

Implementation of the SEARCH Slow Start Algorithm in the Linux Kernel

NetDev 0x18

San Clara, CA, July 2024



Jae Chung

Feng Li

Benjamin Peters



Maryam Ataei

Mark Claypool



Joshua Chung

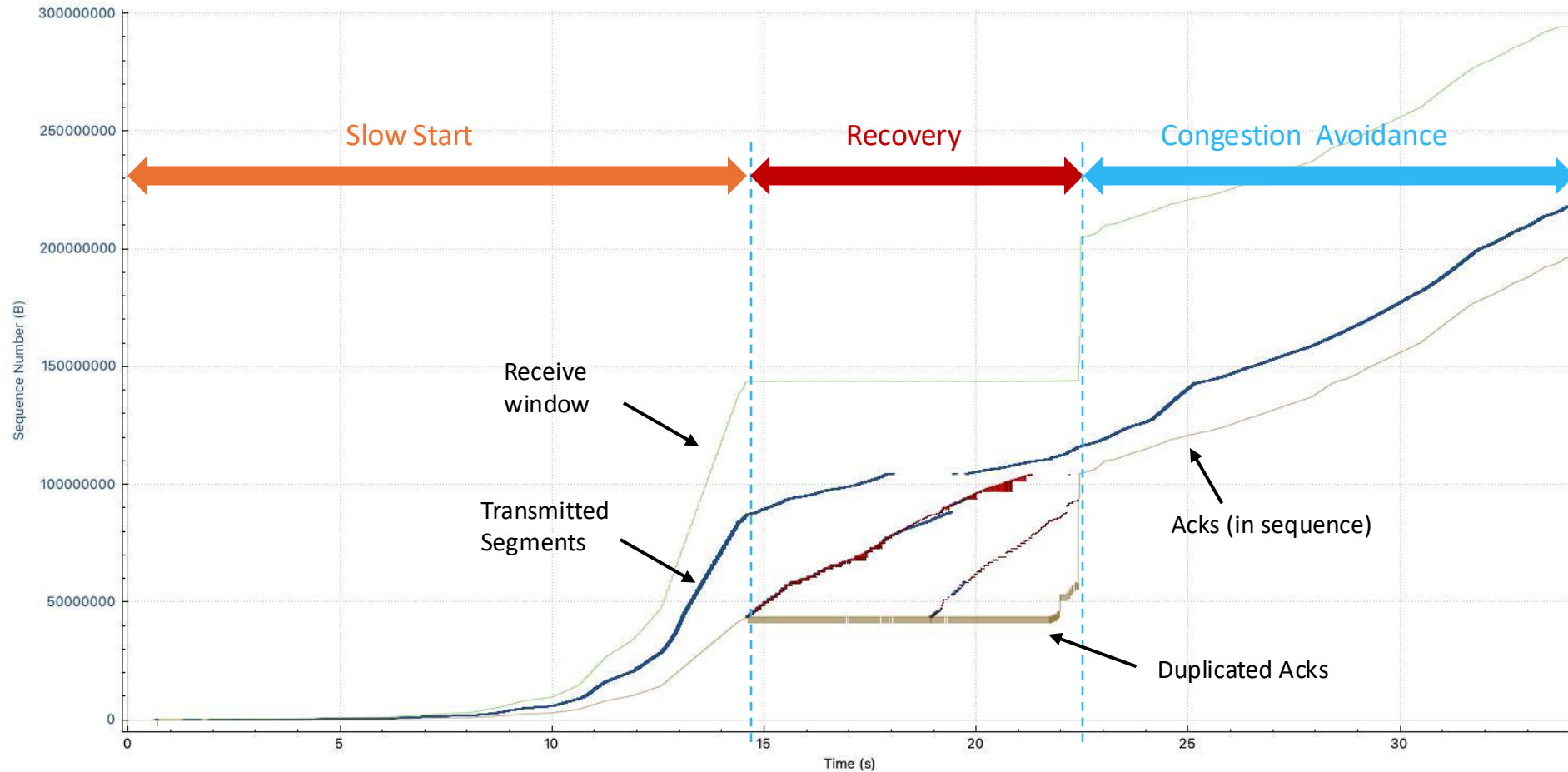
Outline

- Motivation and Problem Statement
- SEARCH Algorithm
- SEARCH Parameter Analysis
- Performance Evaluation
- Conclusion and Future work

Why Slow Start Important?

Case Study: TCP CUBIC on Broadband GEO Satellite (Capacity: 144Mbps, Duplex Propagation Latency = 600ms)

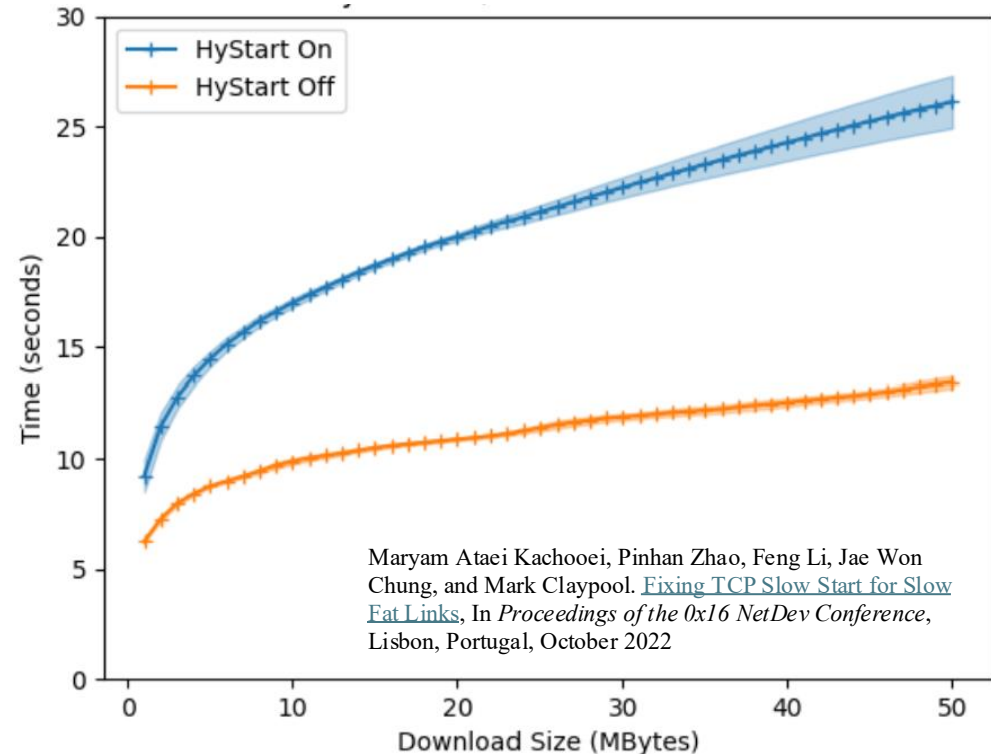
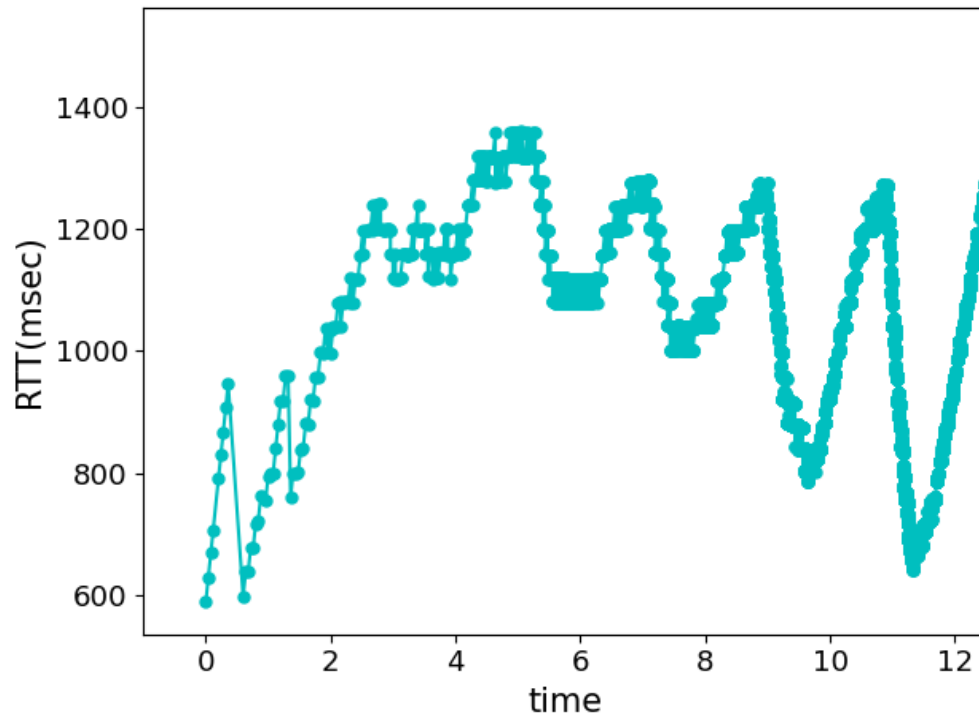
Time Sequence Graph of a TCP CUBIC w/o HyStart at the sender (bulk transfer)



