth of the congestion window size, as each ACK doubles the number of segments sent in the next RTT. This gradual incrementing continues until it reaches a point where network congestion is encountered, causing

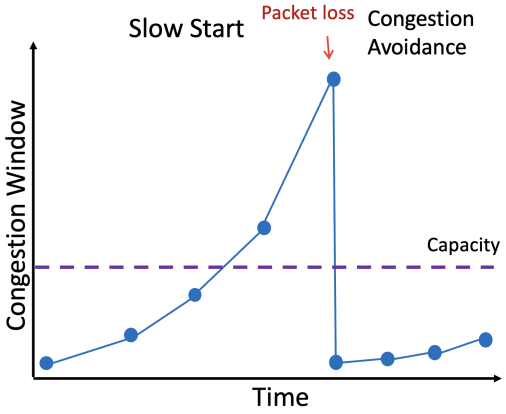


Figure 2.1: Traditional TCP

packet loss or indications of network strain. At this juncture, TCP's congestion avoidance mechanisms are invoked to regulate the data flow more conservatively, transitioning from the rapid growth of the slow start phase to a more moderate incrementing, thereby preventing network congestion while optimizing data transmission rates.

2.1.2 HyStart

HyStart is an enhancement to the traditional slow start algorithm, designed to improve the performance of TCP connections in diverse network environments [15]. It modifies the way TCP detects congestion, enabling an earlier transition from the slow start phase to the congestion avoidance phase.

Unlike the traditional slow start, which increases the congestion window exponentially until packet loss occurs, HyStart monitors the delay and inter-arrival time of acknowledgments during the startup phase. By identifying

when these metrics begin to show signs of congestion, such as increased round-trip times or jitter, HyStart can exit slow start before packet loss happens. This early detection helps to avoid the typical burst of packet loss associated with the standard slow start.

2.1.3 TCP CUBIC

TCP CUBIC [16] is a TCP congestion control algorithm designed for high-bandwidth, high-latency networks. It is an enhancement of the traditional TCP Reno [9]. TCP CUBIC is the default congestion control algorithm in several major operating systems, including Linux, due to its efficiency in utilizing available network capacity.

The core innovation of TCP CUBIC lies in its cubic growth function, which governs the size of the congestion window. TCP CUBIC employs a cubic function to modulate the cwnd. When the cwnd is far below the network's saturation point, it grows rapidly to recover throughput efficiently. As the cwnd approaches the saturation point, the growth slows to avoid overshooting and congestion. After surpassing the saturation point, the growth accelerates again to fully utilize the available bandwidth. Upon detecting packet loss, TCP CUBIC reduces the cwnd and sets slow start threshold (ssthresh) to a value based on the cwnd size at the time of loss. The cubic growth function then takes over, allowing the cwnd to recover efficiently.

2.1.4 TCP BBR

TCP BBR (Bottleneck Bandwidth and Round-trip propagation time) [8] is a congestion control algorithm developed by Google to optimize data transmission over the Internet that takes a different approach compared to traditional loss-based algorithms. Unlike traditional congestion control algorithms like TCP CUBIC, which detect network congestion based on packet loss, BBR focuses on maximizing throughput while keeping latency low. It does this by regularly probing the network to measure the available bandwidth and round-trip time, then adjusting the sending rate to match the measured bottleneck.

BBR's innovation lies in its departure from loss-based congestion control methods, which can often lead to inefficient use of network capacity and increased latency, particularly in modern networks with large buffers. By decoupling the congestion control decisions from packet loss and instead basing them on real-time estimates of the network's bottleneck capacity, BBR avoids the excessive queuing delays and periodic packet loss cycles common in other algorithms.

2.2 Wireless Networks

In our testing environment, we focus on evaluating network performance across various wireless network technologies, each offering unique characteristics and challenges. Our testbeds include satellite networks, 4G LTE, and

Wi-Fi, allowing us to comprehensively analyze and compare performance metrics in diverse scenarios.

2.2.1 Satellite Networks Overview

Satellite networks are a type of high BDP network. In recent years, satellite networks have gained significant popularity, particularly in remote areas where terrestrial networks are not available. Low Earth Orbit (LEO) satellite networks employ numerous small satellites that orbit the Earth at relatively low altitudes, covering between 3 and 12 percent of Earth's surface at altitudes of 400 and 2000 km, respectively. These networks have asymmetric bitrates upwards and downwards, with delays and capacities at 20-40 ms and 100 Mb/s, respectively. On the other hand, Geostationary Earth Orbit (GEO) satellite networks use a single satellite at a much higher altitude of 36,000 km, resulting in round-trip delays of around 600 ms (Figure 2.2). The inherent delay and the potential for high error rates due to long transmission distances present challenges for TCP performance.

2.2.2 4G LTE Networks Overview

4G Long-Term Evolution (LTE) networks are widely deployed cellular networks known for their high-speed data capabilities and relatively low latency compared to satellite networks. Typically, 4G LTE networks have latencies of about 60 ms and can achieve download speeds of up to 100 Mb/s under

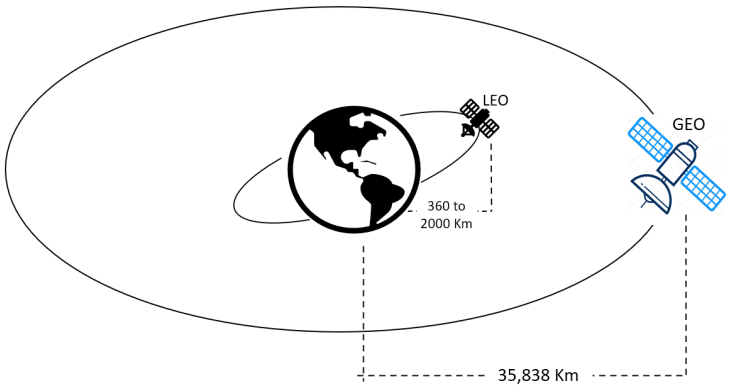


Figure 2.2: Satellite geometric configuration observation in different orbits

optimal conditions. The mobility support and widespread availability of 4G LTE make it a versatile option for various applications, from mobile browsing to video streaming. However, factors such as signal strength, network congestion, and handover events can impact performance, posing additional challenges for TCP protocols [4].

2.2.3 WiFi Overview

WiFi networks, based on IEEE 802.11 standards, provide high-speed wireless connectivity over short distances. Operating primarily in the 2.4 GHz and 5 GHz frequency bands, WiFi networks offer varying levels of coverage and throughput. Under ideal conditions, modern WiFi networks can achieve latencies as low as a few milliseconds (about 1-2 ms) and data rates exceeding 1 Gbps [10]. However, WiFi networks present unique challenges that significantly impact the performance of the TCP. In congested environments, multiple devices contend for the shared link capacity, leading to congestion.

Interference from neighboring WiFi networks or other devices operating on the same frequency as a WiFi access point can degrade signal quality, resulting in packet loss and increased latency.

Our testbed incorporates all these network types used for evaluating the SEARCH algorithm.

Chapter 3

Related Work

This chapter explores related work on TCP slow start mechanisms and the performance over high BDP networks, highlighting their limitations.

3.1 TCP Slow Start

The TCP slow start mechanism is an important part of TCP's congestion control strategy since its introduction. Over the years, enhancements and alternatives have been proposed to address its limitations, especially in different network conditions.

Ye et al. [28] introduce Personalized FAST TCP, an algorithm designed to improve the performance of FAST TCP in personalized healthcare systems. This approach uses the number of remaining link buffers to determine slow start exit times, maintaining the queuing delay ratio within expected ranges

despite variations in bandwidth, leading to faster system convergence. However, while FAST TCP adjusts quickly based on queuing delay, it may not scale well with dynamic bandwidth changes or handle different environments effectively. On the other hand, SEARCH focuses on exiting slow start only when the link capacity is reached, rather than relying solely on queuing delay. This makes SEARCH more effective in networks with high variability in delay and bandwidth, such as satellite networks, where queuing delay alone is insufficient to indicate capacity limits.

Kasoro et al. [23] propose ABCSS, a method combining Appropriate Byte Counting (ABC) and the Slow Start (SS) algorithm to increase the congestion window more effectively during initial round trips, addressing issues like TCP burstiness and potential buffer overflows. ABCSS attempts to address issues like TCP burstiness and buffer overflows during initial round trips by combining byte counting with slow start, but it can still suffer from congestion in high BDP environments. SEARCH handles high BDP networks by normalizing delivery rates and smoothing latency variation over several RTTs. This is designed to provide slow start exit point that is more accurate, even in fluctuating environments.

Huang and Olson's IETF draft called HyStart++ [19] offers an alternative to HyStart, implemented in the Microsoft Windows TCP CUBIC stack. HyStart++ removes the ACK train method and introduces a Limited Slow Start (LSS) phase, which grows the congestion window more gradually than traditional slow start but faster than congestion avoidance. This approach

transitions from slow start to LSS, and then to congestion avoidance only when packet loss occurs, potentially reducing packet loss compared to legacy slow start. However, like HyStart, HyStart++ may still exit slow start prematurely in certain scenarios. In contrast, SEARCH improves upon this by analyzing delivered bytes instead of relying on RTT increments. This allows SEARCH to exit slow start precisely at the capacity limit, enhancing throughput without increasing packet loss.

Lübben et al. [25] propose an innovative approach to improve TCP slow start by forecasting the TCP rate using a neural network, which improves slow start performance in terms of completion time and fairness compared to traditional TCP algorithms. However, neural network-based forecasting can struggle with high variability in latency and capacity, which is common in wireless networks. In contrast, SEARCH improves slow start performance by dynamically adjusting based on actual bytes delivered, making it more robust in fluctuating environments like satellite and LTE networks, where accurate forecasting is challenging.

3.2 TCP Performance over High BDP Networks

High BDP networks present unique challenges for TCP, often requiring modifications to traditional congestion control mechanisms to optimize performance.

Li et al. [24] propose Fast Bandwidth Estimation (FBE), designed to optimize the slow start phase in high-bandwidth networks like WiFi 6 and 5G. While FBE improves convergence time by using refined ACK intervals to estimate bandwidth, it can still suffer from inaccuracies in rapidly changing environments. SEARCH can improve upon FBE by dynamically adjusting based on actual delivered bytes over multiple RTTs, making it more resilient to high variability in bandwidth and latency, as seen in satellite networks.

Jasim et al. [1] propose a technique for approximating the TCP congestion window thresholds for high latency connections. Their method uses a packet-pair technique to estimate bandwidth and improve congestion window thresholds in high-latency environments. However, packet-pair techniques may not accurately estimate bandwidth in highly variable networks, leading to premature slow start exits or underutilization. SEARCH, in contrast, monitors delivered bytes over a longer time window, for more accurate slow start exits even in fluctuating conditions.

G'al et al. [11] introduce BIC (Binary Increase Congestion Control) and Hybla, which aim to enhance TCP performance by modifying slow start and congestion avoidance to achieve RTT fairness. However, both algorithms still struggle with accurate bandwidth estimation in high BDP networks. SEARCH offers a more precise exit from slow start by analyzing byte delivery patterns, leading to better utilization of available capacity.

Grazia et al. [13] identify TCP BBR's inefficiency in WiFi networks, proposing BBRp to improve throughput. While BBRp enhances performance

by increasing throughput, it also raises latency, which can be detrimental to performance for some flows. In contrast, SEARCH attempts to exit slow start at the capacity point, making it better suited for maintaining high throughput and low delay in variable networks.

Nguyen et al. [26] evaluate TCP BBR's fairness in a smart device WiFi network compared to TCP CUBIC. They find that TCP CUBIC often outperforms TCP BBR due to differences in queue management and congestion window sizing, especially in upload scenarios, which are becoming increasingly relevant with the rise of 5G. SEARCH improves upon CUBIC by using delivered bytes to identify the correct slow start exit.

Grazia et al. [12] explore TCP Small Queues (TSQ) in WiFi networks, which can limit throughput but improve it when relaxed. While TSQ helps control latency by managing queue sizes, it lacks the capability to accurately identify slow start exit points, especially in fluctuating environments. In contrast, SEARCH determines the slow start exit based on actual delivery patterns and capacity limits.

Bruhn et al. [7] propose adjustments to TCP CUBIC tailored for low-delay cellular networks, improving performance by modifying congestion window growth. However, these adjustments can still lead to premature slow start exits or overshooting the link capacity due to the rapid changes typical of low-delay networks. SEARCH addresses these issues by using delivered bytes to determine the slow start exit point, adapting more effectively to varying conditions and minimizing packet loss.

