UPDATE

SEARCH – a New Slow Start Algorithm for TCP and QUIC

Jae Chung
Feng Li
Maryam Ataei Kachooei
Mark Claypool

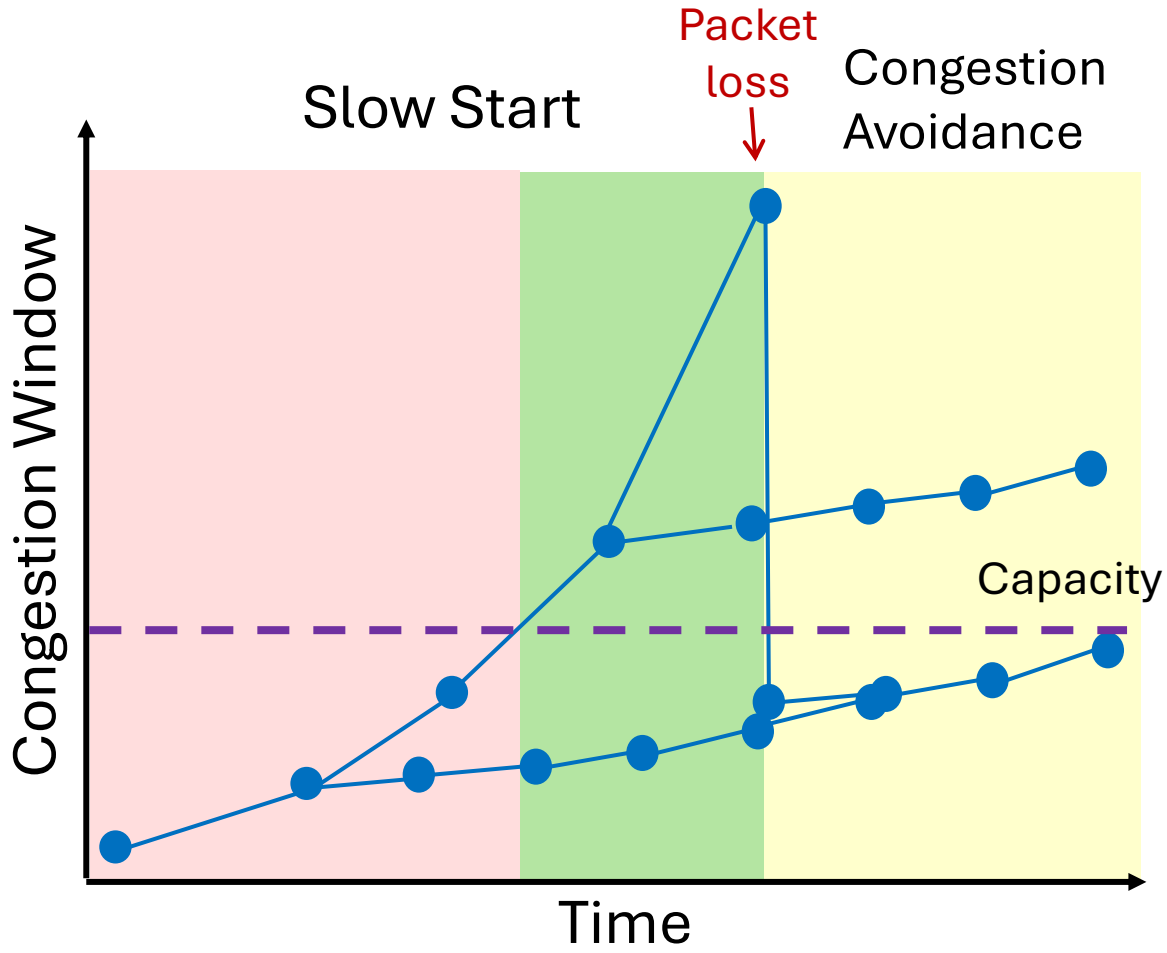
IETF CCWG
Bangkok, Thailand
March 2025

Viasat

WPI

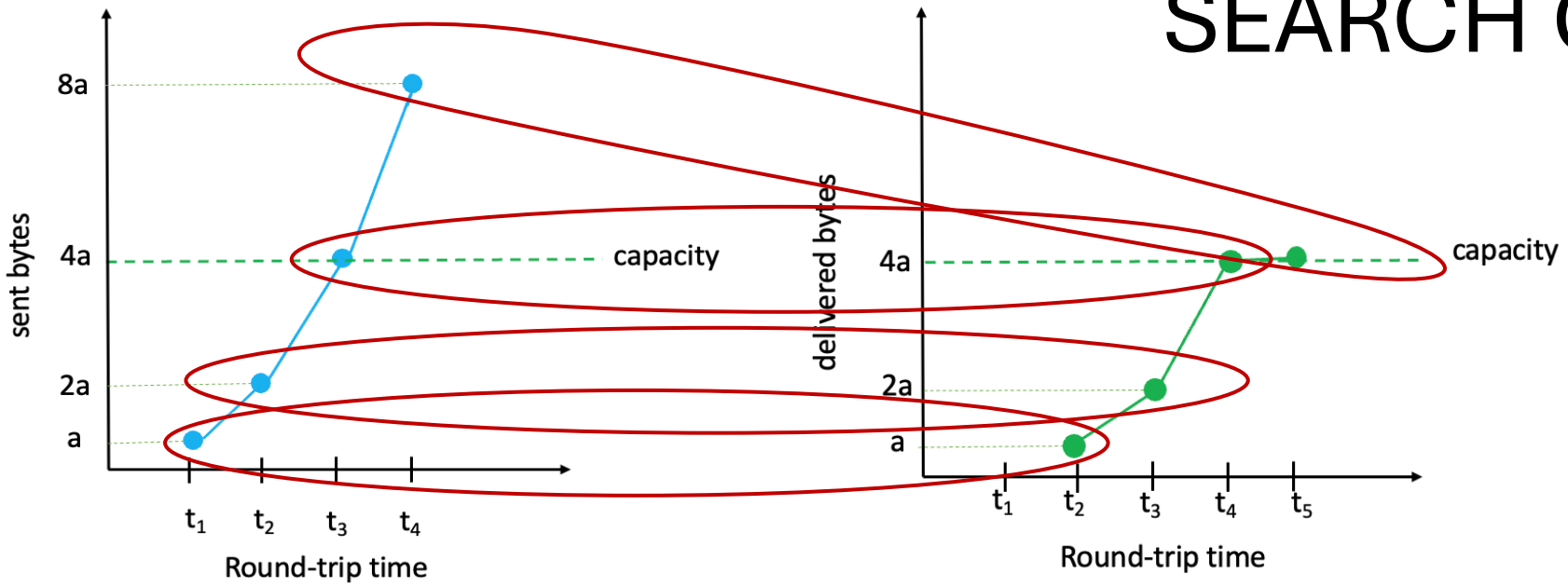
TCP Slow Start