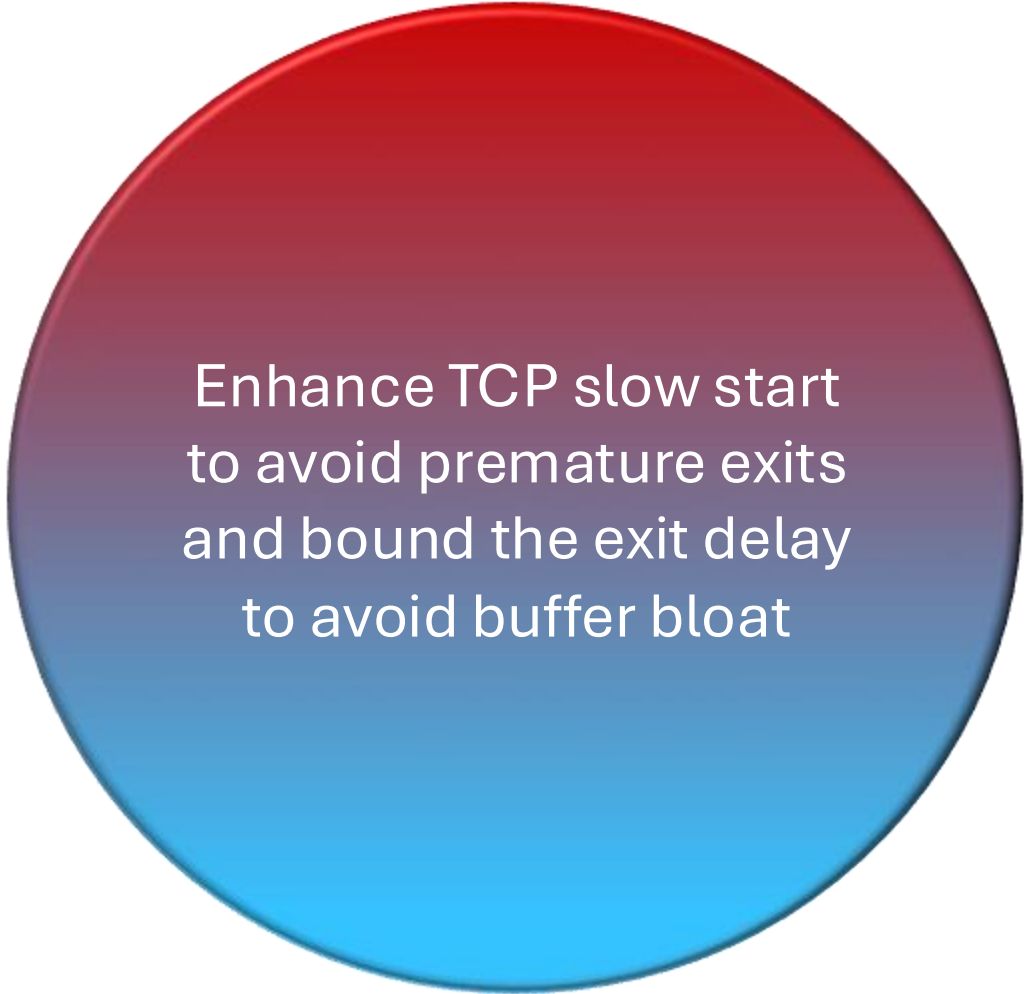
Why Slow Start Important ?

Case Study: TCP CUBIC on Broadband GEO Satellite (Capacity: 144Mbps, Base-RTT = 600ms)

- Large RTT variations confused TCP CUBIC HyStart (Linux Default) and caused **early slow start exit**
- The early exit resulted in bandwidth underutilization and **impacted object delivery time**



Mission Statement

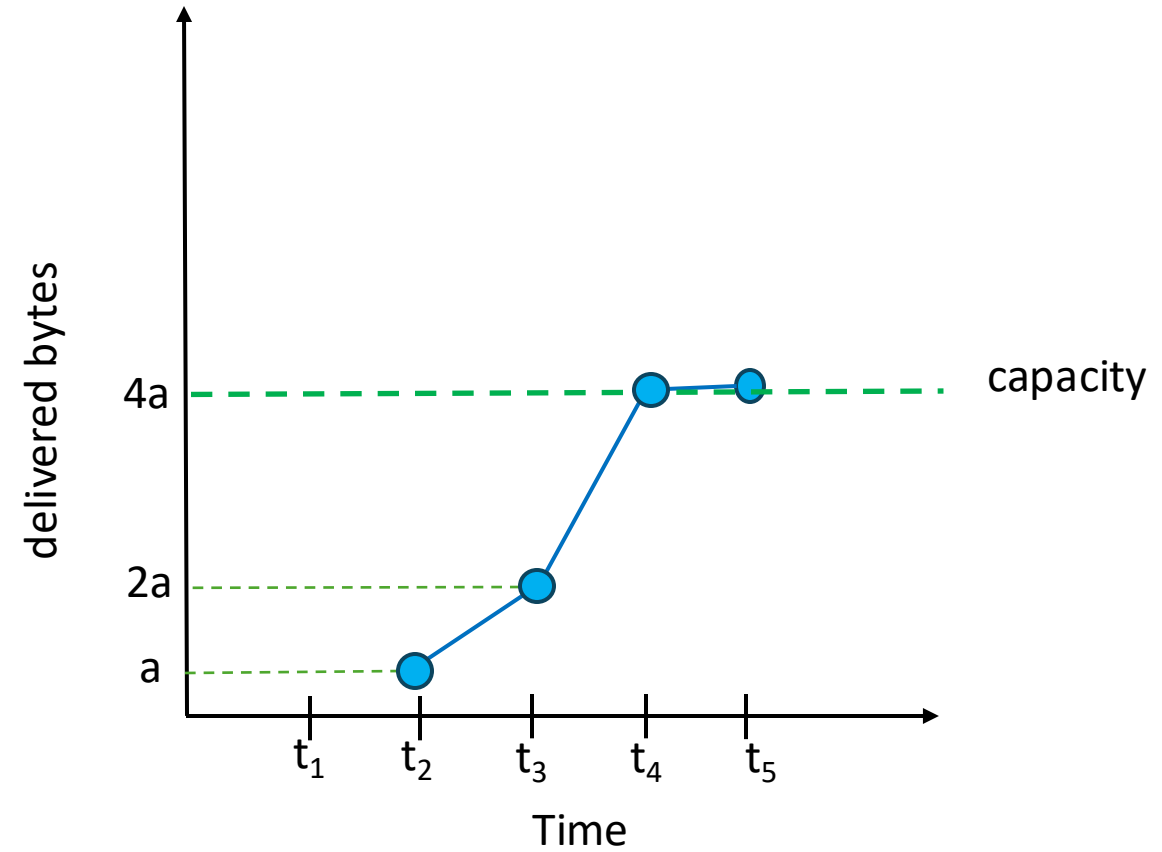
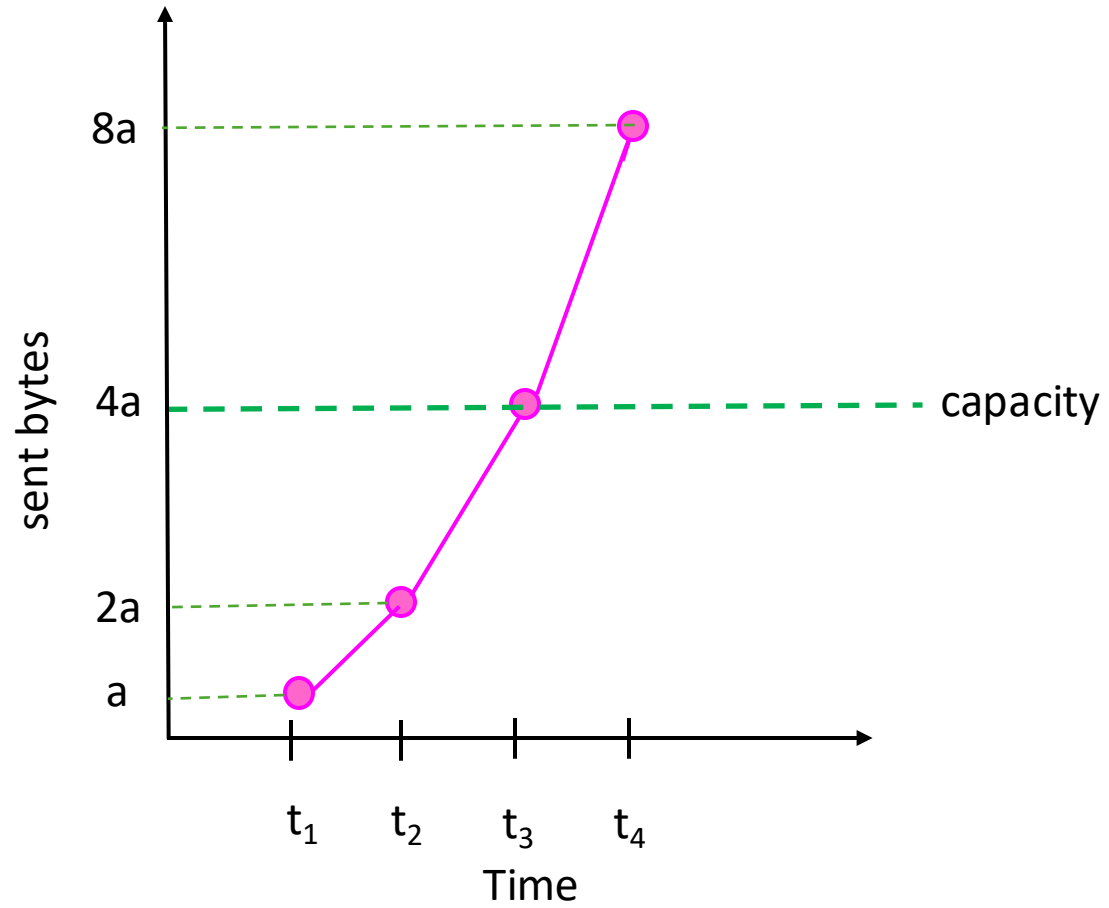


Enhance TCP slow start
to avoid premature exits
and bound the exit delay
to avoid buffer bloat