Guo et al. [14] propose S-CUBIC, a stateful approach to TCP that uses information from previous flows to estimate path bandwidth, setting the initial congestion window accordingly. However, S-CUBIC can inaccurately estimate bandwidth in variable environments.

Kachooei et al. [22] present the BEST algorithm, a bandwidth estimation technique using packet-pair measurements. While it shows promise, BEST faces challenges in environments with high variability in bandwidth and RTT, leading to potential inaccuracies in bandwidth estimation. SEARCH addresses this limitation by smoothing out delivery rates over several RTTs.

Arghavani et al, [3] introduce SUSS (Speeding Up Slow-Start), which uses real-time feedback to rapidly adjust the congestion window. While this approach aims to accelerate throughput, its reliance on real-time feedback may lead to premature exits from slow start in high BDP networks. In contrast, SEARCH achieves a more precise slow start exit by comparing delivered bytes to expected delivery rates over multiple RTTs, maintaining higher throughput with reducing packet loss.

Brown et al. [6] propose adaptable congestion control strategies for high BDP networks, aiming to improve overall efficiency through fairness, adaptability, and robustness. However, these strategies are designed to optimize TCP behavior across various phases, which may not provide the granularity needed to accurately determine slow start exits, especially in variable network conditions. In contrast, SEARCH is specifically focused on identifying the slow start exit point by dynamically monitoring delivered bytes relative

to expected rates, leading to full utilization of link capacity and avoiding packet loss in high BDP networks.

While the approaches discussed above offer various improvements to the traditional slow start, they still face challenges with premature exits or inaccurate bandwidth estimation, particularly in high BDP networks. In contrast, SEARCH specifically addresses the problem of exiting slow start at the capacity point, avoiding both early exits and unnecessary congestion.

Chapter 4

SEARCH Approach

Effective congestion control in TCP relies on interpreting signals from the network to deduce congestion and adjust the load accordingly. Traditional TCP congestion control relies on loss-based mechanisms, where packet loss signals congestion. Packet loss occurs when the switch buffers along a network path reach capacity and incoming packets must be discarded.

HyStart, a congestion control approach, is designed to circumvent packet loss by using delay-based and ack-train-based strategies to exit the slow start phase before packet loss. By monitoring the ACK train and comparing predicted and actual RTT samples, HyStart identifies congestion. Despite its effectiveness in certain scenarios, HyStart proves inadequate in some wireless networks, exiting the slow start phase prematurely. This premature exit hinders the network from fully utilizing its capacity, resulting in lower resource utilization and decreased overall throughput.

Balancing the challenge of premature exits with HyStart and the risk of overshooting and subsequent packet loss by exiting late without HyStart highlights the complexity of finding the right exit point during the slow start phase. To address these challenges, this study introduces the Slow start Exit At Right CHokepoint (SEARCH) approach.

SEARCH operates on the principle that during the slow start phase, the congestion window increases by one for each ACK received, leading to the transmission of two packets, and effectively doubling the sending rate each RTT. However, when the network exceeds its capacity, the corresponding delivery rate fails to double as expected, indicating that the congestion point has been reached and it is time to exit the slow start phase. Leveraging this insight, SEARCH allows the sender to dynamically estimate link capacity by comparing the delivered bytes to the expected sent bytes. It does this by monitoring the difference between the delivered bytes in an RTT and the estimated sent bytes in the previous RTT. A large difference between delivered bytes and estimated sent bytes is a reliable indicator that slow start is at the network chokepoint and should exit.

4.1 Challenges and Solutions

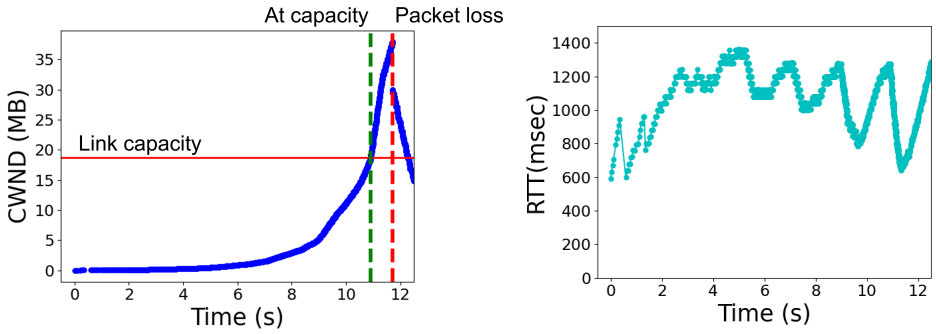
RTT variation, especially in high BDP networks, complicates the procedure of SEARCH. In standard networks, increased RTT signals congestion, but in some networks like satellites, RTT fluctuations may not align with network

capacity. 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. An additional challenge arises from our restricted data access. Since the server does not have access to receiver data, and there is a memory limit for each flow on the server side, the ability to store historical data is limited. This means we cannot store a vast amount of past data, although we do need to track sent and delivered bytes. Addressing these challenges, the following subsections describe the solutions we have developed.

4.1.1 RTT Variation

Measuring round-trip samples involves calculating the elapsed time between sending a data flight and receiving its acknowledgment. The accuracy of RTT measurements is challenged by packet queuing in switch buffers. This queuing phenomenon can introduce variations in RTT, leading to increased and, at times, inaccurate RTT samples. Figure 4.1a shows the congestion window size versus time during a download via the GEO satellite link. The cwnd growth occurs steadily, doubling each RTT during slow start. This is in contrast to the RTTs during this same time, shown in Figure 4.1b, which vary considerably even though the downlink is not at capacity.

To mitigate the impact of RTT variation and enhance the precision of



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

Figure 4.1: GEO satellite network.

our measurements, instead of considering delivered bytes within one RTT, we employ a sliding window of bytes that spans multiple RTTs. This strategy allows us to smooth out the effects of transient fluctuations caused by packet queuing. By considering a broader time frame, our windowing technique provides a more stable foundation for assessing RTT variation.

4.1.2 Limited Memory Per Flow on The Server Side

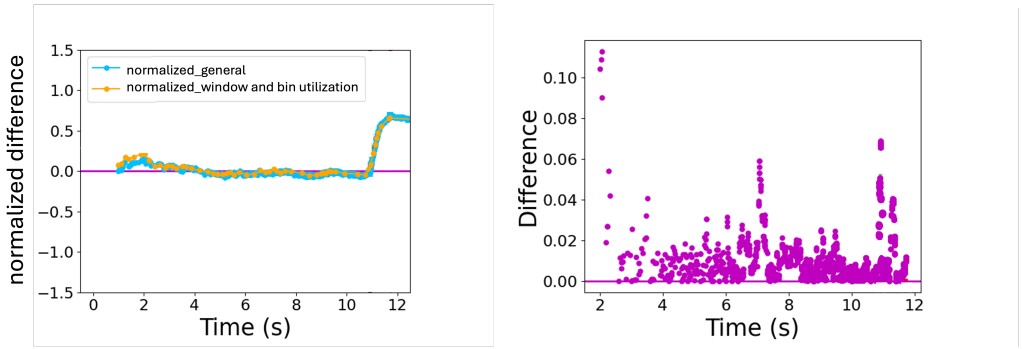
Due to memory constraints, tracking individual packet delivery times and bytes is impractical. We employ a pragmatic approach in which the data within the sliding window is organized into bins, each representing smaller, fixed time periods. This segmentation aggregates ACK data for memory efficiency. The sliding window then operates on a bin-by-bin basis rather than for every ACK packet. By triggering SEARCH only at the bin boundary, this strategy reduces the computational load in addition to the memory requirements.

This memory management approach maintains essential information related to link capacity estimation while efficiently navigating the limitations imposed by server memory constraints.

4.2 Impact of Window and Bin Utilization on Performance

To validate the precision of SEARCH's window and bin utilization, we conducted a comparative analysis of the difference between sent and delivered bytes using window and bin techniques against general measurement (not using window and bins). The sent-delivered bytes difference is normalized by dividing it by the sent bytes.

In Figure 4.2a, both normalized difference graphs are presented, with the blue line representing the original data and the orange line representing the data using windows and bins. The close alignment of the two lines indicates that the original and the values derived from the window and bin techniques are similar. In Figure 4.2b, the error between the original values and those obtained through window and bin utilization is depicted. Notably, all error values are relatively small, with an average error of approximately 2 percent. This emphasizes the effectiveness of SEARCH's window and bin utilization in providing accurate values without introducing significant errors.



(a) Normalized difference: general vs. window and bin utilization.

(b) Error in general case and window-bin utilization from (a).

Figure 4.2: General measurement vs. window and bin utilization - GEO satellite network.

4.3 SEARCH 1.0

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 the number of bytes sent in the previous RTT.

Figure 4.3 visually illustrates the core idea, mapping RTTs on the x-axes and bytes sent on the y-axis of the left graph and bytes delivered on the y-axis of the right graph. In this depiction, all bytes sent at time t_1 are acknowledged as delivered at time t_2, exemplifying the typical behavior during the slow start phase. This continues until sending capacity is surpassed at time t_4 whereupon the delivered bytes remain the same at time t_5. Thus, detecting

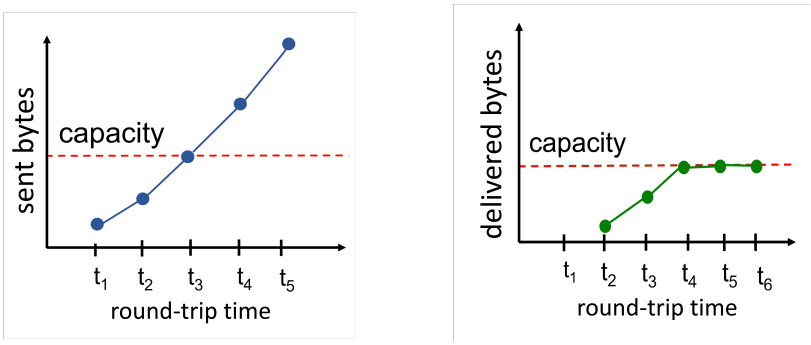


Figure 4.3: Theoretical transmission: sent and delivered bytes.

when bytes delivered are significantly lower than bytes sent one RTT earlier signals that the congestion window limit has been reached and slow start can safely exit.

To implement this concept, we introduce SEARCH Version 1.0, an algorithm that monitors both sent and delivered bytes over a large time window.

In Figure 4.4, we have two sets of data shown in windows: one for what is sent (blue window) and one for what gets delivered (green window). Each window is divided into a specific number of bins. The sending window is larger than the delivered window, allowing a look back in time (one RTT) to examine what was sent. By reaching each bin boundary, SEARCH checks if there are enough bins to shift back by one RTT. If so, SEARCH figures out how much got delivered in the current window, looks back one step, and calculates how much is sent.

Under normal network conditions, these two sets of data should be about the same. But when the link is at capacity, the sent data exceeds what is

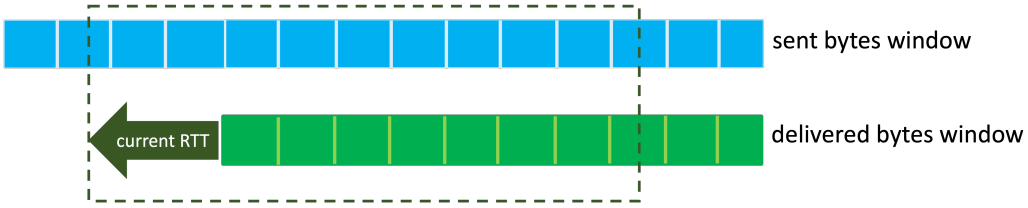


Figure 4.4: Sent and delivered bytes window

delivered. To detect this, SEARCH calculates the difference between sent and delivered data, dividing the difference by the current delivered bytes for normalization. In Figure 4.5, this dynamic interplay is visualized. As shown in the left figure, delivered bytes align with sent bytes until the capacity point (marked by a vertical dashed blue line). Post-capacity sent bytes increase while delivered bytes do not. The right figure highlights the normalized difference between these two windows. A zero difference indicates that the amount delivered one RTT later equals the amount sent, suggesting link capacity has not been reached. Any difference above zero signifies an increase in sent bytes beyond the network's capacity. The SEARCH algorithm introduces a threshold. When the normalized difference breaches a pre-set threshold, SEARCH triggers an exit from slow start phase.