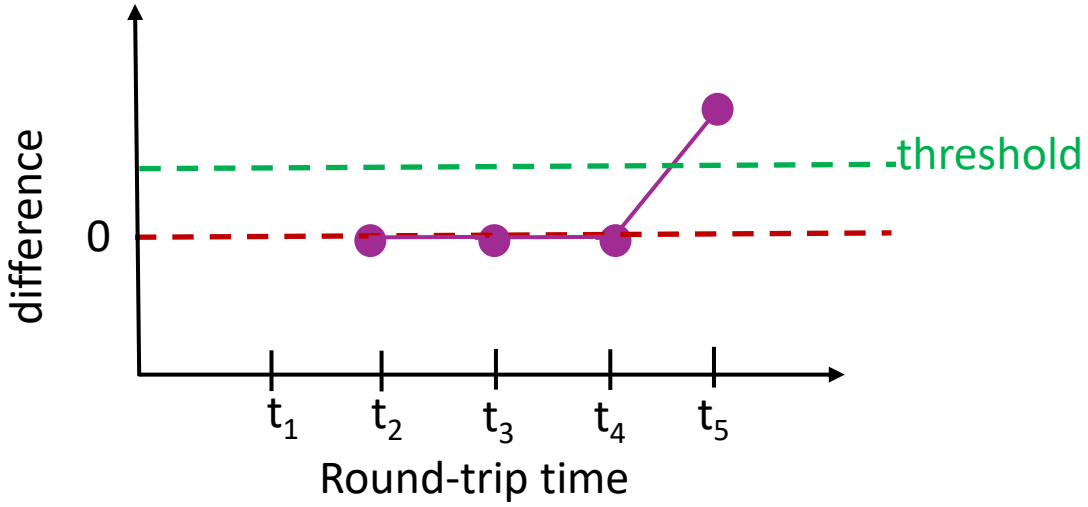
- Congestion window doubles each RTT
- Packet loss ➡ Exit slow start: **Too late (After loss)**
- Idea: Exit after reaching capacity and before loss
- HyStart (Linux default)
- HyStart over wireless ➡ **Too early (Before capacity)**