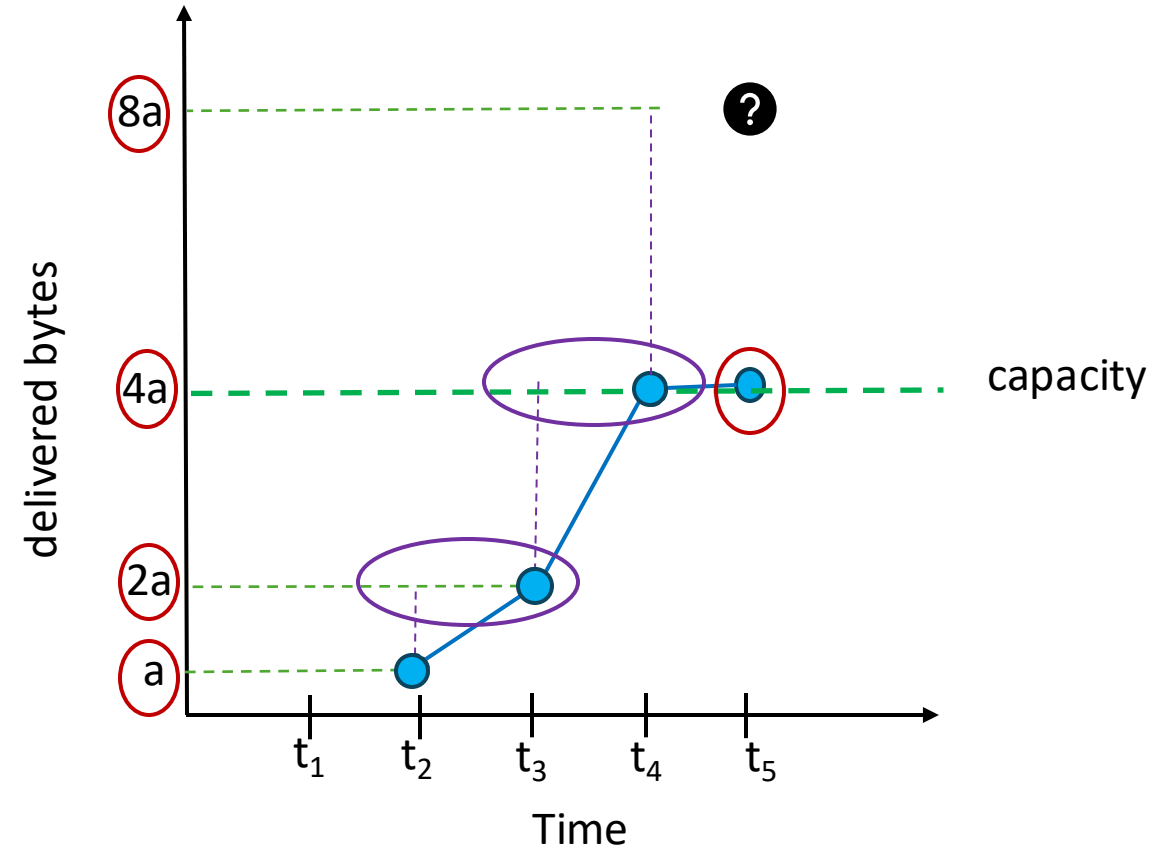
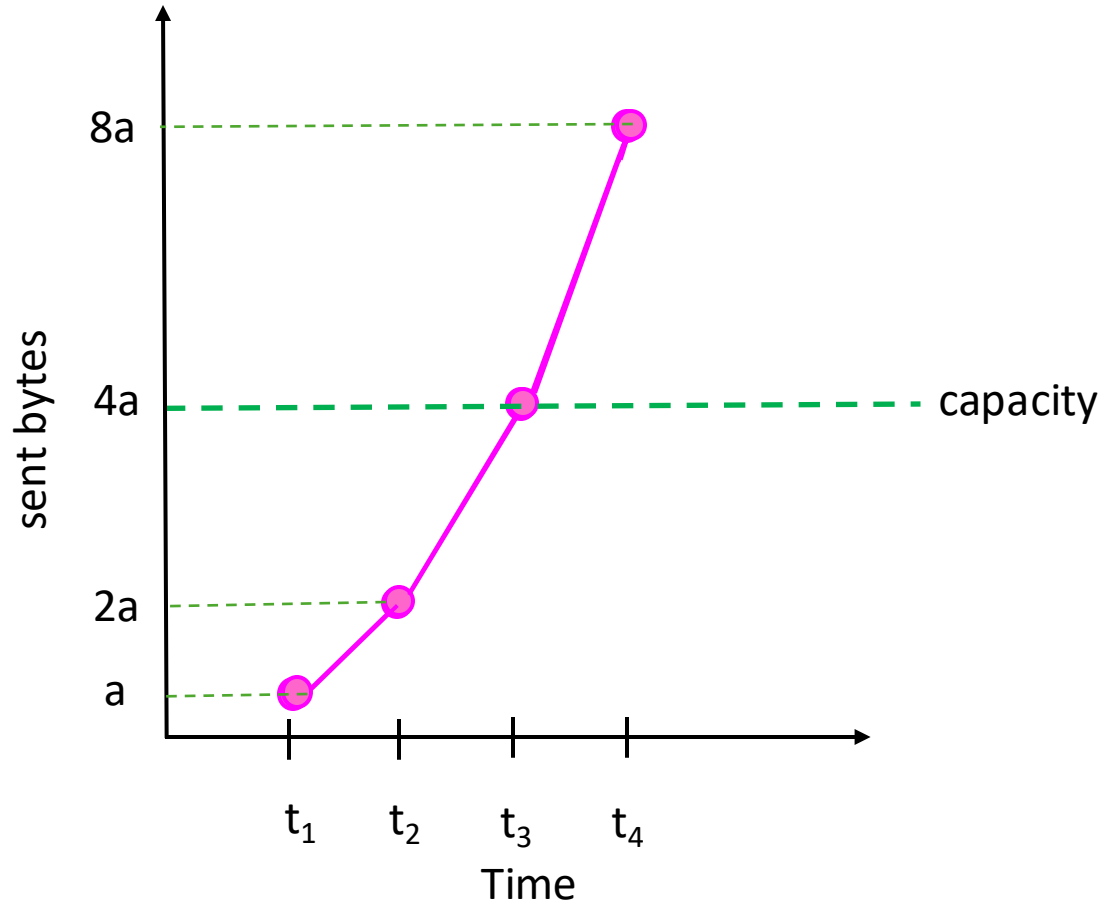
Outline

- Motivation and Problem Statement
- **SEARCH Algorithm**
- SEARCH Parameter Analysis
- Performance Evaluation
- Conclusion and Future work

Slow start Exit At Right Chokepoint (SEARCH)



Slow start Exit At Right Chokepoint (SEARCH)



estimated sent = $2 \times$ prev delv

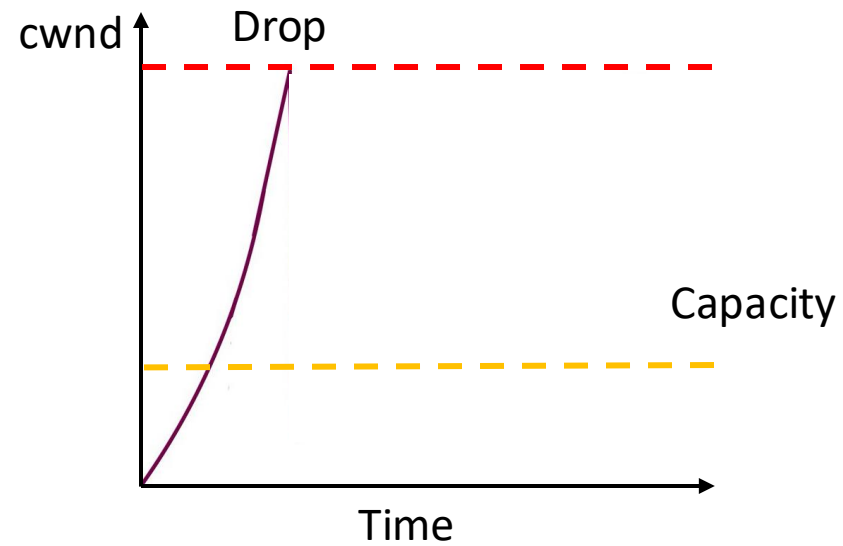
diff = estimated sent – curr delv

normalized diff = diff / estimated sent

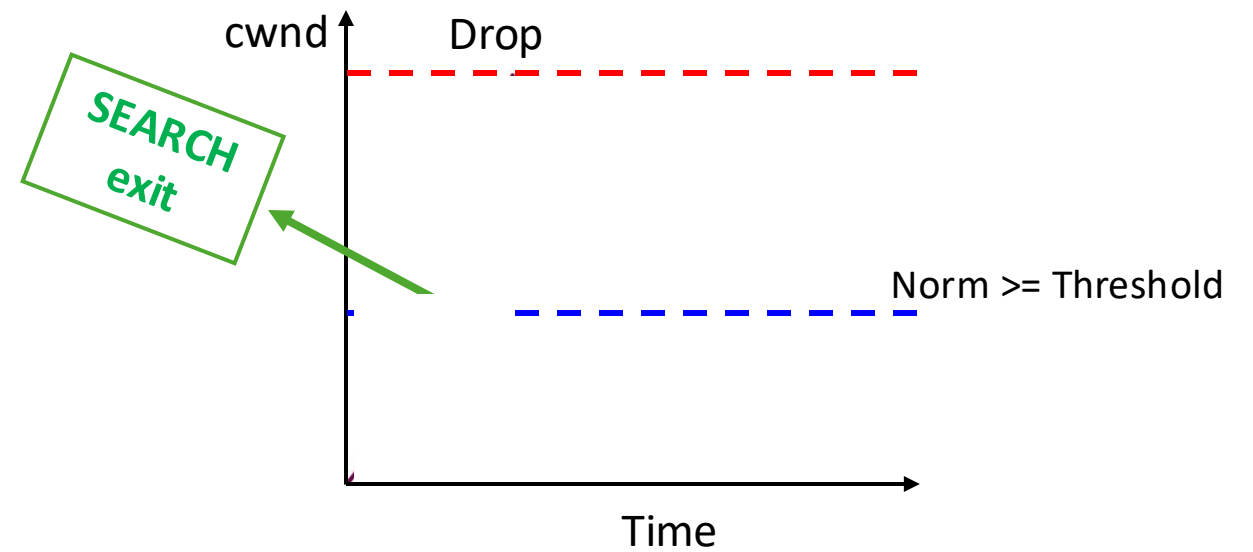
if (normalized diff \geq predefined threshold):
exit safely from slow start