4.4 SEARCH 2.0

In this section, we present SEARCH version 2.0, a version of the algorithm that simplifies the analysis and per-flow memory needed by focusing on delivered bytes. The key idea is that SEARCH can estimate the sent bytes

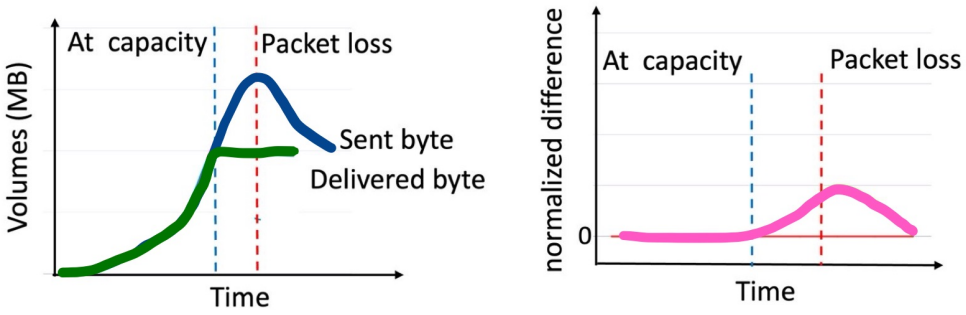


Figure 4.5: Visualization of sent and delivered bytes, and normalized difference

by doubling the delivered bytes in the same time period. This is based on the behavior of TCP in slow start, where receiving an ACK packet leads to an increment in the cwnd, effectively sending two packets. This doubles the sending rate each RTT. As shown in Figure 4.6, the number of delivered bytes (represented by the green line) is compared with twice the number of delivered bytes one RTT earlier (indicated by the blue line) along the y-axis against the RTTs on the x-axis. For instance, at t_3, $2a$ bytes are delivered, indicating that the delivered data doubles compared to the previous RTT (t_2). Also, at t_4, the number of delivered bytes remains consistent with twice the amount delivered during the preceding RTT (t_3), indicating that the network capacity has not been reached. This pattern persists at t_4, further indicating uncongested network conditions. However, at t_5, the number of delivered bytes remains $4a$, signaling a departure from the previously observed doubling trend. Despite attempting to double the congestion window and transmit doubled data, the receiver did not receive these additional bytes,

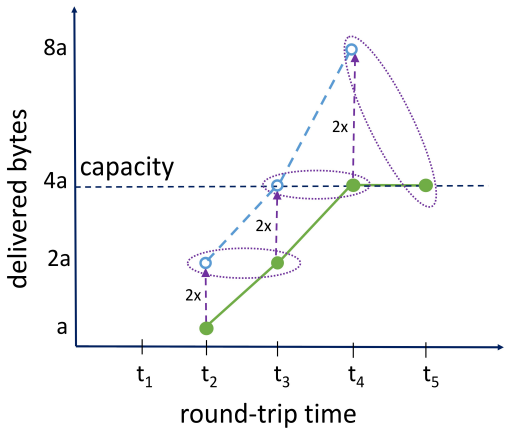


Figure 4.6: Estimating sending rate from acknowledged delivery of data over time.

resulting in a stagnant delivery rate. This difference between the delivered bytes and the expected delivered bytes (i.e., delivered bytes not doubling) is the signal to SEARCH that the congestion point has been reached and slow start can be exited before encountering packet loss.

In version 2.0, significant refinement comes with a reduced memory footprint, cutting memory use by approximately 50%. This reduction is significant for TCP servers which must store state per-flow so minimizing a flow's memory footprint is beneficial. In SEARCH 2.0, when the bytes delivered one RTT prior is one-half the bytes delivered now, the bitrate is not yet at capacity, whereas when the bytes delivered prior are greater than one-half the bytes delivered now, the link capacity has been reached and TCP exits slow start.

4.4.1 SEARCH 2.0 Algorithm Pseudocode

The SEARCH 2.0 algorithm is shown in Algorithm 1. In Lines 1-6, SEARCH initializes with predefined parameters for window size, number of bins(Extra bins and Window bins), bin duration, and threshold. The window size (WINDOW_SIZE) is set to 3.5 times the initial RTT (obtained via the first round-trip time measured in the TCP connection) to smooth out variation in link latency. The W constant represents the window size used for computing delivered bytes, which is set to 10 bins. To enable comparisons between the current delivered bytes and those from one RTT earlier, an additional 15 bins, referred to as EXTRA_BINS, are stored. These extra bins serve as a buffer for historical data, ensuring the mechanism can analyze trends or detect changes in delivery performance over time. The total number of bins allocated, NUM_BINS, is calculated as the sum of the window size and the extra bins, resulting in a total of 25 bins (NUM_BINS = W + EXTRA_BINS). 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 difference between the previous delivered bytes and the current delivered bytes (normalized) above which slow start exits.

After one-time initialization in Lines 7-9, the algorithm operates upon the arrival of each acknowledgment. On line 10, When the current time (**now**) has passed the end time of the last bin, in line 11, it updates the bin statistics in the function update_bins() defined in lines 23-30.

In update_bins, under most TCP connections, the time (**now**) is in the

successive bin, but in some cases (such as during an RTT spike or a TCP connection without data to send), more than one bin boundary may have been passed. Line 23 computes how many bins have been passed. In lines 25-27, for each bin passed, the bin[] variable is set to 0. For the latest bin, the delivered bytes are updated by taking the latest sequence number in the last bin boundary (line 29). In line 30, the current sequence number is stored (in prev_seq_num) for computing the delivered bytes the next time a bin boundary is passed.

Once the bins are updated, lines 12-13, check for enough filled bins for running SEARCH. This required covering at least a W bins, also it should make sure after shifting back by an RTT, there are enough bins for computing a W of delivered bytes that were delivered one RTT ago.

If there is enough bin data to run SEARCH, lines 14-16 compute the current and previous delivered bytes over a window (W) of bins, respectively. This sum is computed in the function sum_bins() in lines 31-37. For previous delivered bytes, shifting by an RTT may mean the SEARCH window lands between bin boundaries, so the sum is interpolated by the fraction of each of the end bins.

In the sum_bins() function, idx1 and idx2 are the indices into the bin[] array for the start and the end of the bin summation, and as explained above, the fraction is the proportion (from 0 to 1) of the end bins to use in the summation. In lines 32-34, the summation loops through the bin[] array for the middle bins, modulo the number of bins allocated (NUM_BINS), and

then adds the fractions of the end bins in lines 35-36.

Line 17 computes the normalized difference between the expected sent bytes one RTT ago (2.prev_delv) and the bytes delivered currently (curr_delv). Line 18 compares this normalized difference (norm_diff) to the threshold (THRESH). If norm_diff is greater than or equal THRESH, that means the current delivered bytes are lower than the expected sent bytes (i.e., they did not double over the previous RTT) and it is time to exit slow start. Therefore, SEARCH exits slow start by setting `ssthresh` to `cwnd`.

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

Parameters

WINDOW_SIZE = Initial_RTT × 3.5
W = 10
EXTRA_BINS = 15
NUM_BINS = W + EXTRA_BINS
BIN_DURATION = WINDOW_SIZE / W
THRESH = 0.35

Initialization():

bin[NUM_BINS] = {}
curr_idx = -1
bin_end = **now** + BIN_DURATION

ACK_arrived(sequence_num, rtt):

// Check if passed bin boundary.
if (**now** > bin_end) **then**
    update_bins ()

    // Check if enough data for SEARCH.
    prev_idx = curr_idx - (rtt / BIN_DURATION)
    if (prev_idx ≥ W **and** (curr_idx - prev_idx) ≤ EXTRA_BINS) **then**

        // Run SEARCH check.
        curr_delv = sum_bins(curr_idx - W, curr_idx)

        fraction = $\frac{\text{rtt mod BIN\_DURATION}}{\text{BIN\_DURATION}}$

        prev_delv = sum_bins(prev_idx - W, prev_idx, fraction)

        norm_diff = $\frac{2 \cdot \text{prev\_delv} - \text{curr\_delv}}{2 \cdot \text{prev\_delv}}$

        if (norm_diff ≥ THRESH) **then**
            // Exit slow start.
            ssthresh = cwnd
        end if
    end if
end if

```
// Update bin statistics, accounting for cases where more than one bin
   boundary might have been passed.
update_bins():
23: passed_bins = (now − bin_end) / BIN_DURATION + 1
24: bin_end += passed_bins × BIN_DURATION
25: for i = curr_idx to curr_idx + passed_bins do
26:     bin[i mod NUM_BINS] = 0
27: end for
28: curr_idx += passed_bins
29: bin[curr_idx mod NUM_BINS] = sequence_num - prev_seq_num
30: prev_seq_num = sequence_num

// Add up bins, interpolating a fraction of each bin on the end (default is
   0).
sum_bins(idx1, idx2, fraction = 0):
31: sum = 0
32: for i = idx1 + 1 to idx2 - 1 do
33:     sum += bin[i mod NUM_BINS]
34: end for
35: sum += bin[idx1] × fraction
36: sum += bin[idx2] × (1 - fraction)
37: return sum
```

4.5 SEARCH Parameters

Within the SEARCH algorithm, essential parameters such as window size, threshold value, and the number of bins require thoughtful consideration for functionality across diverse network scenarios. In this section, our focus shifts to a comprehensive analysis aimed at establishing default values for these key parameters.

4.5.1 Window Size

The SEARCH window smooths over baseline RTT fluctuations in a connection. The window size must be large enough to encapsulate link variation, yet small in order to allow SEARCH to respond near the point at which slow start reaches link capacity. To determine an appropriate window size, we do an analysis of RTTs for TCP during slow start.

The SEARCH window size should be chosen large enough to capture the observed periodic oscillations in the RTT values. To determine the oscillation period, a Fast Fourier Transform (FFT) [18] is employed to convert RTT values from the time domain to the frequency domain. A dominant peak (f_{peak}) can signify the duration of a periodic cycle, where the cycle occurs about $T_{\text{cycle}} \approx \frac{1}{f_{\text{peak}}}$.

For this purpose, we conducted an analysis of RTTs for GEO, LEO, and 4G LTE links during TCP slow start:

- **GEO satellite links:** Figure 4.7a shows the RTT observed over time averaged over 77 samples from a GEO network during TCP slow start preceding congestion. The FFT results are depicted in Figure 4.7b, where the x-axis represents frequency (in Hz) and the y-axis denotes 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 implies the cycle occurs about every $\frac{2}{0.6} \approx 3.33$ RTTs. Hence, a window size of

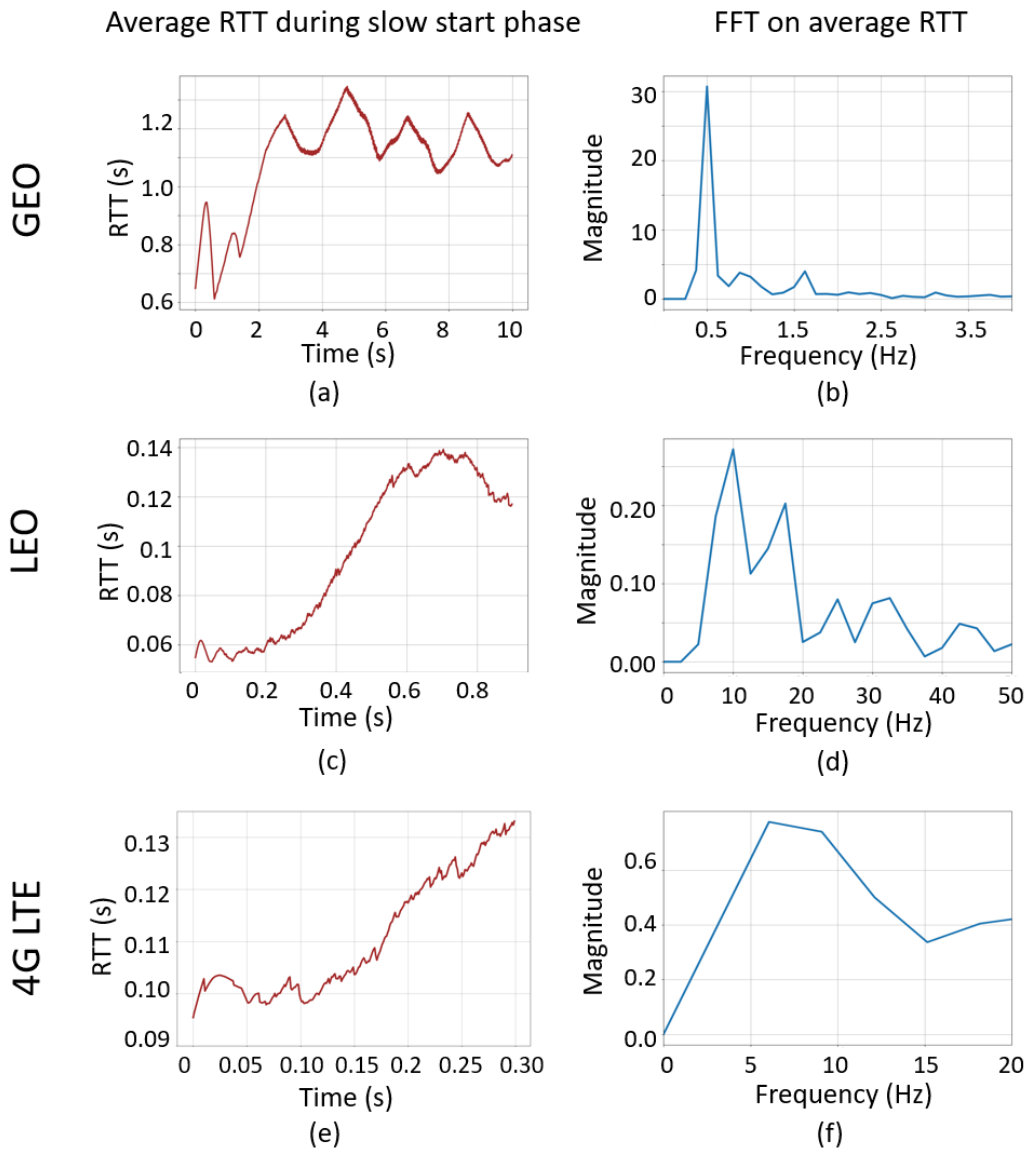


Figure 4.7: Average RTT and FFT over GEO, LEO, ad 4G LTE links

approximately 3.5 times the minimum RTT should effectively smooth out latency variations for this type of link.