Goal: ➡ **At chokepoint: after capacity, before loss**

SEARCH Concept





SEARCH Algorithm

curr_delv

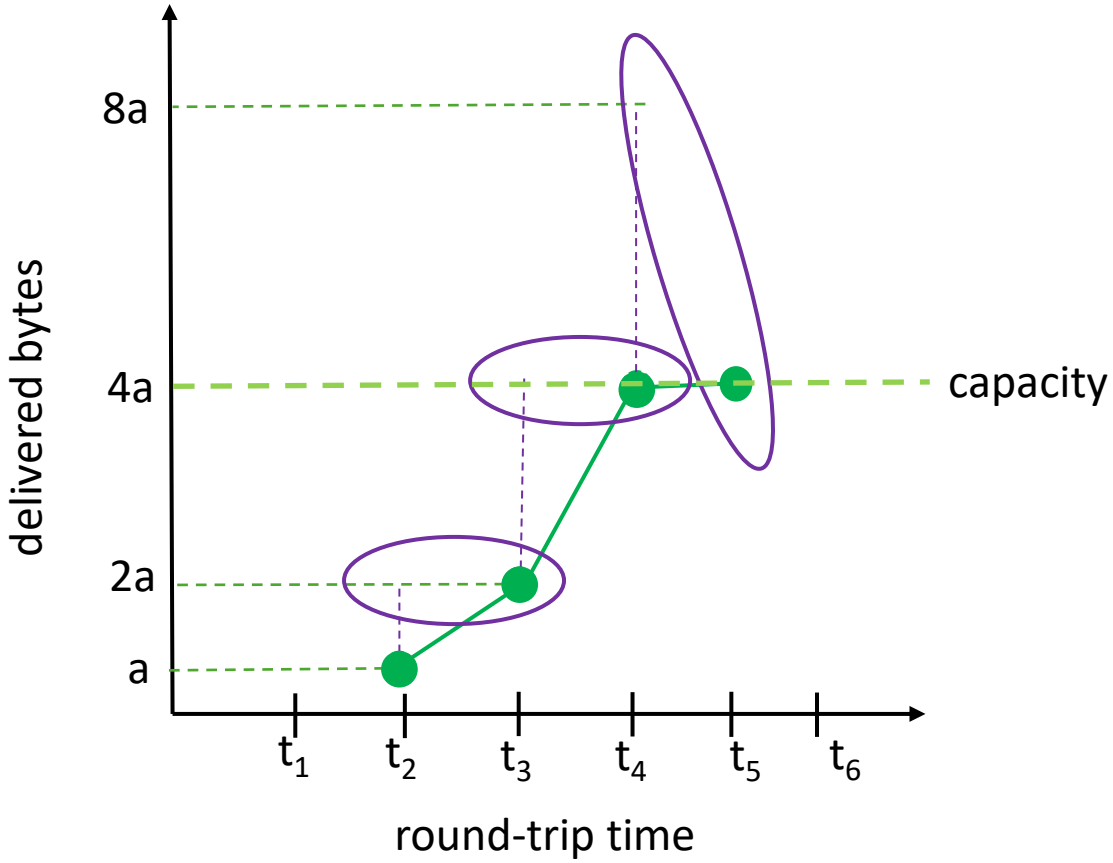
prev_delv

diff = 2 prev_delv – curr delv

normalized diff = diff / 2 prev_delv

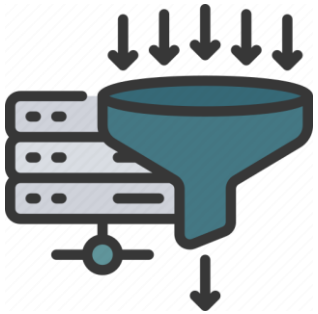
if (normalized diff ≥ threshold):