SEARCH Behavior

TCP Default

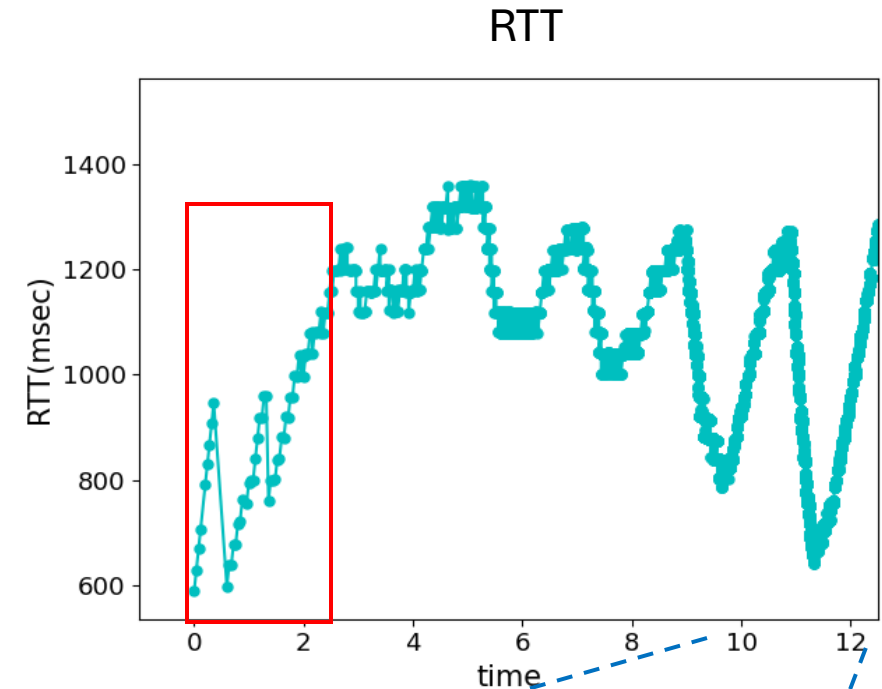


TCP with SEARCH



Challenges

- **Variation of RTTs**
 - RTT increase may not be caused by congestion on forward link.
 - Attributable to uplink acknowledgments on backward link.
- **Limited memory on server**
 - Memory allocated per flow
 - Unable to store history for each received ACK



window



SEARCH Algorithm 2.0

Algorithm 1 SEARCH 2.0 Algorithm for Linux

```
Parameters
1: WINDOW_FACTOR = 3.5
2: W = 10
3: EXTRA_BINS = 15
4: NUM_BINS = W + EXTRA_BINS
5: THRESH = 0.35

initialization( initial_rtt):
6: window_size = initial_rtt × WINDOW_FACTOR
7: bin_duration = window_size / W
8: bin[NUM_BINS] = {}
9: curr_idx = -1
10: prev_seq_num = 0
11: bin_end = now + bin_duration

ack_arrival( seq_num, rtt):
  // Check if passed bin boundary.
12: if (now > bin_end) then
13:   update_bins ()

  // Check if enough data for SEARCH.
14:   prev_idx = curr_idx - (rtt / bin_duration)
15:   if (prev_idx ≥ W and
16:     (curr_idx - prev_idx) ≤ EXTRA_BINS) then

     // Run SEARCH check.
17:     curr_delv = sum_bins(curr_idx - W, curr_idx)
18:     f =  $\frac{rtt \% bin\_duration}{bin\_duration}$ 
19:     prev_delv = sum_bins(prev_idx - W, prev_idx, f)
20:     norm_diff =  $\frac{2 \cdot prev\_delv - curr\_delv}{2 \cdot prev\_delv}$ 

21:     if (norm_diff ≥ THRESH) then
22:       sthresh = cwnd
23:     end if
24:   end if // Enough data for SEARCH.
25: end if // Each ACK.
```

```
// Update bins - more than one might have passed.
```

```
update_bins( seq_num ):
```

```
26: passed_bins =  $\frac{(now - bin\_end)}{BIN\_DURATION} + 1$ 
```

```
27: bin_end += passed_bins × bin_duration
```

```
28: for i = curr_idx to curr_idx + passed_bins do
```

```
29:   bin[i % NUM_BINS] = 0
```

```
30: end for
```

```
31: curr_idx += passed_bins
```

```
32: bin[curr_idx % NUM_BINS] = seq_num - prev_seq_num
```

```
33: prev_seq_num = seq_num
```

```
// Add up bins, interpolating fraction end bins (default is 0).
```

```
sum_bins( idx1, idx2, fraction = 0 ):
```

```
34: sum = 0
```

```
35: for i = idx1 + 1 to idx2 - 1 do
```

```
36:   sum += bin[i % NUM_BINS]
```

```
37: end for
```

```
38: sum += bin[idx1] × fraction
```

```
39: sum += bin[idx2] × (1 - fraction)
```

```
40: return sum
```

- Window size
- Number of Bins
- Threshold

Window Size – Parameter Selection (I)

- Windows Size Selection Consideration
 - Large enough to encapsulate meaningful link variation
 - Small enough to respond quickly when reaching link capacity
- Select window size with FFT
 - Convert measured RTT from Time domain to frequency domain
 - Choosing **3.5** as default, which is a balanced results for
 - GEO (3.33), LEO (3.33), WiFi (3.75) and LTE (2.8)

Number of Bins – Parameter Selection (II)

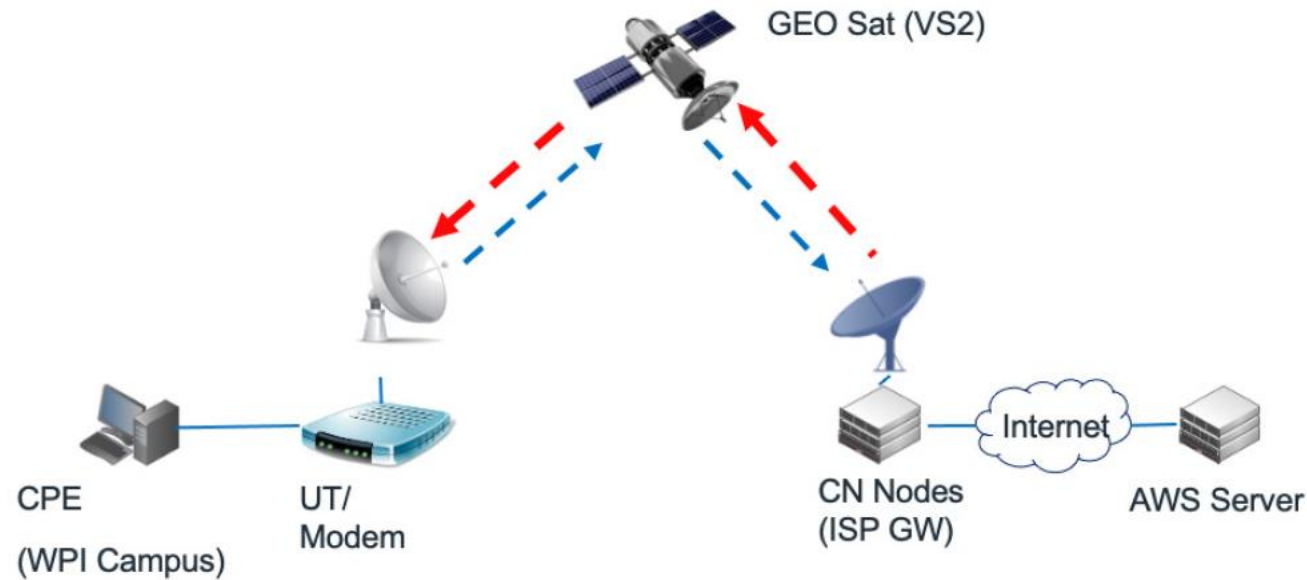
- *Bins* are introduced to reduce computational and memory load.
 - Impractical to track every ACK.
 - More bins provide more fidelity to actual delivered bytes total, reduce SEARCH convergence time. But more memory consumption.
- Through numbers cases analysis [KCC24], we choose **10 bins** which
 - Nearly identical performance as with more bins
 - Minimize "early exits" when closing to the maximum link capacity.

Threshold – Parameter Selection (III)

- Exit threshold demines when the difference between current delivered bytes and delivered bytes during previous RTT is large enough to exit from slow start.
- Based on analysis [KCC24], normalized difference would in range [0, 0.5).
 - Choose **0.35** as default value after tons of evaluation and analysis.
 - Detect congestion point in less 2 RTTs.