- **LEO satellite links:** Figure 4.7c and Figure 4.7d show similar analysis

for a LEO satellite link, using RTT values for 77 TCP transfers during slow start before 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 $\frac{0.1}{0.03} \approx 3.33$ 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.

- **4G LTE networks:** Figure 4.7e and Figure 4.7f show analysis of a 4G LTE network for RTT values collected across 55 TCP transfers during slow start before congestion. Similar 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 of 60 ms, this means a window size of at least $\frac{0.17}{0.06} \approx 2.8$ 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.

Consequently, a 3.5 $\times$ RTT is recommended for the window size.

4.5.2 Threshold

The threshold determines when the normalized difference between the bytes delivered currently and the bytes delivered in the 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.

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 previous delivered window. Thus, SEARCH would compute the normalized difference 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 previous delivered window is $2 + 4 + 8 + 16 = 30$. Here, the normalized difference between twice the previous 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 4.8 shows the congestion window as a function of time. The curve has an exponential growth (doubling) during slow start and the window. The window labeled 'x' is 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.

The theoretical underpinnings of this behavior can be quantified by integrating the area under the congestion window curve. During exponential growth, 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, 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 4.8 window 'y'), the area is computed in pieces, exponential up to the at capacity point and then constant to the end of the window.

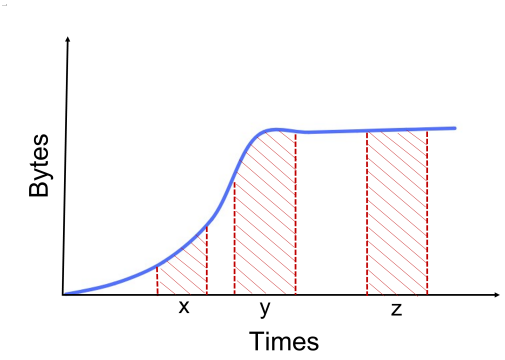


Figure 4.8: Window position based on congestion point.
The shaded areas represent SEARCH windows: 'x' is before capacity and window growth double each RTT, 'y' is at capacity where the left side of the window has doubled growth and the right side has plateaued, and 'z' has plateaued for the whole window.

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 4.8 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:

$$\begin{aligned} \text{prev\_delv} &= \left\lceil \frac{2^x}{\ln(2)} \right\rceil_{r-1-w}^{r-1} \\ \text{curr\_delv} &= \left\lceil \frac{2^x}{\ln(2)} \right\rceil_{r-w}^{r} \end{aligned} \tag{4.1}$$

If capacity has been reached before the window (e.g., Figure 4.8 window 'z'), $c < (r - w)$:

$$\underset{\text{c before window}}{\uparrow} \quad \overbrace{, 2^c, 2^c, \ldots, 2^c, \underbrace{2^c}_{r}}^{w}$$

So:

$$\begin{aligned} \text{prev\_delv} &= 2^c \cdot ((r-1) - (r-1-w)) \\ \text{curr\_delv} &= 2^c \cdot ((r) - (r-w)) \end{aligned} \tag{4.2}$$

If congestion occurs within the window (e.g., Figure 4.8 window 'y'), $(r-w) < c < r$:

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

So:

$$\begin{aligned} \text{prev\_delv} &= \left[\frac{2^x}{\ln(2)}\right]_{r-1-w}^{c} + 2^c \cdot (r-1-c) \\ \text{curr\_delv} &= \left[\frac{2^x}{\ln(2)}\right]_{r-w}^{c} + 2^c \cdot (r-c) \end{aligned} \tag{4.3}$$

With closed-form equations for both the current delivered bytes (curr_delv) and the previous delivered bytes (prev_delv), the normalized difference can be computed based on the RTT round relative to the at capacity round. Figure 4.9 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 -1 means one RTT before capacity will be reached. The y-axis is the normalized difference between current delivered bytes and previous delivered bytes. 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 once x $>$ 0 in the normalized difference values is approximately the same regardless of the window size.

While SEARCH seeks to detect the capacity point as soon as possible after reaching it, it must also avoid premature exit in the case of noise in the measurements on the link. The normalized difference reaches the 0.35 point almost exactly half-way between when the norm difference reaches the

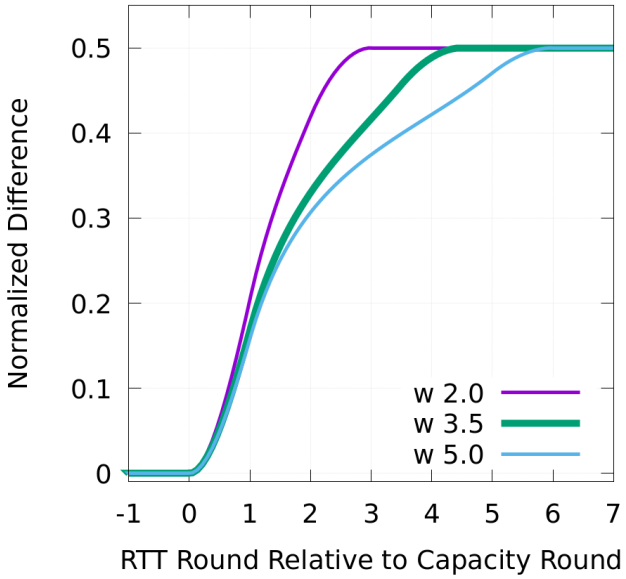


Figure 4.9: Normalized difference versus RTT round.

minimum (0) and the maximum (0.5). The 0.35 threshold value chosen does this and can be detected with 2 RTTs of reaching capacity.

Figures 4.10 and 4.11 depict the outcomes of SEARCH for exiting from slow start, presented as percentages, with varying window sizes over GEO and LEO satellite links, respectively. Each graph encompasses the results of 50 runs for each satellite network, highlighting the performance for different window sizes. The green point denotes the exit at the chokepoint (exit after reaching capacity and before packet loss), the blue point represents the late exit (exit after packet loss), and the red point indicates the early exit (exit before reaching the link capacity). As illustrated in these figures, for GEO satellite links, the 3.5RTT window size, coupled with a threshold of

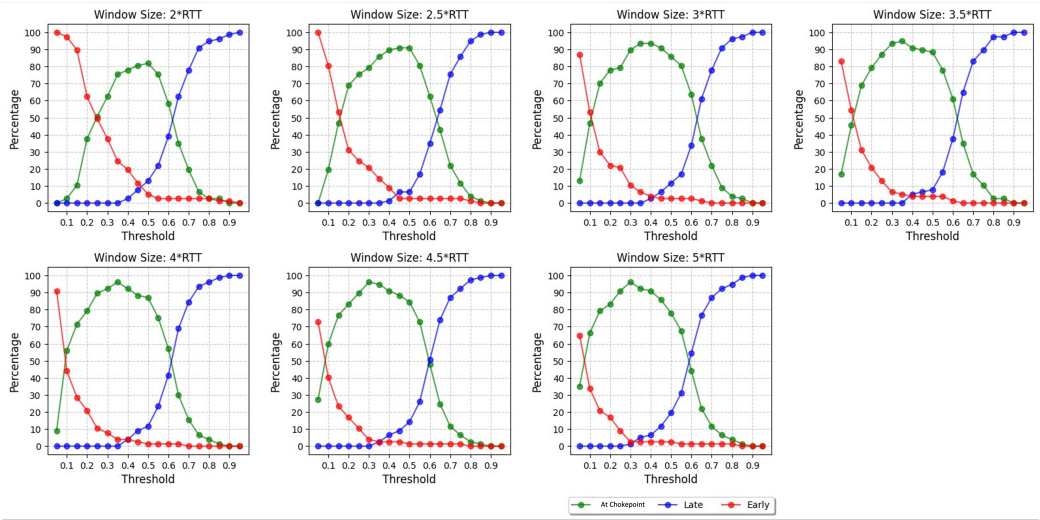


Figure 4.10: Sensitivity analysis of different window size and threshold over GEO links

0.35, attains the highest chokepoint exit percentage. This configuration effectively minimizes both premature and delayed exits. In the case of LEO satellite links, a positive outcome is observed, with a maximum chokepoint exit percentage and a minimized early exit percentage.

Overall, based on the above analysis, we recommend setting the default threshold to 0.35.

4.5.3 Number of Total 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

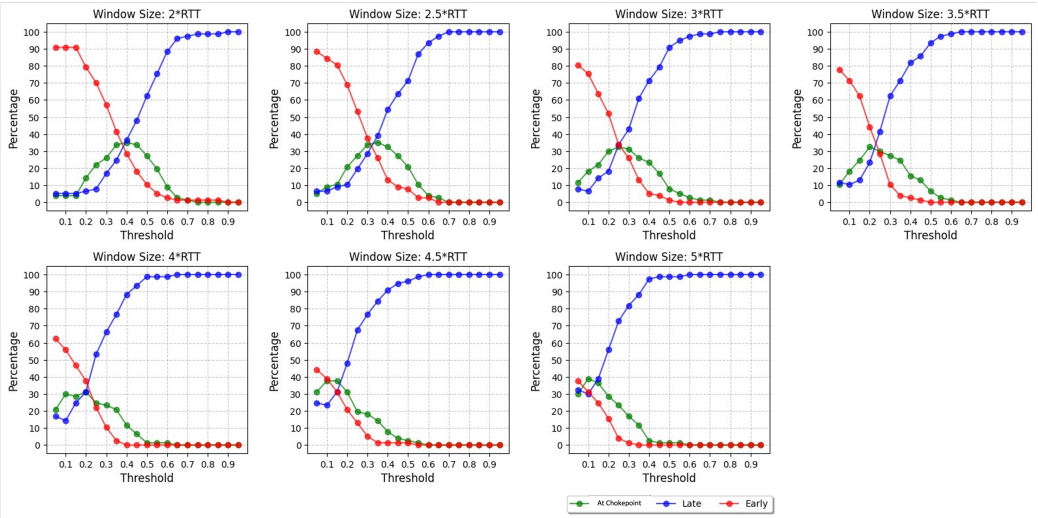


Figure 4.11: Sensitivity analysis of different window size and threshold over LEO links

more fidelity to actual delivered rates 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 aims to identify the impact of the number of bins used by SEARCH and the ability to exit slow start in a timely fashion. There are two distinct types of bins considered in this analysis: the number of bins for a window of data (Window bins) and the number of extra bins utilized when shifting back to cover the data after one RTT shift (Extra Bins).