exit from slow start

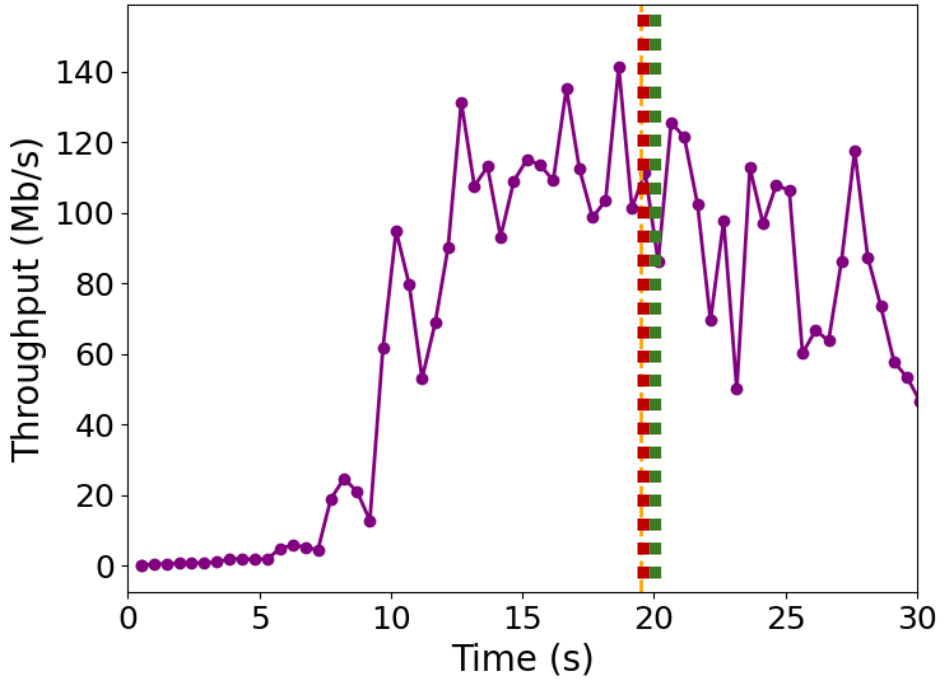


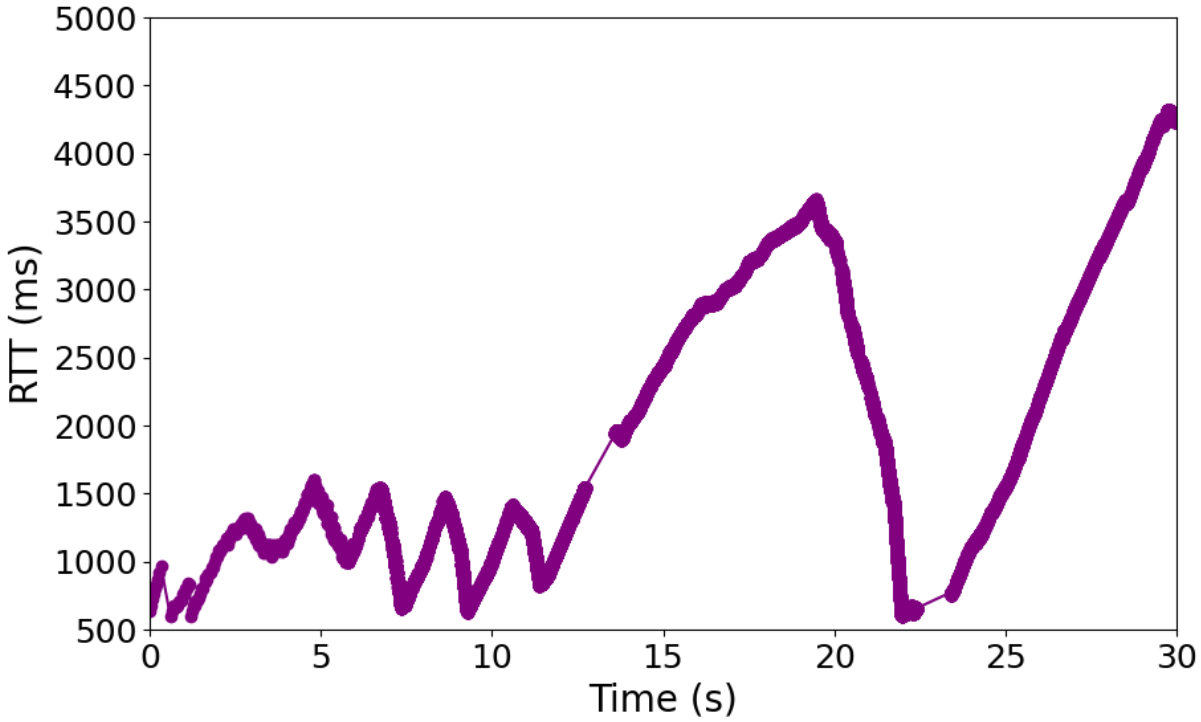
Outline

- Place SEARCH in slow start space
  - Delay-based: HyStart, HyStart++
  - Delivered Bytes-based: BBR, SEARCH
- Memory use delivered byte history
- Test framework
- Summary



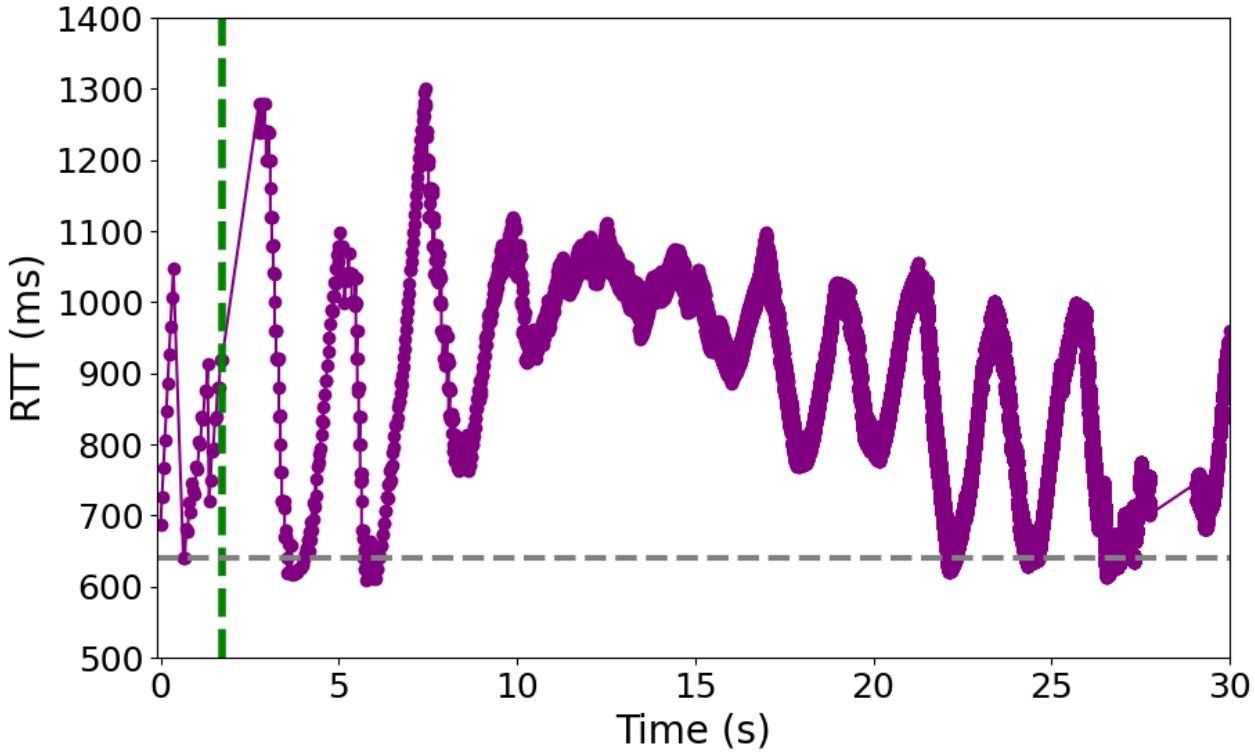
Cubic (without HyStart)