Outline

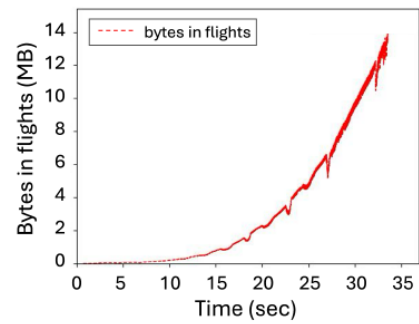
- Motivation and Problem Statement
- SEARCH Algorithm
- SEARCH Parameter Analysis
- Performance Evaluation
 - Evaluation w/ GEO by example
 - Evaluation w/ Wifi
- Conclusion and Future work



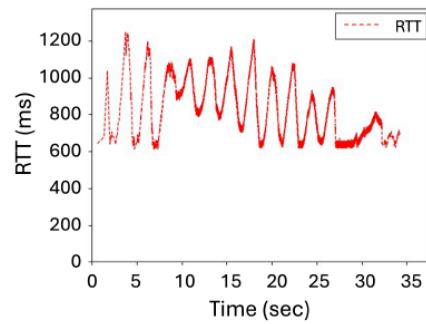
- AWS Server EC2 Instance w/ 32 GB RAM, Ubuntu 22.04 w/ 5.13.12 customized kernel for SEARCH support.
- CPE (Client) PC w/ 32GB RAM, Intel i7 – 5820 CPU, w/ Ubuntu 20.04 w/ 5.4.0 kernel.
- ISP Gateway supports up to 36MB buffer per flow with AQM capability.
- PEP are bypassed in this study.

Figure 1: GEO satellite measurement testbed.

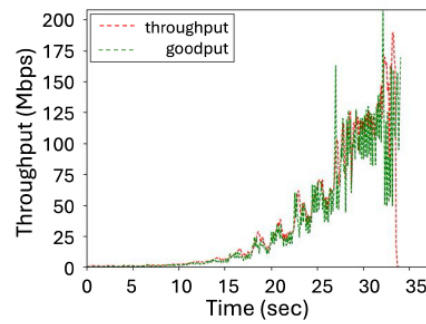
GEO Case Study by Examples



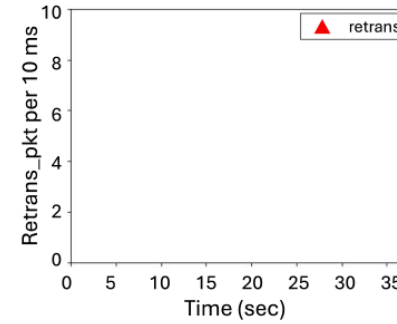
(a)



(b)

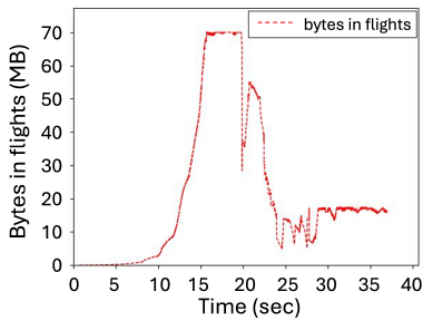


(c)

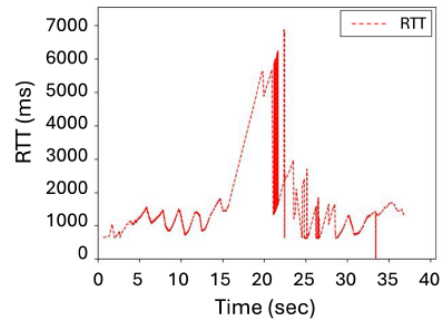


(d)

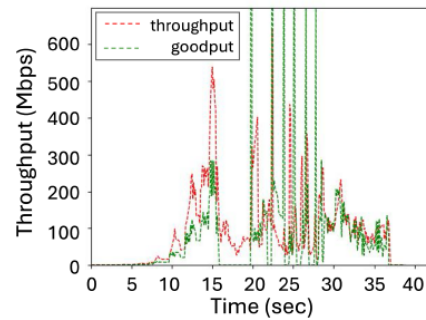
CUBIC w/ Hystart



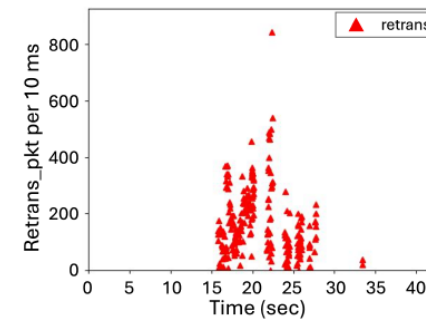
(e)



(f)

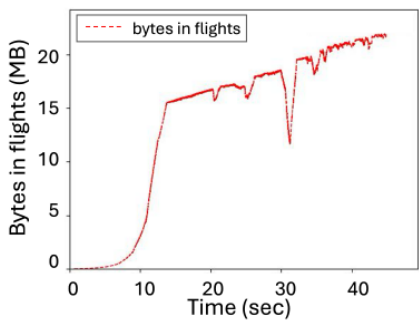


(g)

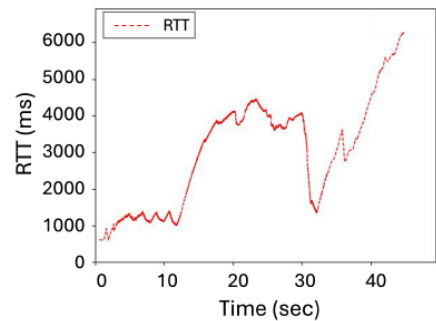


(h)

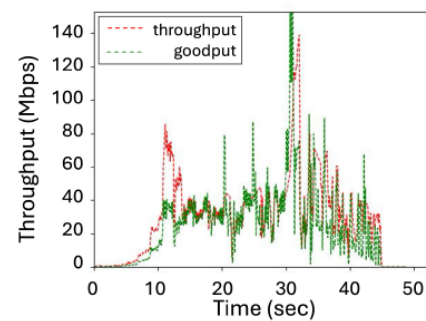
CUBIC w/o Hystart



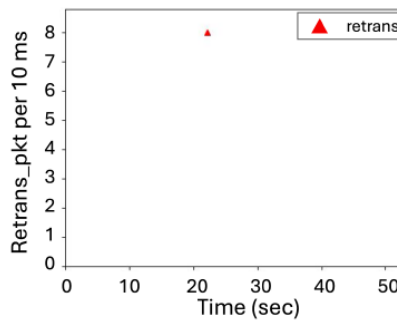
(i)



(j)



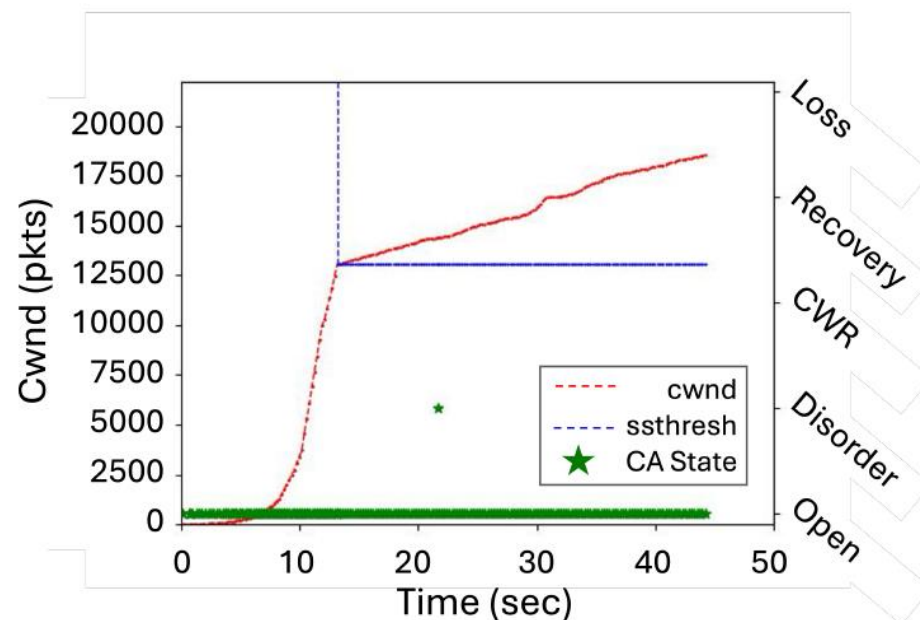
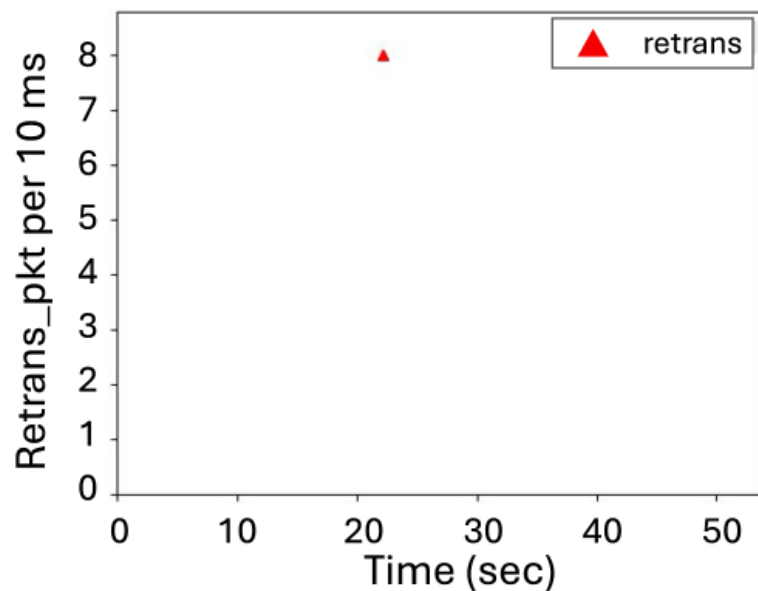
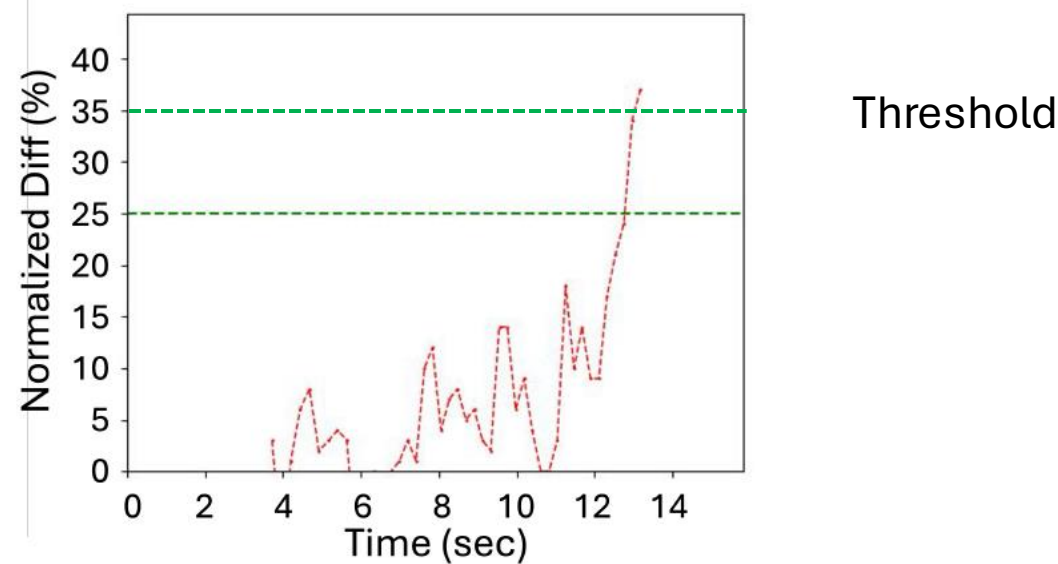
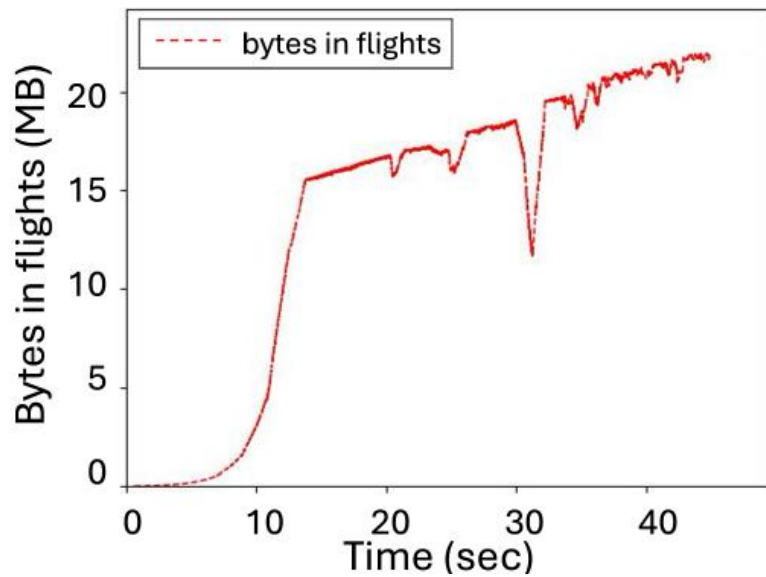
(k)