Number of Window Bins

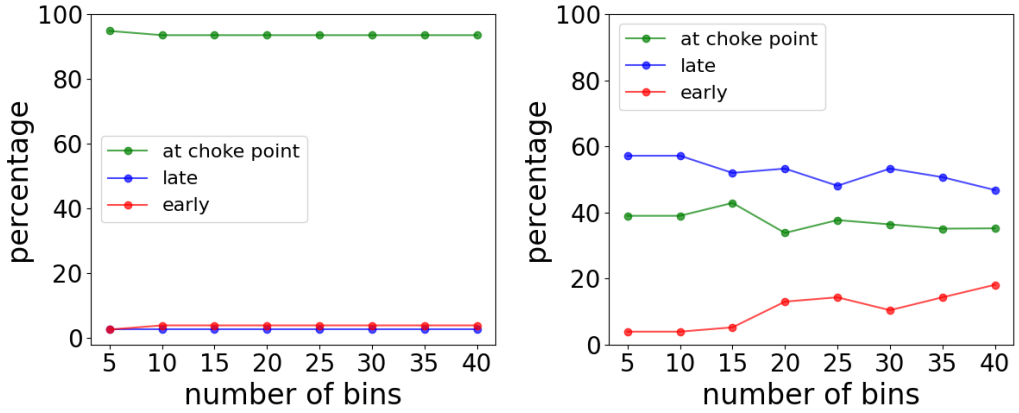
Using a window size of 3.5× the minimum RTT and a threshold of 0.35, we vary the number of bins for a window from 5 to 40 and observe the impact

on SEARCH's performance over 77 GEO TCP downloads, 77 LEO TCP downloads, and 55 4G LTE downloads. Figure 4.12 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 4.12a), the trendlines are all flat which indicates SEARCH performance is relatively stable no matter what bin size is chosen. For the LEO link (Figure 4.12b) and 4G LTE link (Figure 4.12c), the trendlines are still mostly flat, albeit the LEO link showing a slight upward trend in early and at 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.

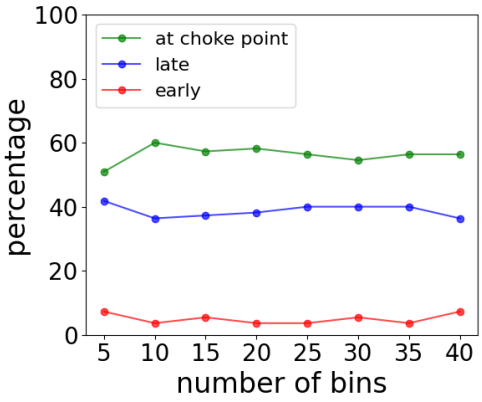
Number of Extra Bins

The extra bins serve an important role by retaining a record of previously delivered bytes. This storage enables SEARCH to perform a backward shift by an RTT, ensuring enough historical data is available to make accurate estimations of expected sent bytes. Balancing the number of additional bins is essential - too few hinder the effectiveness of SEARCH, while too many



(a) GEO

(b) LEO



(c) 4G LTE

Figure 4.12: Bin sensitivity analysis.

require more memory per flow. The following analysis aims to determine the most effective number of extra bins for improved performance.

Enough bins are required after a shift by the maximum RTT for effective performance. The Cumulative Distribution Function (CDF) of maximum RTT until exit time is illustrated in Figures 4.13a, 4.13c, and 4.13e for GEO, LEO, and 4G LTE links, respectively. Figures 4.13b, 4.13d, and 4.13f depict the corresponding initial RTT for each link type. Based on the maximum RTT CDF graphs, for GEO and LEO, the points are well-distributed. However, in the case of 4G LTE, while the majority of points fall between 0.2 and approximately 0.6 of the maximum RTT, there are some significant outlier values, with the maximum reaching 1.6. Also, a similar pattern is observed in the initial RTT distribution, where most values for GEO and LEO fall within a certain range, while 4G LTE exhibits outliers with notably higher values.

To determine the appropriate extra bin numbers based on max RTT, we employ the following formula:

$$\text{bin\_duration} = \frac{\text{W\_size}}{\text{window\_bin}} = \frac{3.5 \times \text{initial\_rtt}}{10} \tag{4.4}$$

$$\text{extra\_bin\_numbers} = \left(\frac{\text{max\_rtt}}{\text{bin\_duration}}\right) = \left(\frac{\text{max\_rtt}}{\text{initial\_rtt}}\right) \times \frac{10}{3.5} \tag{4.5}$$

Figure 4.14 presents the CDF of the ratio $\frac{\text{max\_rtt}}{\text{initial\_rtt}}$ and the CDF of extra bin numbers based on this ratio for GEO, LEO, and 4G LTE links. This ratio helps in determining the appropriate number of extra bins.

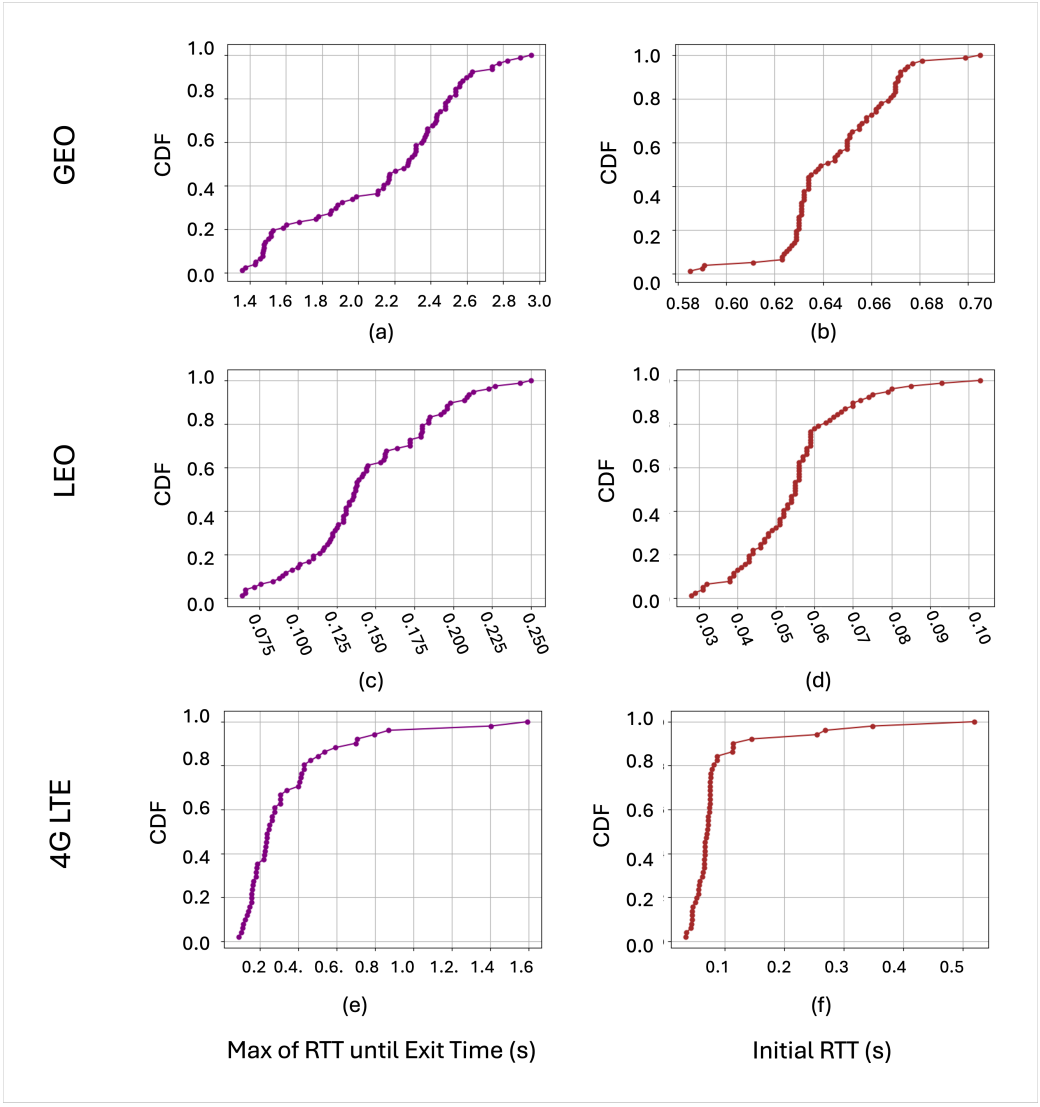


Figure 4.13: CDF of maximum RTT until exit time and initial RTT for GEO, LEO, and 4G LTE links

Based on Figure 4.14:

- For GEO links, $\max\left(\frac{\text{max\_rtt}}{\text{initial\_rtt}}\right) = 4.6$, suggesting the need for approximately 14 bins(4.14b).
- For LEO links, $\max\left(\frac{\text{max\_rtt}}{\text{initial\_rtt}}\right) = 4.4$, indicating the need for about 13 bins(4.14d).
- For 4G LTE links, $\max\left(\frac{\text{max\_rtt}}{\text{initial\_rtt}}\right) = 10.8$, suggesting the need for approximately 31 bins(4.14f).

However, In Figure 4.14f, 80% of the distribution of points are within 15 bins.

For additional analysis, refer to Figure 4.15. The x-axis represents the number of extra bins (ranging from 0 to 40, while the number of bins for the window remains constant at 10), and the y-axis shows the average percentage of exits at the chokepoint over 77 cases of GEO and LEO, and 55 cases of 4G LTE. The results indicate that using 15 bins is effective for GEO and LEO links, with a slight variation observed for 4G LTE, where the exit percentages stabilize after 15 bins.

In conclusion, based on these analyses, we recommend setting the number of extra bins to 15 and the total bins to 25.

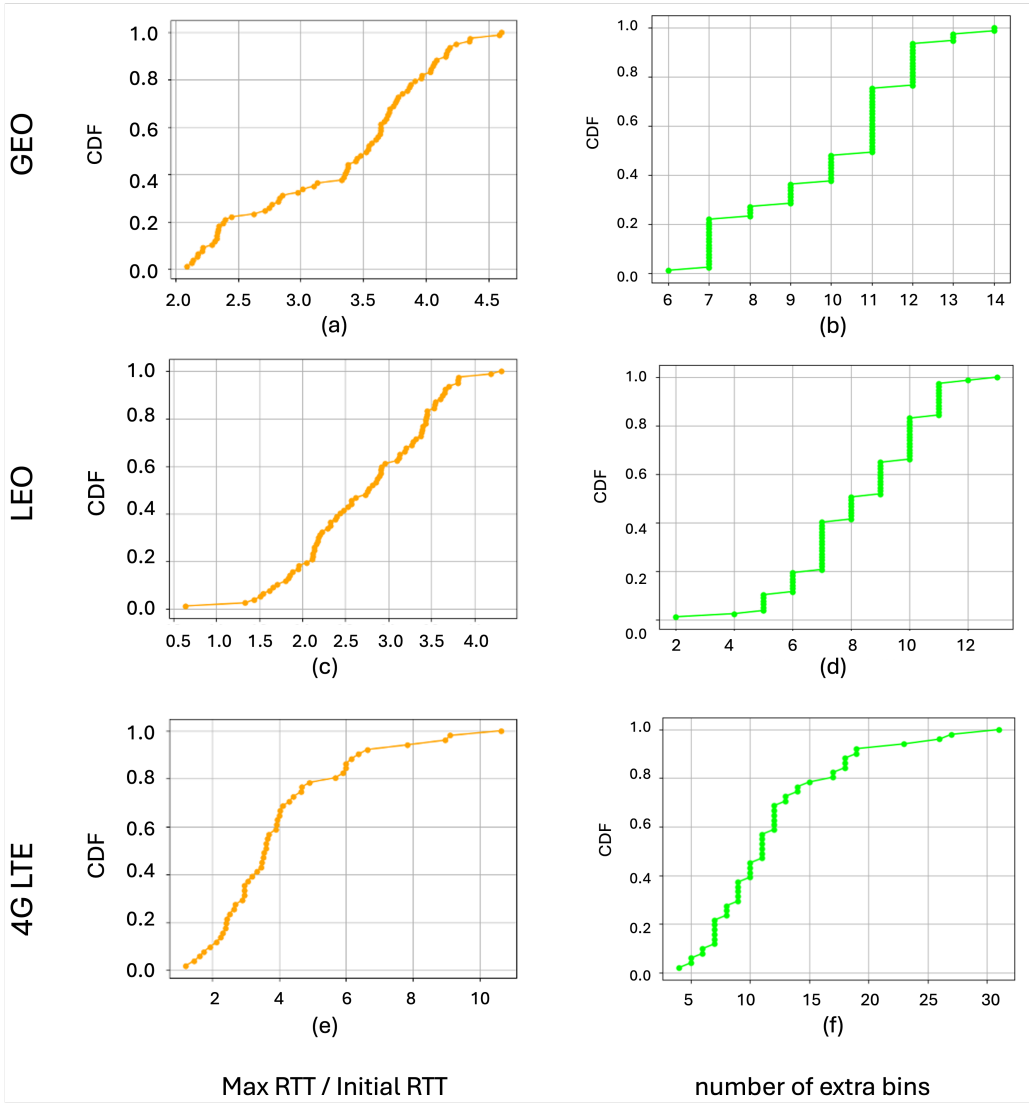
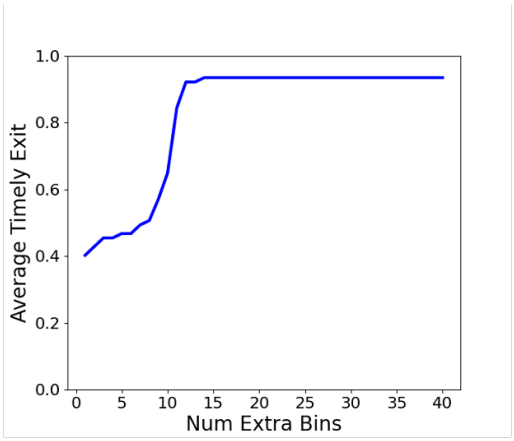
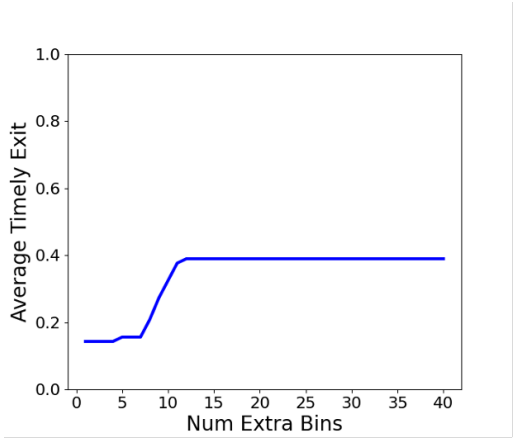