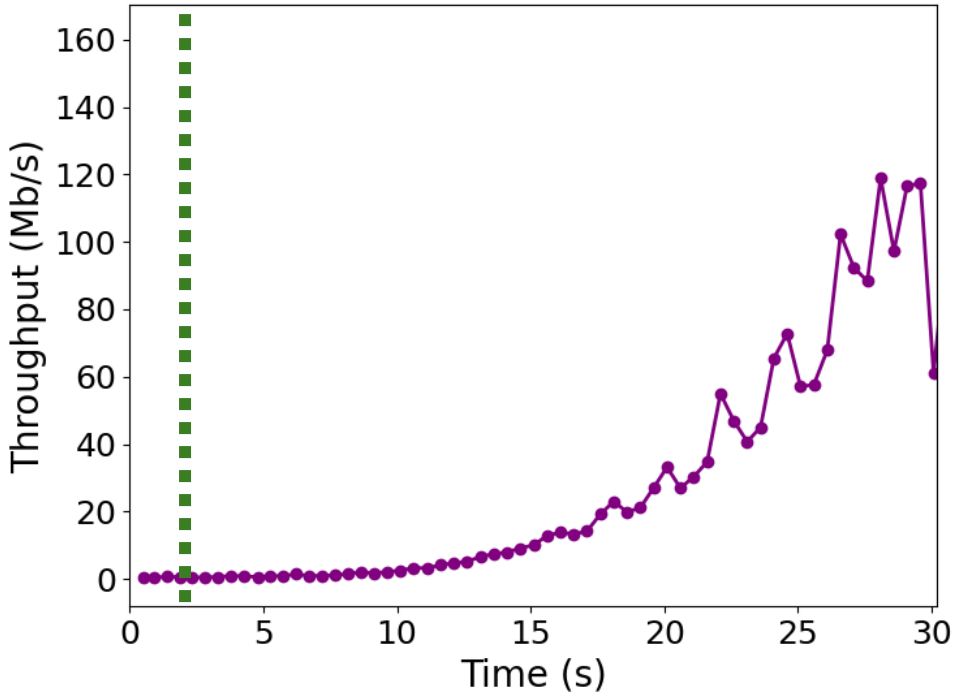




Delay-Based: HyStart

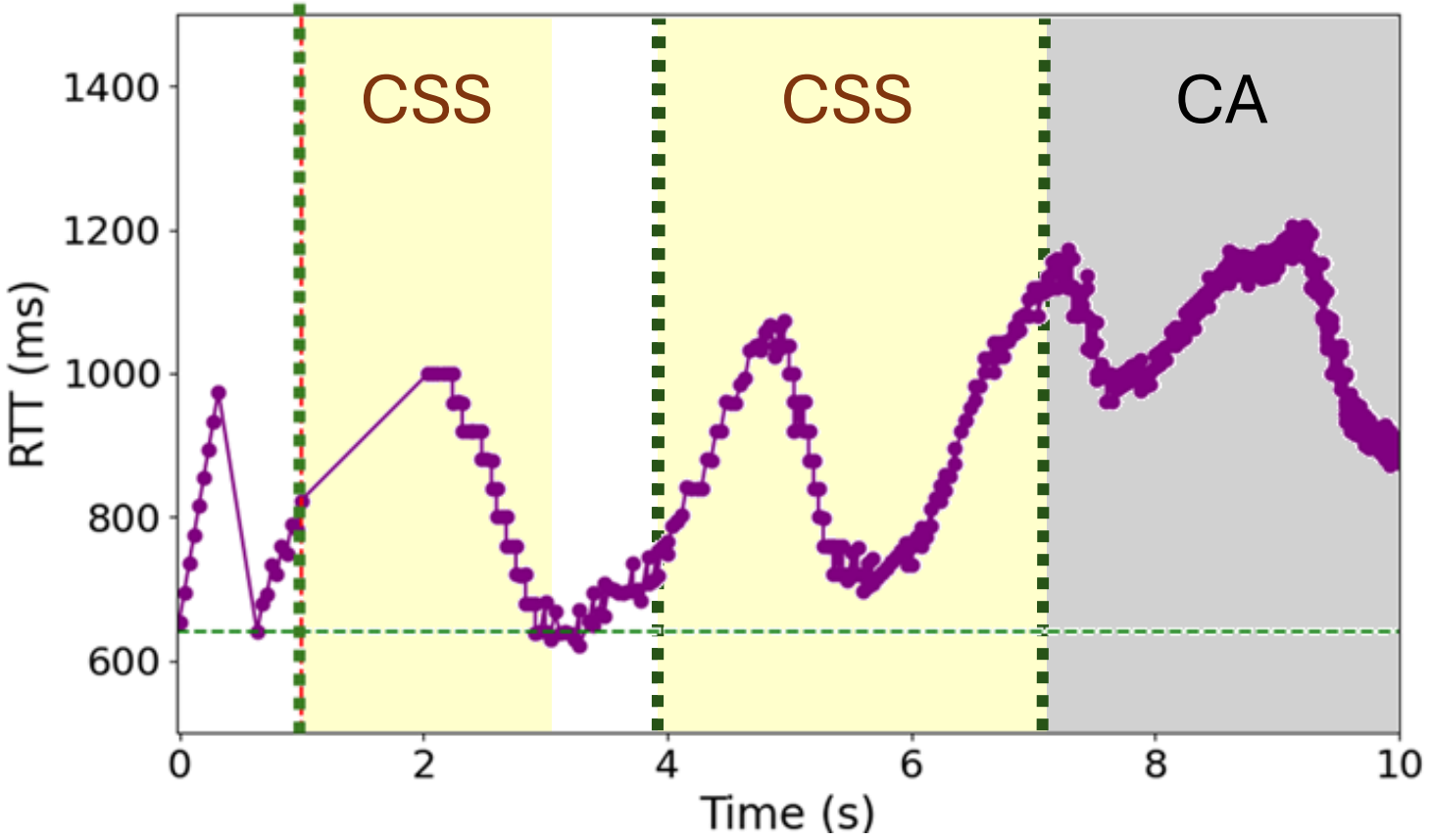
- RTT “spikes” → exit SS
- Sensitive to RTT variation → premature exit, underutilization
  - Bad in high BDP networks