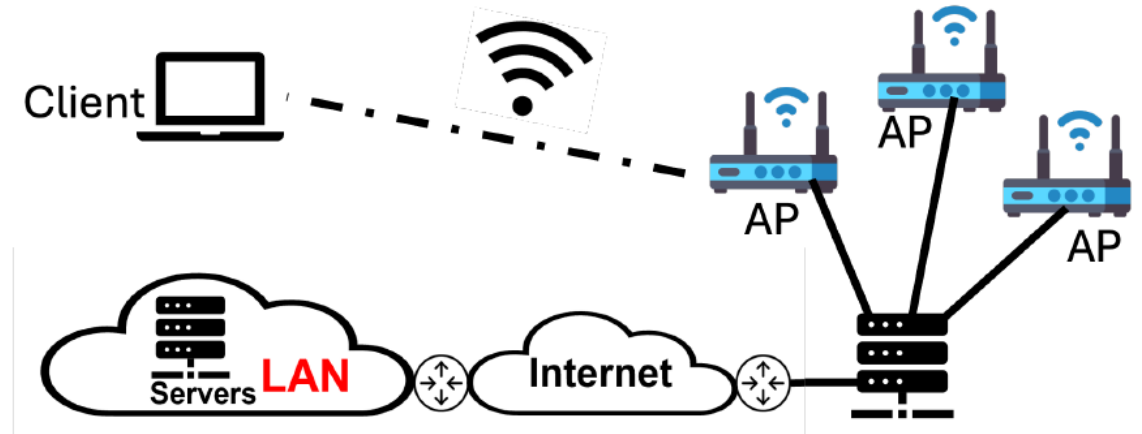
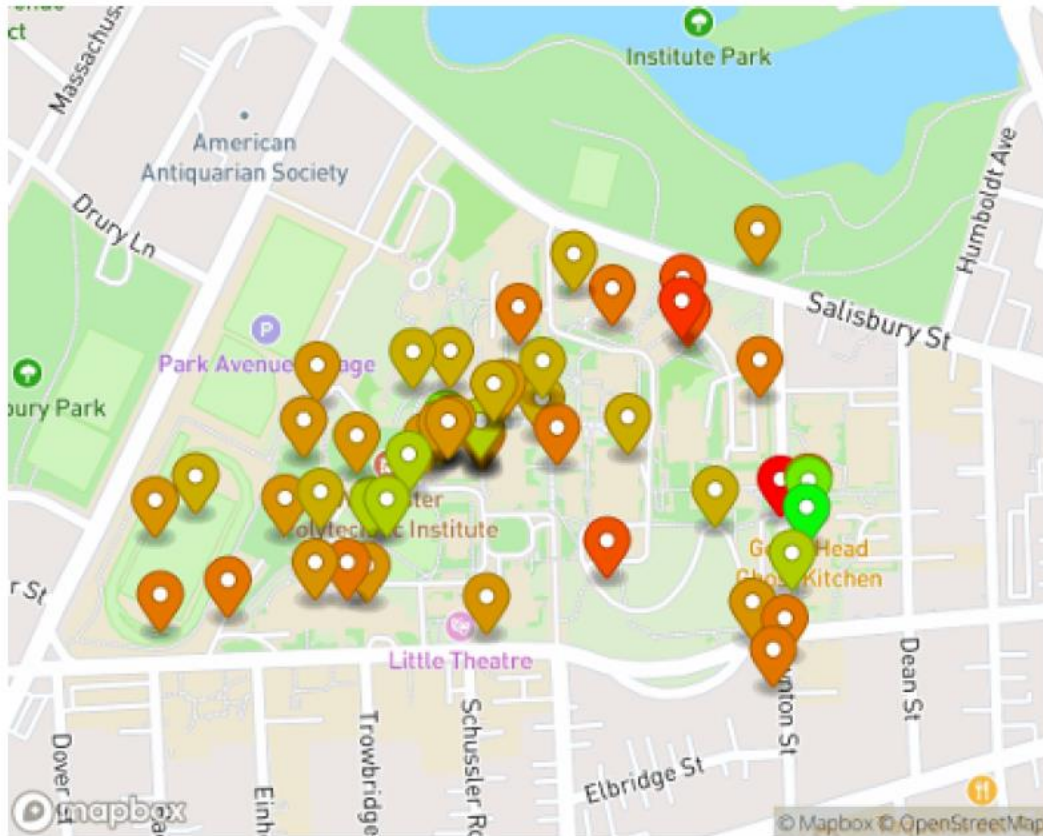
(l)