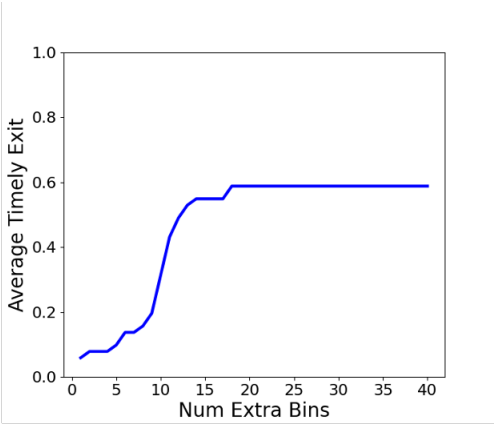
Figure 4.14: CDF depicting the ratio of maximum RTT to initial RTT and number of extra bins based on this ratio for GEO, LEO, and 4G LTE links



(a) GEO



(b) LEO



(c) 4G LTE

Figure 4.15: Average exit percentage at chokepoint for varying extra bin numbers.

Chapter 5

Evaluation

This chapter describes our measurement testbed, a heuristic to determine the at capacity time, and evaluation results.

5.1 Measurement Testbed

We set up a satellite (GEO and LEO), 4G LTE, and WiFi measurement testbeds to evaluate SEARCH over different wireless networks.

5.1.1 Satellites Testbed

Our satellite testbed includes GEO and LEO satellite networks, depicted in Figure 5.1. The client is a PC with an Intel i7-5820K 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 the 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 which 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's connection to the LEO satellite was established through an electronic phased array outdoor antenna. The LEO network delivers a peak downlink rate of approximately 100 Mb/s and showcases a lower minimum RTT of about 30 ms.

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

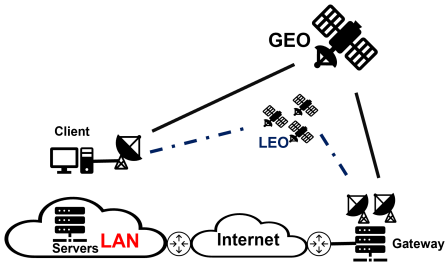


Figure 5.1: GEO and LEO satellite measurement testbed.

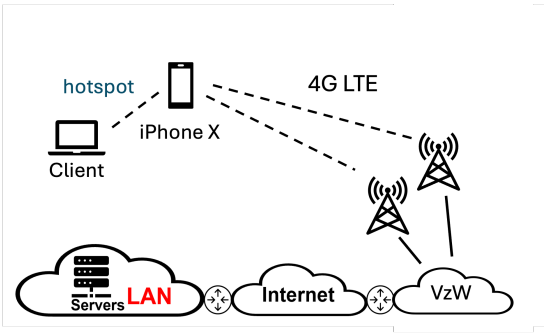


Figure 5.2: 4G LTE measurement testbed.

5.1.2 4G LTE Testbed

Our 4G LTE testbed shown in Figure 5.2 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 Intel Celeron 2957U CPU and 4 GB of RAM running Ubuntu version 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.

5.1.3 WiFi Testbed

Figure 5.3 depicts our WiFi testbed on the WPI campus. The campus, spanning 95 acres in an urban setting, has wireless coverage across virtually every building. The wireless infrastructure employs a controller drop-off design, routing all user traffic through one of six campus controllers via encrypted tunnels. Each Access Point (AP) is individually wired to the network, avoiding a mesh network setup.

To manage network traffic, users connecting through guest or open authentication methods face a bitrate cap to prevent them from negatively impacting critical academic or research activities. The network employs typical QoS measures, prioritizing voice, video, and control traffic based on traffic type classification.

For security, the network incorporates distributed denial-of-service protection, limiting packets per second for various protocols and blocking sessions that appear malicious. There is considerable, legitimate on campus use that competes with our experiments, and the urban environment of the campus can also introduce competing wireless signals and potential signal degradation.

Performance measurements were conducted using a fixed, dedicated on-campus server and a mobile laptop client for bulk TCP downloads, with different TCP configurations controlled at the server. The server, a PC with an Intel i5-8500 CPU @ 3GHz and 8 GB RAM, runs Mint-20.3 with Linux kernel version 5.10.79. The client is an Intel Core Ultra 7 155U×14 CPU and

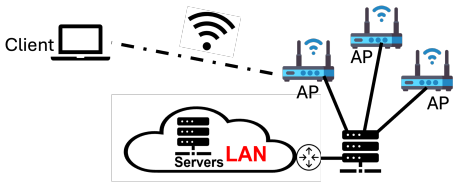


Figure 5.3: WiFi measurement testbed.

16 GB of RAM running Ubuntu version 22.04 and Linux kernel version 6.8.0. TCP Segmentation Offload (TSO) was disabled on the server, and Generic Receive Offload (GRO) was disabled on the client.

5.2 Determining when TCP has Reached Link Capacity

Determining when TCP has reached the link capacity is needed to assess our algorithm's performance over a TCP connection. Link capacity, often referred to as the bandwidth limit of a connection, is the maximum rate at which data can be transmitted over a network path. When TCP reaches this limit, it indicates that the network is being fully utilized, and further attempts to increase the data rate will not result in higher throughput but may instead lead to congestion, packet loss, or increased latency.

SEARCH is designed to exit the slow start phase when the congestion window has sufficiently grown, reaching the link capacity before encountering packet loss. We employ two different heuristics depending on the network's RTT characteristics to identify the at capacity point for an unknown link.

In high RTT networks, we use the one-way latency offset, recorded via Wireshark on the client, to determine when the downlink has been saturated. This method allows us to identify the point at which additional data flow starts to impact latency, indicating that the link is at capacity. For low RTT networks, SEARCH compares the median throughput to the bitrate at which the slow start phase exits. If the exit bitrate exceeds the median throughput, it suggests that the link has reached its capacity.

5.2.1 High RTT Networks

To identify the at capacity point for a link with high RTT, we leverage a heuristic utilizing the one-way latency offset, observed via Wireshark on the client [21].

At the initiation of data transmission, a low latency offset signifies a link operating below its congestion threshold. As the transmission progresses and the downlink saturates, the latency offset increases, indicating that the capacity limit has been attained and that it has led to elevated latency due to queuing.

Our heuristic for pinpointing the at capacity point involves the following steps:

1. Identify the moment when the latency offset first exceeds twice the minimum RTT. This occurrence signifies that the link capacity has been reached at some point in the past.

2. Trace backward from this identified point in time to determine when the latency offset drops below one-half the minimum RTT. This serves as our heuristic's definition of the at capacity point.

In Figure 5.4, we illustrate latency offset values recorded at the client via Wireshark during a TCP download from a server over a GEO satellite link, characterized by a minimum RTT of 600 ms. The x-axis represents time in seconds, and the y-axis depicts the latency offset in milliseconds, with the blue curve depicting the latency offset values.

For the initial 10 seconds, latency offsets remain low, well below half of the minimum RTT. Post the 10-second mark, the latency offset values exhibit a continuous increase until surpassing the 25-second mark. Notably, the first packet loss occurs around 17 seconds, marked by the detection of three duplicated ACKs in the Wireshark trace.

At approximately 12 seconds, the latency offset first exceeds twice the minimum RTT (1200 ms). To determine the at capacity point, we retrospectively examine preceding latency offsets before this instance. In this example, this analysis reveals that the offset drops below half the minimum RTT (300 ms) at around 10.5 seconds, serving as our identified at capacity point for this particular transfer.

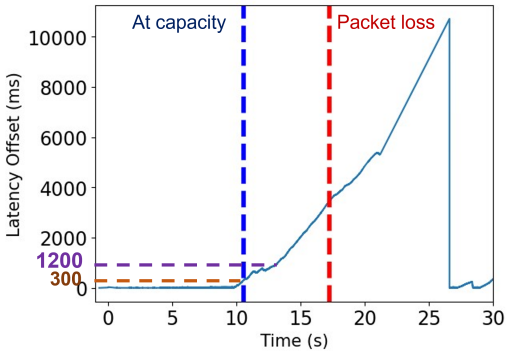


Figure 5.4: Determining the at capacity point using the latency offset for high RTT networks.

5.2.2 Low RTT Networks

The technique used for high RTT networks is not effective for low RTT environments, such as WiFi. To determine whether slow start successfully reached the link capacity in these networks, we compare the slow start exit bitrate to the median throughput. The slow start exit bitrate is calculated by dividing the congestion window by the RTT at the moment of slow start exit. If the exit bitrate exceeds the median throughput, it indicates that the link capacity was reached before slow start was exited.

Figure 5.5 illustrates this method using a sample from the WiFi dataset. The graph shows the CDF of throughput for one scenario. The vertical dashed light blue line represents the median throughput, while the orange dashed line indicates the slow start exit bitrate. Since the orange line is positioned after the median throughput, it confirms that the link capacity was reached, and slow start exited after hitting this capacity.

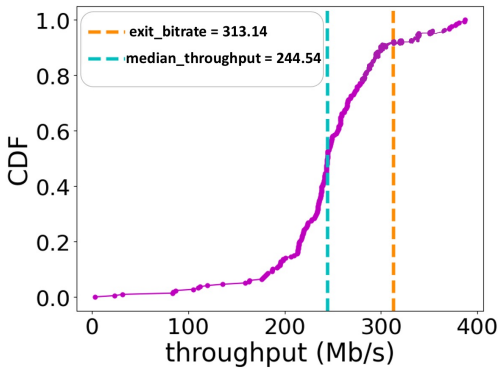


Figure 5.5: Determining the at capacity point using the median throughput for low RTT networks.

5.3 Results

In this section, we evaluate the performance of the SEARCH algorithm across different network environments, including GEO, LEO, 4G LTE, and WiFi links. Our algorithm, SEARCH, has been integrated into TCP CUBIC – the default TCP congestion control algorithm in Linux, resulting in a hybrid approach referred to as SEARCH_CUBIC. Therefore, our approach governs the slow start phase, while the congestion avoidance phase aligns with the standard CUBIC behavior. The iperf3 tool initiates downloads from the server to the client for each experimental run. The downloads are performed in Linux using the SEARCH_CUBIC algorithm (our approach) for SEARCH analysis. Additionally, default CUBIC configurations are employed for HyStart with HyStart enabled and HyStart disabled. To account for any time-of-day effects, where possible, we schedule 24-hour tests, running a series of SEARCH, CUBIC with HyStart enabled, and CUBIC with HyStart disabled

throughout the day.

5.3.1 SEARCH Performance Across Diverse Network Environments: Case Studies and Examples

Understanding how SEARCH behaves in these diverse scenarios is essential to demonstrating its effectiveness in exiting the slow start phase after full link capacity utilization and before packet loss occurs.

Figure 5.6 presents a representative example of SEARCH's performance during a single download on various links: GEO, LEO, 4G LTE, and WiFi. Across all graphs, the x-axis represents the time elapsed since the download began (in seconds).

In the left column of the graphs, the data transferred is shown in megabytes (Mbytes). The green line represents the current delivered bytes, while the blue line shows twice the previously delivered bytes (Estimated sent bytes). Vertical dashed red lines mark the moment of the first packet loss. In each graph, the green and blue lines start close together at the beginning of the download but gradually diverge before packet loss occurs. This divergence, where the current delivered bytes no longer match twice the previous delivered bytes, suggests that the link's capacity has been reached.