Delay-Based: HyStart++

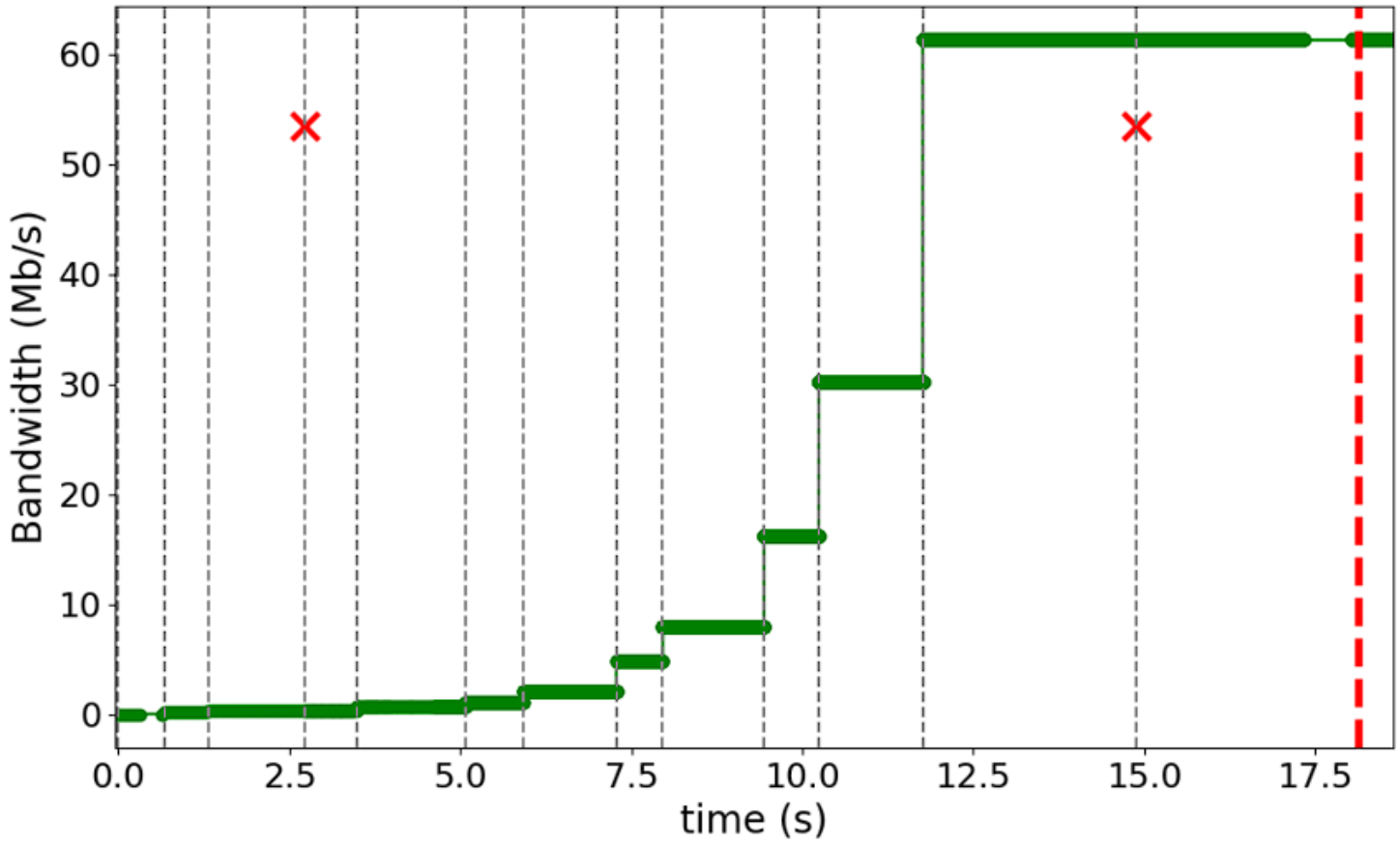
- Recognizes that delay increase a "noisy" signal
  - Doesn't have better signal!
- Conservative Slow Start (CSS)
  - Slower growth
  - Check delay
- Delay goes down?
  - Yes → back to SS
  - No → go to CA



- While in CSS grow at slower rate than 2x
- If HyStart was "wrong" – grow slower for some time
- If HyStart was "right" – CSS pushes past capacity limit