CUBIC w/ SEARCH

GEO Case Study by Examples



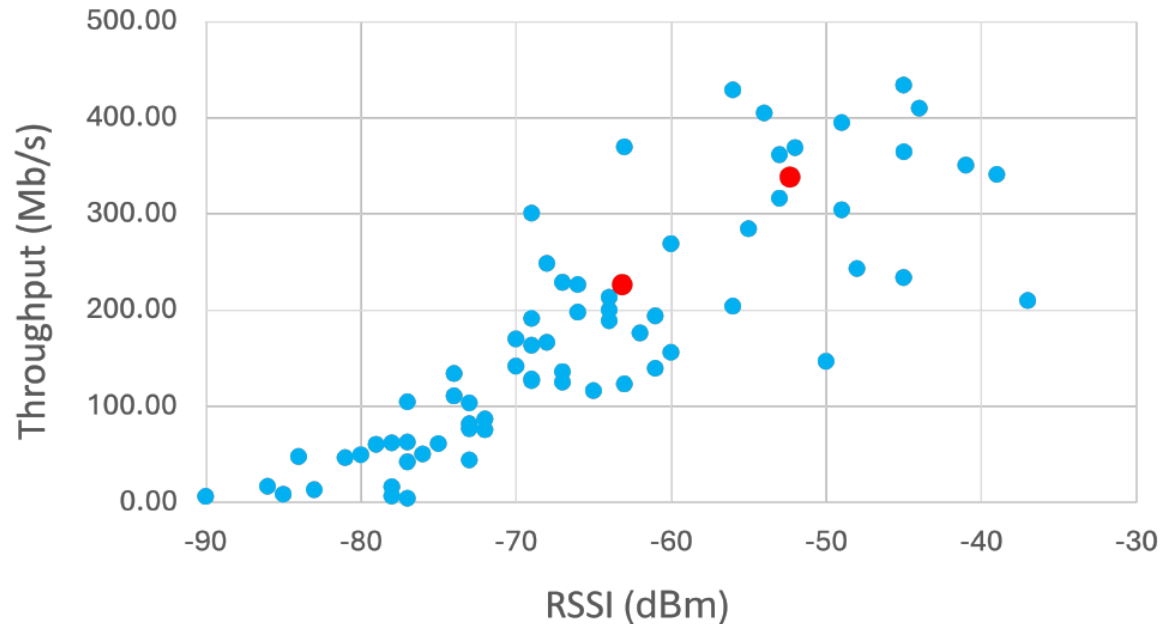
Wifi TestBed



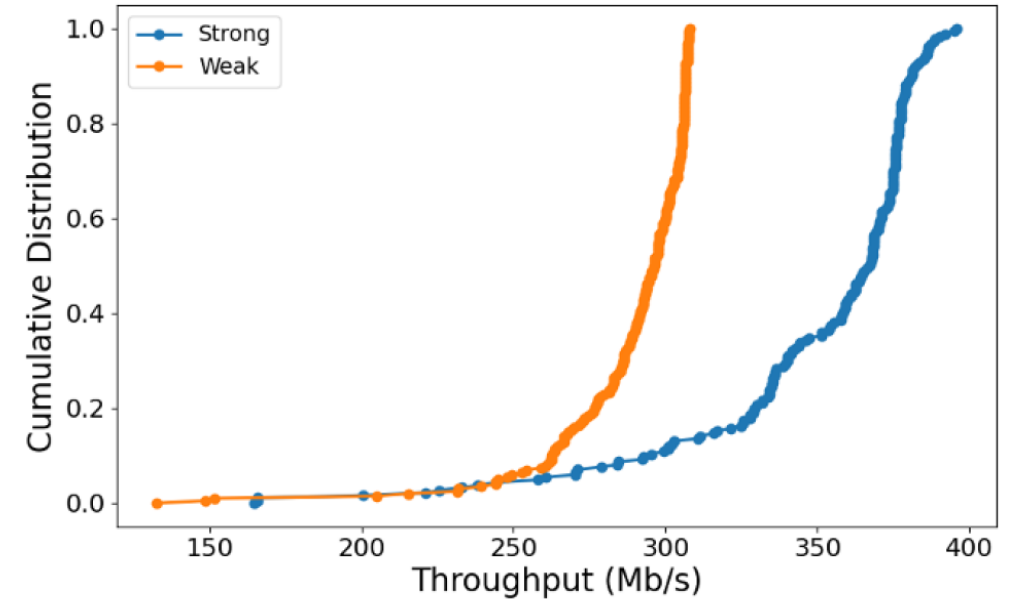
- Wired server connected to campus network
 - Intel i5-8500 CPU, 8GB RAM, mint 20.3 Linux kernel 5.10.79 w/ SEARCH module.
 - Client laptop Intel Core Ultra 7 CPU, w/ 16GB RAM, ubuntu 22.04 w/ 6.8.0 kernel.
- Each iteration has a 3 second long iperf3 session with four CCA configurations:
 - CUBIC w/HyStart, CUBIC w/o HyStart, BBRv1, and SEARCH
- Repeat 200 times in 24 hours.

WiFi Throughput and RSSI

* Throughput measured w/ Default TCP settings

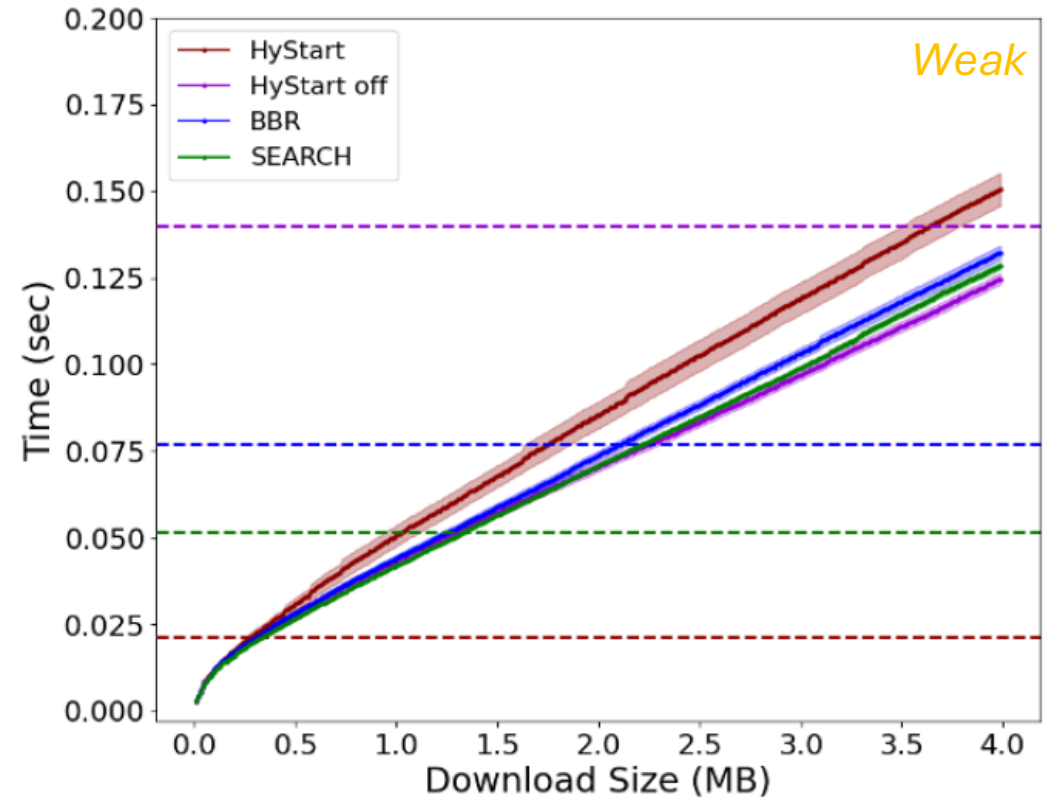
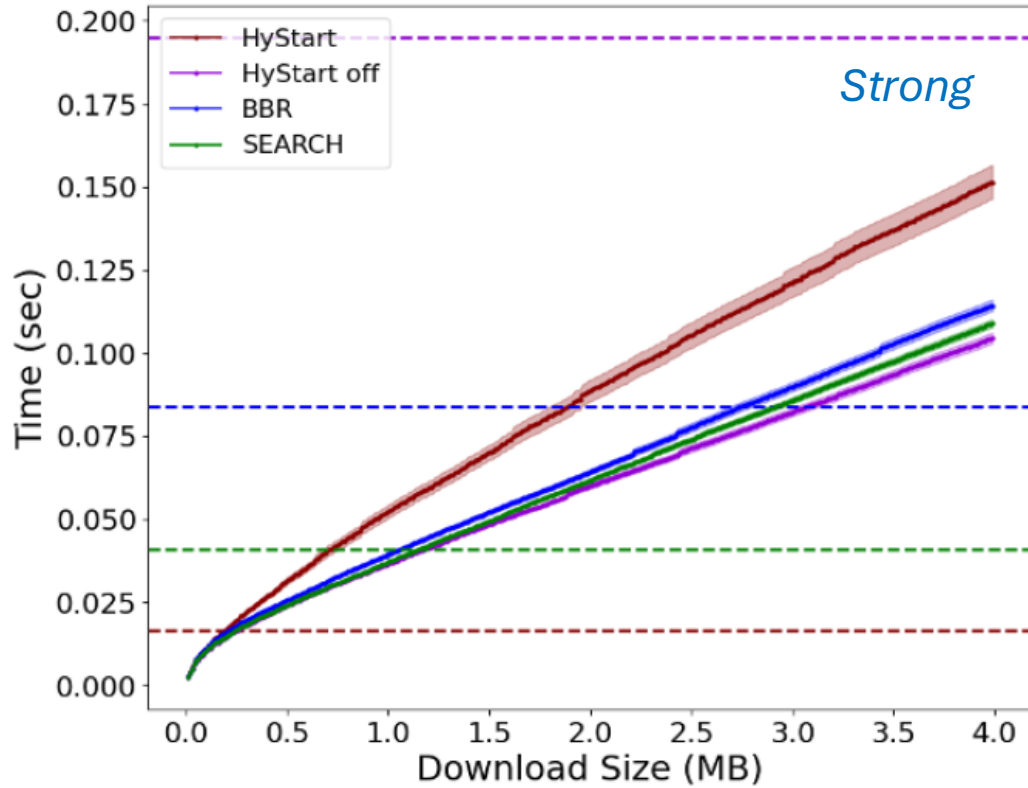


- General strong correlation between higher RSSI and higher throughput.
- Other factors may cause inconsistency such as shared nature of campus APs.
- Two red dots are manually examined locations.