The right column of the graphs displays the normalized difference as a light blue trendline. Vertical blue dashed lines indicate when the link's capacity is reached (based on the heuristic described in Section 5.2), green dashed

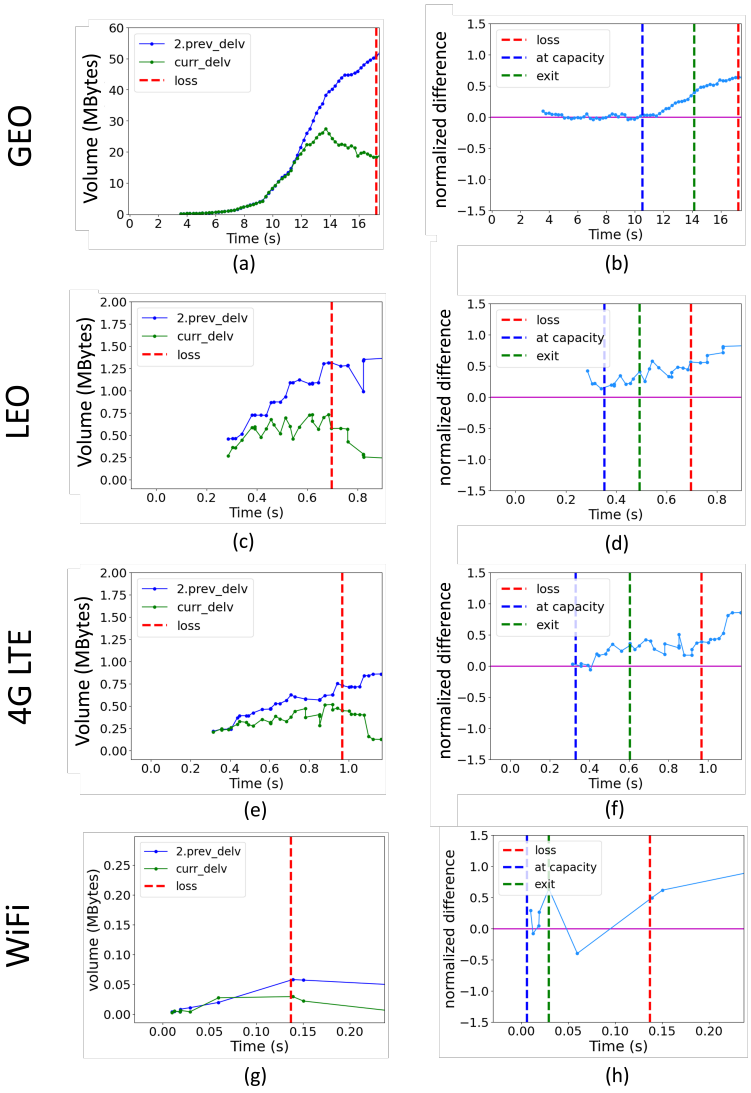


Figure 5.6: Bytes delivered and normalized difference over GEO, LEO, 4G LTE, and WiFi links.

lines show when SEARCH exits slow start, and red dashed lines mark the first packet loss. Across all four scenarios, SEARCH effectively exits the slow start phase after reaching capacity but before packet loss occurs. Comparing

the different links, the GEO link exhibits a much clearer indication of capacity being reached, as evidenced by lower noise (variation) in the normalized difference measurements. This suggests that while a lower threshold could benefit the GEO link, the higher noise levels in the LEO, 4G LTE, and WiFi measurements could lead to premature exits from slow start if the threshold were lowered.

5.3.2 Evaluation over High RTT Networks

As previously discussed, satellite networks (both GEO and LEO) and 4G LTE networks are examples of high RTT environments. To assess the performance of SEARCH in these scenarios, 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 at capacity point during a download is determined using the heuristics described in Section 5.2.1.

The results are summarized in Table 5.1: '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.

Table 5.1: SEARCH evaluation.

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

5.3.3 Evaluation over a Low RTT Network: WiFi

WiFi networks, typically characterized by low RTT, were evaluated to assess the performance of TCP CUBIC with the SEARCH algorithm. We compared its performance against other TCP configurations, including TCP CUBIC with HyStart enabled, TCP CUBIC with HyStart disabled, and TCP BBR (version 1.0), using the iperf3 tool across multiple download sessions.

The client initiated the download sessions sequentially with the server, using one of the following configurations: 1) HyStart enabled, 2) HyStart disabled, 3) BBR, or 4) SEARCH. Each download session lasted 3 seconds, followed by a 7-second pause. After completing one download with each configuration, the client paused for approximately 6 minutes to allow system stabilization before repeating the process. This cycle was repeated 200 times

over 24 hours, resulting in a total of 800 downloads per session. Each session was conducted at two different Received Signal Strength Indicator (RSSI) locations: one with strong signal strength and one with weak signal strength.

In WiFi networks, the Received RSSI is a critical indicator of the wireless connection quality. Higher RSSI values typically signify stronger signal reception and better network performance, while lower RSSI values may indicate weaker signal reception and potential connectivity issues.

To evaluate the wireless signal strength across various locations within our campus WiFi environment, we measured the RSSI at multiple points. Figure 5.7 charts the distribution of RSSI values superimposed on our campus map. Each pin represents a location where a TCP download was performed, with the RSSI first measured by the laptop's WiFi driver. Locations with strong signal strength are indicated in green, while locations with weak signal strength are marked in red. This visual depiction of RSSI across the campus allowed us to identify areas with strong and weak signal reception.

To better understand the relationship between RSSI and download performance, we analyzed the correlation between RSSI levels and throughput. Figure 5.8 presents these results, with the x-axis representing the RSSI measured by the client and the y-axis showing the throughput for a 3-second TCP download using the default Linux TCP settings. As expected, there is a general trend of higher RSSI values corresponding to increased throughput. However, significant variability in throughput at higher RSSI values suggests that the access point may be shared with other devices experiencing lower

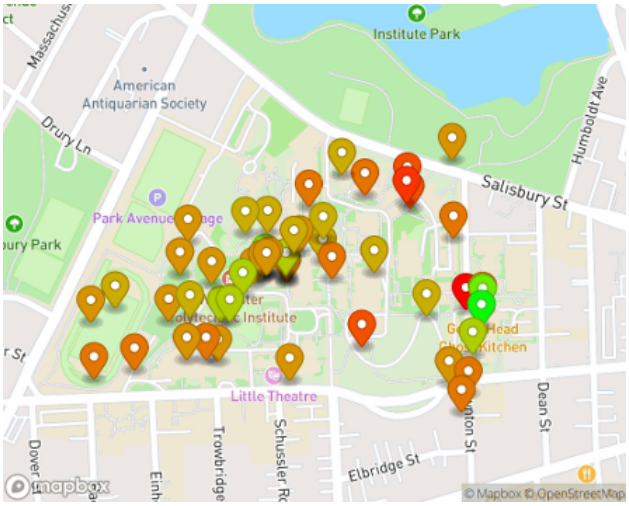


Figure 5.7: RSSI on WPI campus.

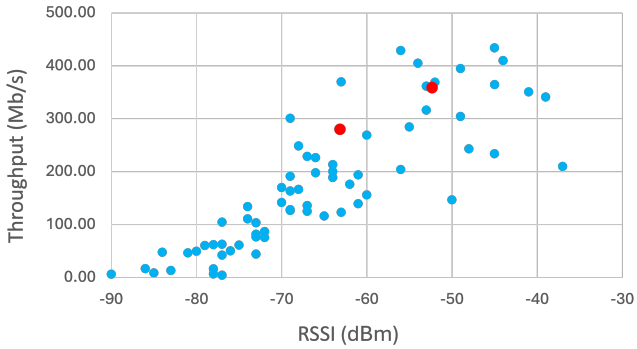


Figure 5.8: Relationship between throughput and WiFi RSSI across WPI

RSSI levels.

We identified two key locations on our campus for a closer examination, marked with red points in Figure 5.8: one with strong signal strength (-53dBm) and one with weak signal strength (-64dBm). These two locations provide a basis for evaluating the performance of TCP with the SEARCH algorithm in comparison to TCP with HyStart enabled, HyStart disabled,

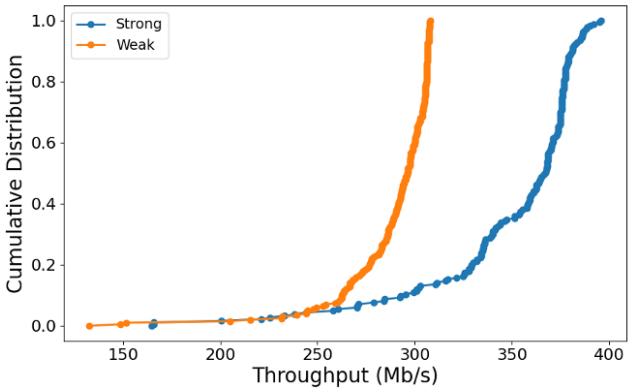


Figure 5.9: Distribution of median throughputs for strong and weak RSSI locations.

and BBR (version 1.0).

Figure 5.9 displays the cumulative distribution function of median throughput values aggregated from 200 cases per location over 24 hours, comparing the strong and weak RSSI locations. The x-axis represents throughput in Mb/s, while the y-axis shows the cumulative distribution. The CDFs reveal that the median throughput for strong RSSI locations is generally higher than that for weak RSSI locations. However, the throughput at strong RSSI locations exhibits more variability, with some overlap in the lowest 5% of the distribution.

To estimate when the congestion point is reached (the at capacity point), we use the median throughput after slow start as the bitrate at the congestion point, as described in Section 5.2.2).

Figure 5.10 illustrates the performance of the 200 downloads at the strong RSSI location, comparing the SEARCH algorithm (green), HyStart enabled

(red), HyStart disabled (purple), and BBR (blue).

Figure 5.10a presents the time required to download between 0 and 20 MB for each configuration. The x-axis represents the download size in Megabytes (MB), and the y-axis represents the download time in seconds. Each point reflects the average download time for 200 cases, measured at 100 KB intervals. The shaded areas indicate the standard error around the mean, while the horizontal dashed lines represent the average slow start exit times for each configuration, color-coded to match the respective lines. As shown, HyStart enabled exits slow start the earliest, leading to the longest download times. For the other configurations—SEARCH, BBR, and HyStart disabled—the download times are similar at first, and after a while, HyStart disabled achieving longer times.

Figure 5.10b displays the corresponding slow start exit times for each configuration, with mean values and standard error bars. HyStart enabled exits slow start the earliest at approximately 0.02 seconds, while HyStart disabled delays exit times to about 0.2 seconds. BBR exits startup at around 0.1 seconds, falling between the two HyStart configurations. TCP with SEARCH exits slow start at roughly 0.04 seconds, earlier than BBR but later than HyStart enabled.

Figure 5.10c shows the corresponding retransmissions (mean values with standard error bars). HyStart off shows the highest number of retransmissions, approximately 2600 packets, whereas the other configurations have substantially fewer retransmissions, all below 300 packets. HyStart on has

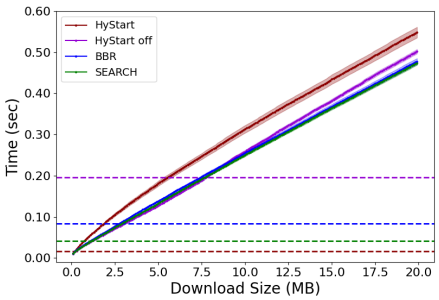
the fewest retransmissions, followed by SEARCH and then BBR.

These results suggest that enabling HyStart in TCP minimizes packet loss and retransmissions but at the cost of increased download time. Disabling HyStart reduces download times but significantly increases packet loss, as evidenced by the higher retransmission count. TCP with SEARCH strikes a favorable balance, achieving shorter download times while maintaining low packet loss. TCP BBR outperforms HyStart enabled in terms of download time and is better than HyStart disabled in terms of packet loss, though it still has higher download times and packet loss compared to SEARCH.

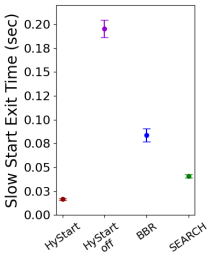
Figure 5.11 presents similar results for the 200 download cases conducted at the weak RSSI location. As expected, downloads at the weak RSSI location generally take longer than those at the strong RSSI location. However, the relative performance of the four configurations remains consistent. As with the strong RSSI location, SEARCH demonstrates the best overall performance by balancing download time and packet loss more effectively than the other configurations.

The results are summarized in Table 5.2. All values of capacity, loss, and exit 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.