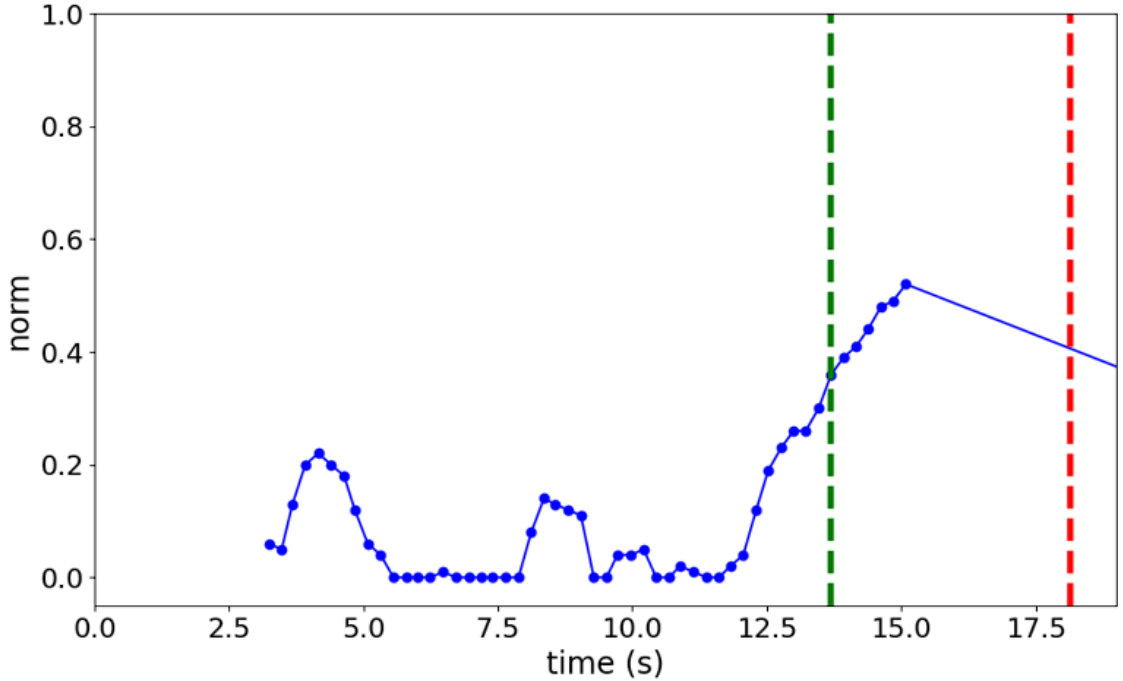
Delivered Bytes-Based: BBR

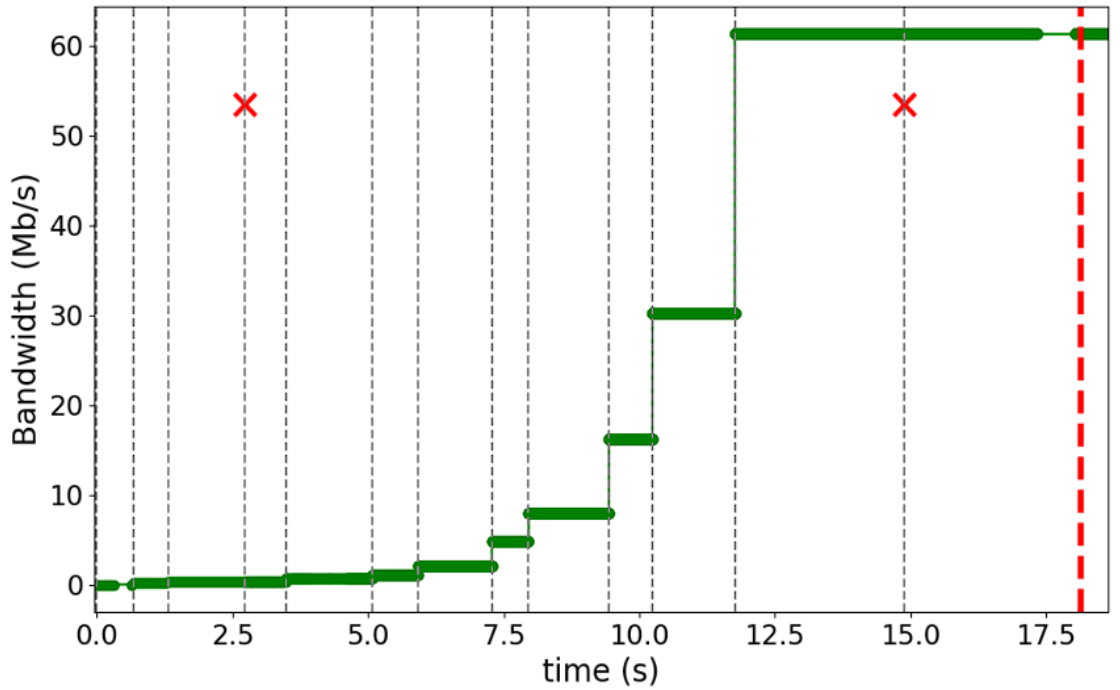
- Exits when delivery rate stops growing
  - 3 consecutive rounds (RTTs), less than 25%
  - Decision each round
- When at capacity, RTTs increase → rounds increase