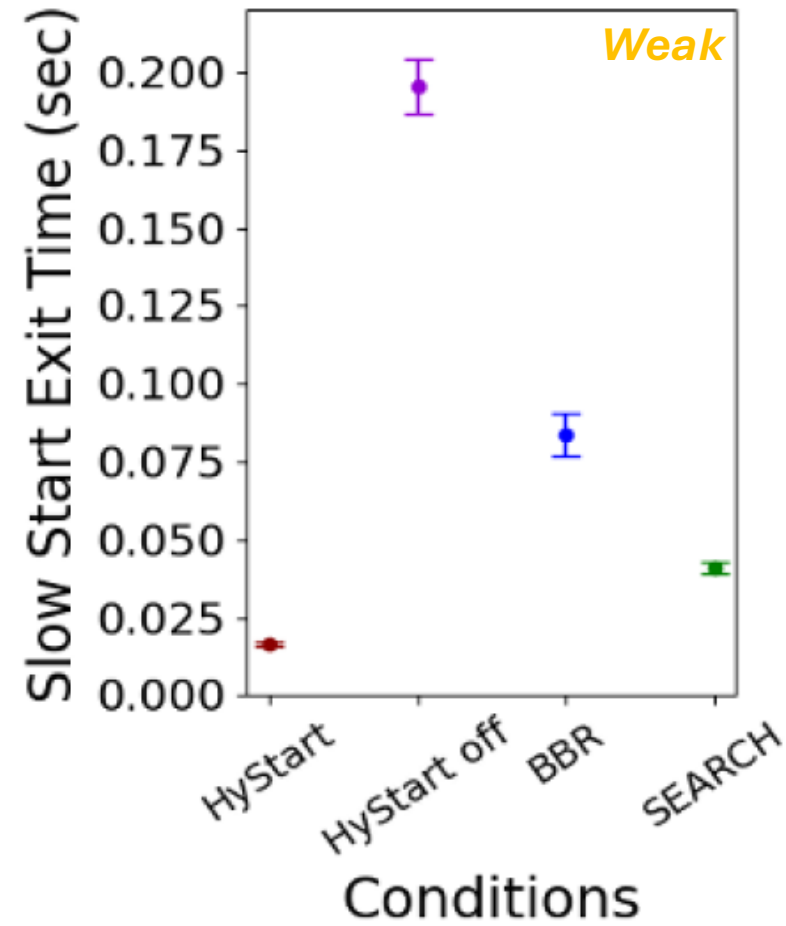
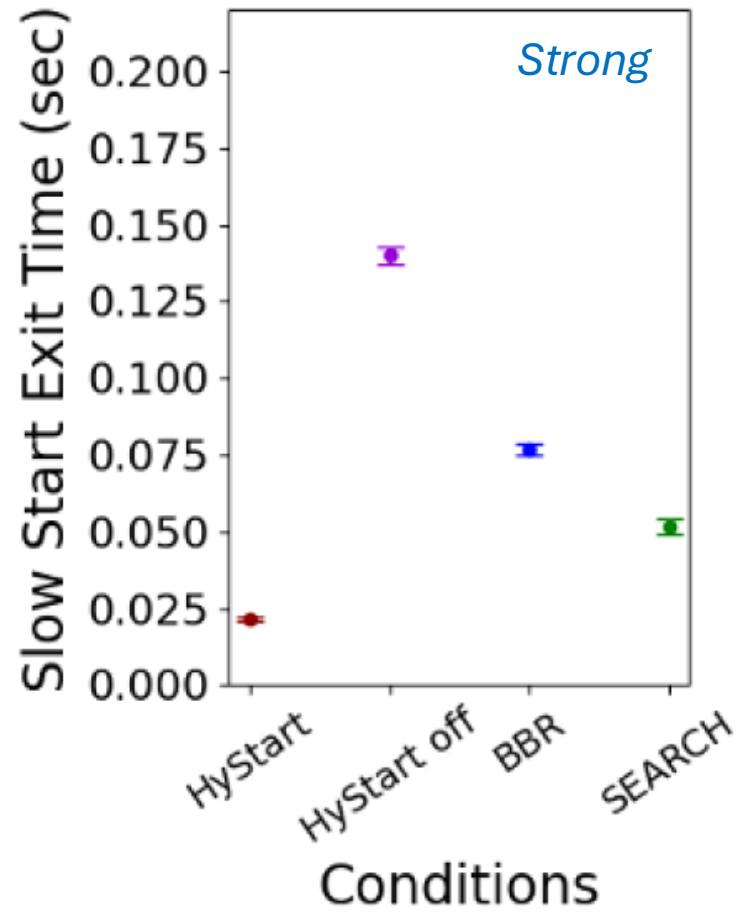
- Median throughput CDFs at *strong* and *weak* locations
- Median throughput at *strong* location is higher than *weak* locations.

Time to Download

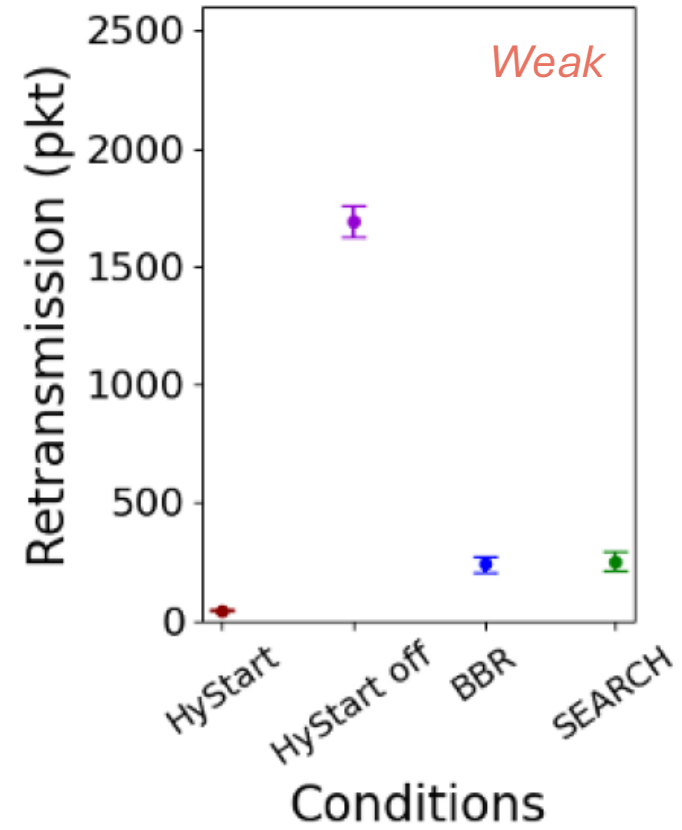
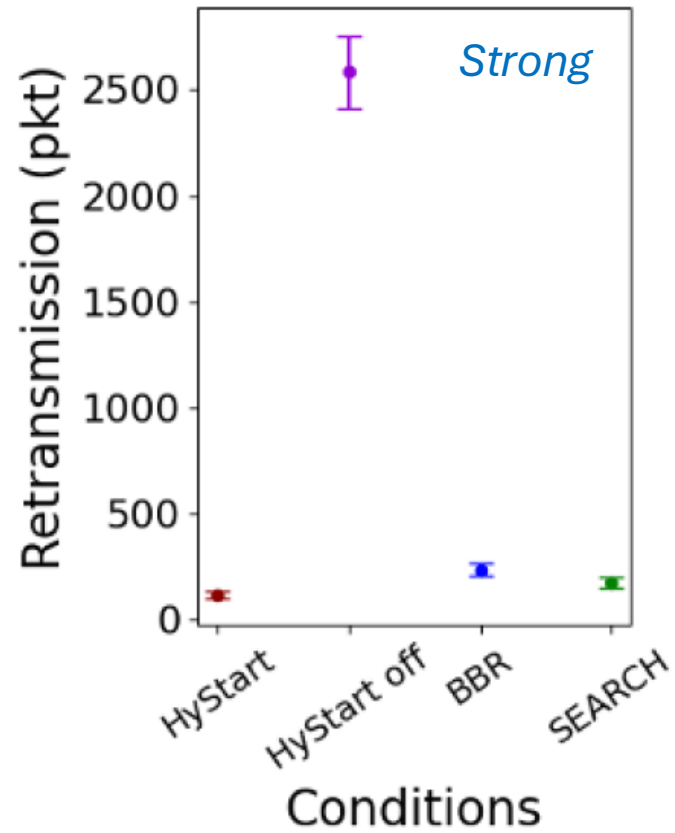


* Horizontal lines indicates the slow start exit time for each CCAs matching their respective color

Slow Start Exit Time



Retransmission



Working in Progress

- Project Page <https://search-ss.wpi.edu>
- Linux Kernel Module Implementations
 - 5.13.x series kernel
 - https://github.com/Project-Faster/tcp_ss_search.git (main branch)
 - 6.10rc2 based (net-next-6.10rc2 branch)
 - https://github.com/Project-Faster/tcp_ss_search/tree/net-next-6.10rc2
- QUIC implementation H2O/Quicly
 - <https://github.com/Project-Faster/quicly/tree/generic-slowstart> (generic-slowstart branch)
- IETF 120 CCWG Draft0
 - <https://datatracker.ietf.org/doc/draft-chung-ccwg-search/>
 - July 24, 2024, IETF 120 CCWG group meeting.
 - Looking for comments on algorithm/design/implementation.

Future works

- Future iteration of the SEARCH algorithm will focus on
 - Refining slow start exit strategy to precisely match congestion condition.
 - **Upstream TCP SEARCH into Linux mainstream and integrate it into open source QUIC.**
 - https://github.com/Project-Faster/tcp_ss_search.git
- More evaluations with search
 - Over other link type (LTE, LEO, 5G/mmWave, Data Center, etc)
 - Varied network condition.
- Incorporate with BBR's probing phase

Conclusions

- HyStart does not work in wireless environments (GEO, LTE, WIFI) and causes premature slow start exits.
 - Need a fix in near future
- SEARCH achieves its design goals
 - Effectively exits Slow Start after reach maximum capacity,
 - Minimizing packet loss and improving network efficiency.
 - Reach comparable maximum throughput as TCP w/o Hystart with fewer retransmissions.

References

- [KCC24] Kachooei, M. A.; Chung, J.; Cronin, A.; Chung, J.; Li, F.; Peters, B.; and Claypool, M. 2024. Improving TCP Slow Start Performance in Wireless Networks with SEARCH. In Proceedings of the IEEE World of Wireless, Mobile and Multimedia Networks (WoWMoM).
- [KCL23] Kachooei, M. A.; Chung, J.; Li, F.; Peters, B.; and Claypool, M. 2023. SEARCH: Robust TCP Slow Start Performance over Satellite Networks. In Proceedings of the IEEE 48th Conference on Local Computer Networks (LCN), 1–4.
- [RFC9406] Huang, Y., and Olson, M. a. 2023. HyStart++: Modified Slow Start for TCP. RFC 9406, RFC Editor.

Thank you for your
attention!

Static Memory Footprint Analysis

Structure	Size
struct sock	736 bytes
struct tcp_sock	2192 bytes

Module	Size (bytes)	Overhead (%)	Additional Details
CUBIC	72	2.5	60 bytes + ack_sample struct: 12 bytes
HyStart	18		Part of CUBIC's 72 bytes
SEARCH2.0	121	4.1	Exclude shared cubic fields
BBR (v1.0)	156	5.3	100 bytes + rate_sample struct: 56 bytes

* Based on 5.10.79 kernel