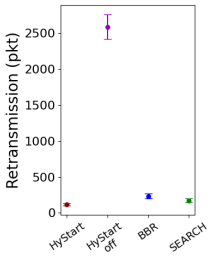
The results from these tests suggest that SEARCH reliably determines the exit point following congestion for all link types. The average slow start exit point is after capacity is reached but before packet loss. For all links,



(a) Time vs. bytes downloaded.

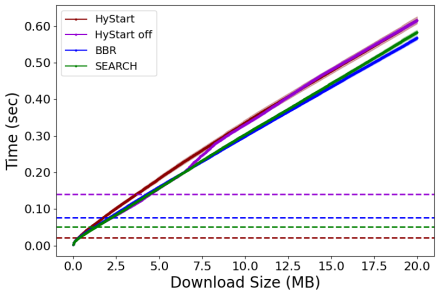


(b) Average slow start exit time.

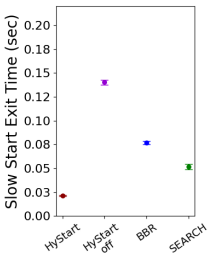


(c) Average retransmissions.

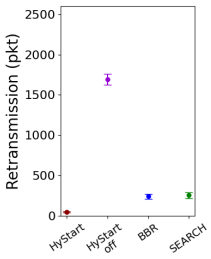
Figure 5.10: Performance metrics at strong RSSI location.



(a) Time vs. bytes downloaded.



(b) Average slow start exit time.



(c) Average retransmissions.

Figure 5.11: Performance metrics at weak RSSI location.

for over 95% of the connections, SEARCH did not exit slow start too early.

Table 5.2: SEARCH evaluation over WiFi.

Signal Strength	Capacity (s)	Loss (s)	Exit (s)	Timely (%)
Strong	0.01 (0.004)	0.77 (0.44)	0.04 (0.02)	95.2
Weak	0.02 (0.004)	0.68 (0.37)	0.04 (0.03)	99.0

5.3.4 Comparative Analysis of TCP Exit Strategies Across All Networks

In our comparative analysis presented in Table 5.3, we compare the performance of TCP with SEARCH compared to TCP with HyStart on (the Linux default) and TCP with HyStart off (i.e., traditional TCP). '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.

Note, 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 quickly, but TCP always overshoots the at capacity point, exiting slow start only when it encounters packet loss. When HyStart is on, over GEO, LEO, 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. In WiFi networks, SEARCH performs well in both strong and weak RSSI conditions, with more than 90% of exits occurring at the chokepoint in both scenarios. In strong WiFi, SEARCH shows slightly fewer late exits and a few more early exits compared to weak RSSI, but overall, the algorithm maintains a low incidence of early exits. HyStart on also performs well in both strong and weak WiFi, but it tends to have more early exits compared to SEARCH.

Table 5.3: SEARCH, HyStart enabled, HyStart disabled over GEO, LEO, LTE links, and WiFi.

Network	Slow Start Method	Early Exit (%)	Late Exit(%)	Choke Point Exit (%)
GEO	SEARCH	3.9	2.6	93.5
	HyStart on	100	0	0
	HyStart off	0	100	0
LEO	SEARCH	3.9	57.1	39.0
	HyStart on	100	0	0
	HyStart off	0	100	0
4G LTE	SEARCH	3.6	36.4	60.0
	HyStart on	100	0	0
	HyStart off	0	100	0
WiFi Strong RSSI	SEARCH	4.8	4.2	91.0
	HyStart on	10.8	0	89.2
	HyStart off	0	100	0
WiFi Weak RSSI	SEARCH	1.0	8.5	90.5
	HyStart on	4.8	0	95.2
	HyStart off	0	100	0

Chapter 6

Conclusion

TCP congestion control is the de facto transmission control protocol on the Internet. Traditional TCP slow start mechanisms, such as CUBIC with HyStart enabled, tend to exit slow start prematurely over many wireless networks, leading to underutilized capacity. Disabling HyStart, on the other hand, can result in overshooting the link capacity and increased packet loss. This thesis presented SEARCH, a novel algorithm designed to replace HyStart, addressing these issues by exiting slow start optimally and improving TCP performance across wireless networks.

SEARCH is designed to exit slow start after capacity is reached but before packet loss occurs. It achieves this by tracking delivered bytes with each ACK and comparing them to expected bytes from previous RTTs. This allows SEARCH to determine if the network has reached the network chokepoint. By implementing a smoothing window that covers 3.5 RTTs and aggregat-

ing delivered bytes to manage memory efficiently, SEARCH mitigates RTT fluctuations and memory constraints. This approach enables more precise capacity estimation, helping prevent both premature exits and overshooting.

Extensive evaluations of SEARCH across diverse networks, including GEO and LEO satellite links, 4G LTE, and WiFi, confirm its effectiveness. In hundreds of test downloads, SEARCH consistently exited slow start after reaching capacity but before packet loss, achieving rapid capacity utilization and minimizing packet loss. In a WiFi campus network with both strong and weak signals, SEARCH outperformed TCP CUBIC with HyStart by consistently reducing download times. It also demonstrated comparable download times to TCP CUBIC without HyStart and TCP BBR, but with lower packet loss than either.

Overall, the results highlight SEARCH as an effective and practical enhancement to TCP slow start, ensuring efficient utilization of link capacity while minimizing packet loss. Its adaptability across different network conditions, from high-latency satellite links to low-latency WiFi networks, makes it a promising extension for real-world applications. By improving bandwidth utilization and congestion management, SEARCH contributes significantly to more reliable and efficient TCP performance.

Chapter 7

Future Work

While SEARCH has demonstrated its effectiveness across various high-BDP network scenarios, several areas for further exploration remain.

Future work includes extending the evaluation of SEARCH to additional network types, such as 5G and Ethernet. This would provide deeper insights into its performance and assess its effectiveness across a broader range of environments. By testing SEARCH under different conditions, researchers could better identify its strengths and areas for additional optimization, expanding its applicability beyond the current set of networks.

Refining SEARCH to handle application-limited flows, where the sending rate is constrained by the application rather than the network, is another valuable direction. This enhancement could improve its robustness across diverse traffic patterns, ensuring accurate slow start exits even with irregular data transmission. Incorporating mechanisms to detect and adjust to

application-induced limitations would allow for more consistent performance in various use cases.

Another potential development is dynamically adjusting the congestion window size when there are insufficient bins for decision-making. This modification could further improve SEARCH's accuracy in highly variable conditions. Adapting the window size based on real-time delivery rates would help maintain consistent performance across changing network states, minimizing the risks of overshoot and premature exits.

Improving SEARCH's handling of extended inactivity periods—such as when applications stop sending data or server acknowledgments are delayed—could enhance its overall utility. Incorporating mechanisms to temporarily freeze the delivered byte window during inactivity, or resetting SEARCH would ensure accurate slow start exits, even with prolonged periods of low or no data transmission.

Another possible next step is implementing SEARCH in FreeBSD. By integrating SEARCH into this operating system, it would become available to a wider user base, enabling comprehensive performance evaluations in FreeBSD environments. This implementation would involve adapting SEARCH to FreeBSD's congestion control architecture, followed by rigorous testing to ensure compatibility and effectiveness.

These areas of future work present significant opportunities for advancing SEARCH, building on its foundation, and expanding its impact across diverse networking scenarios.

Bibliography

[1] Ghassan A. Abed and Ali M. Jasim. An Effective Practice to Approximating TCP Congestion Window Threshold in High Latency Connections. *Al-Iraqia Journal for Scientific Engineering Research*, 1, 01 2022.

[2] Mohammed M Alani. TCP/IP Model. *Guide to OSI and TCP/IP models*, pages 19–50, 2014.

[3] Mahdi Arghavani, Haibo Zhang, David Eyers, and Abbas Arghavani. SUSS: Improving TCP Performance by Speeding Up Slow-Start. In *Proceedings of the ACM SIGCOMM Conference, Australia*, pages 151–165, 2024.

[4] Indika AM Balapuwaduge and Frank Y Li. Cellular Networks: An Evolution from 1G to 4G. In *Encyclopedia of Wireless Networks*, pages 170–175. Springer, 2020.

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

[6] Lloyd Brown, Albert Gran Alcoz, Frank Cangialosi, Akshay Narayan, Mohammad Alizadeh, Hari Balakrishnan, Eric Friedman, Ethan Katz-Bassett, Arvind Krishnamurthy, Michael Schapira, et al. Principles for Internet Congestion Management. In *Proceedings of the ACM SIGCOMM Conference, Australia*, pages 166–180, 2024.

[7] Philipp Bruhn, Mirja Kuehlewind, and Maciej Muehleisen. Performance and Improvements of TCP CUBIC in Low-delay Cellular Networks. *Computer Networks*, 224, 2023.

[8] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based Congestion Control. *Communications of the ACM*, 60(2):58–66, 2017.

[9] Kevin Fall and Sally Floyd. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. *ACM SIGCOMM Computer Communication Review*, 26(3):5–21, 1996.

[10] Mirko Franceschinis, Marco Mellia, Michela Meo, Maurizio Munafo, et al. Measuring TCP over WiFi: A Real Case. In *1st workshop on Wireless Network Measurements (Winmee), Riva Del Garda, Italy*. Citeseer, 2005.

[11] Zoltán Gál, Gergely Kocsis, Tibor Tajti, and Robert Tornai. Performance Evaluation of Massively Parallel and High Speed Connectionless vs. Connection Oriented Communication Sessions. *Advanced Engineering Software*, 157(C), July 2021.

[12] Carlo Augusto Grazia. IEEE 802.11 n/AC wireless network efficiency under different TCP congestion controls. In *International Conference on Wireless and Mobile Computing, networking and Communications (WiMob)*, pages 1–6. IEEE, 2019.

[13] Carlo Augusto Grazia, Martin Klapez, and Maurizio Casoni. Bbrp: Improving TCP BBR Performance Over WLAN. *IEEE Access*, 8:43344–43354, 2020.

[14] Lingfeng Guo and Jack YB Lee. Stateful-TCP—A New Approach to Accelerate TCP Slow-Start. *IEEE Access*, 8:195955–195970, 2020.

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

[16] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[17] Mustafa Maad Hamdi, Sami AbdulJabbar Rashid, Mahamod Ismail, Mohammed A. Altahrawi, Mohd Fais Mansor, and Mohammed K. Abu-Foul. Performance Evaluation of Active Queue Management Algorithms

in Large Network. In *Proceedings of the IEEE 4th International Symposium on Telecommunication Technologies (ISTT)*, pages 1–6, 2018.

[18] Paul Heckbert. Fourier Transforms and The Fast Fourier Transform (FFT) Algorithm. *Computer Graphics*, 2(1995):15–463, 1995.

[19] Yu Huang and Matt Olson. HyStart++: Modified Slow Start for TCP. RFC 9406, RFC Editor, 05 2023.

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

[21] Maryam Ataei Kachooei, Jae Chung, Feng Li, Benjamin Peters, Joshua Chung, and Mark Claypool. Improving TCP Slow Start Performance in Wireless Networks with SEARCH. In *IEEE 25th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 279–288. IEEE, 2024.

[22] Maryam Ataei Kachooei, Zhao Pinhan, Feng Li, Jae Chung, and Mark Claypool. Fixing TCP Slow Start for Slow Fat Links. In *Proceedings of the 0x16 NetDev Conference*, October 2022.

[23] N.M. Kasoro, S.K. Kasereka, G.K. Alpha, and K. Kyamakya. ABCSS: A Novel Approach for Increasing the TCP Congestion Window in a Network. *Procedia Computer Science*, 191, Jan 2021.

[24] Lingang Li, Yongrui Chen, and Zhijun Li. Small Chunks can Talk: Fast Bandwidth Estimation without Filling up the Bottleneck Link. In *IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, 2023.

[25] Ralf Lübben. Forecasting TCP's Rate to Speed up Slow Start. *IEEE Open Journal of the Computer Society*, 3:185–194, 2022.

[26] Kouto Miyazawa, Saneyasu Yamaguchi, and Aki Kobayashi. Performance Evaluation of TCP BBR and CUBIC TCP in Smart Devices Downloading on Wi-Fi. In *IEEE International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan)*, pages 1–2. IEEE, 2020.

[27] W Stevens. RFC2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, 1997.

[28] Jiyang Ye, Bowen Huang, and Xiaolong Chen. An Improved Algorithm to Enhance the Performance of FAST TCP Congestion Control for Personalized Healthcare Systems. *Wireless Commun. and Mobile Computing*, 2021.

[29] Pinhan Zhao, Benjamin Peters, Jae Chung, and Mark Claypool. Competing TCP Congestion Control Algorithms over a Satellite Network. In *IEEE Consumer Communications & Networking Conference (CCNC)*, 2022.