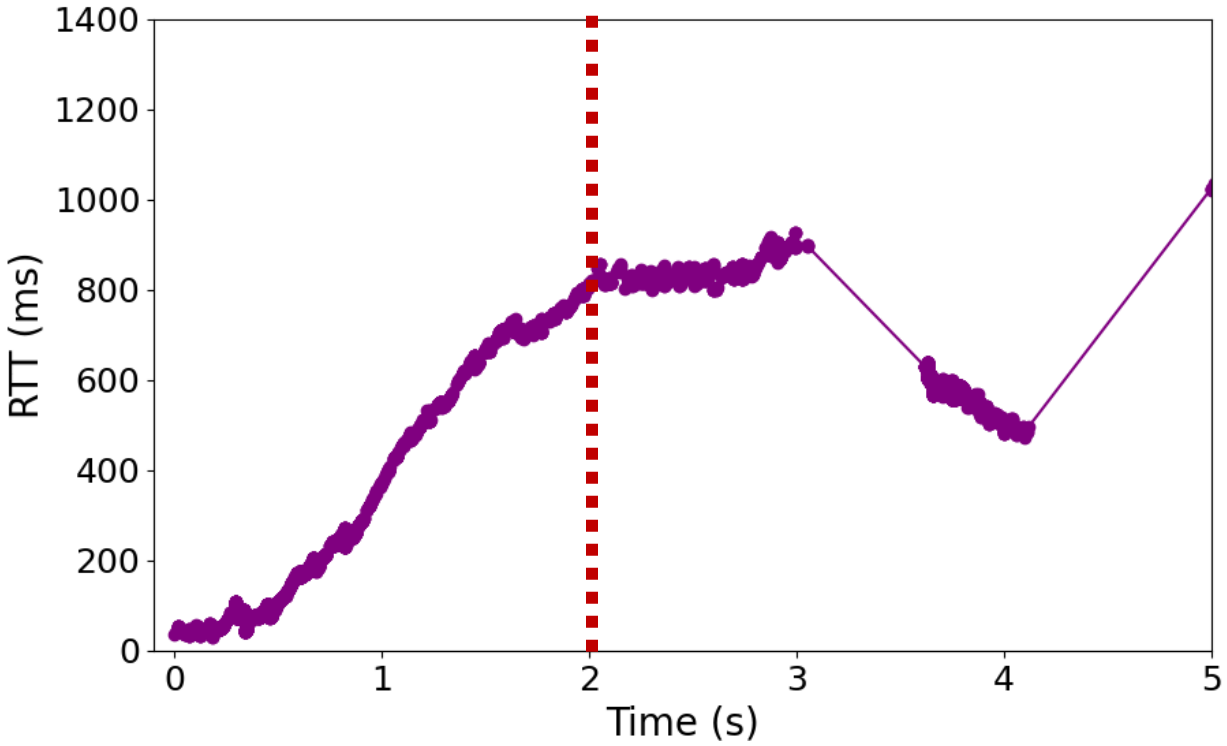
Delivered Bytes-Based: SEARCH

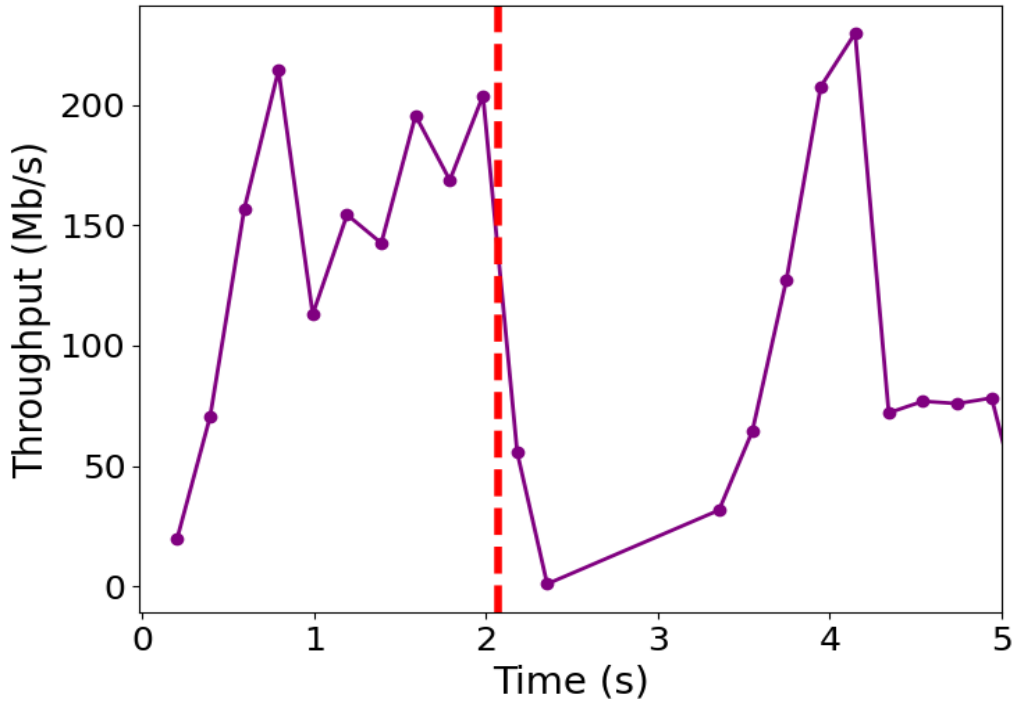
- Exits when delivered bytes increase less than 30% from previous RTT
- Decision each bin boundary (about ½ RTT)
- Uses initial RTT, not current → resilient to congestion delay



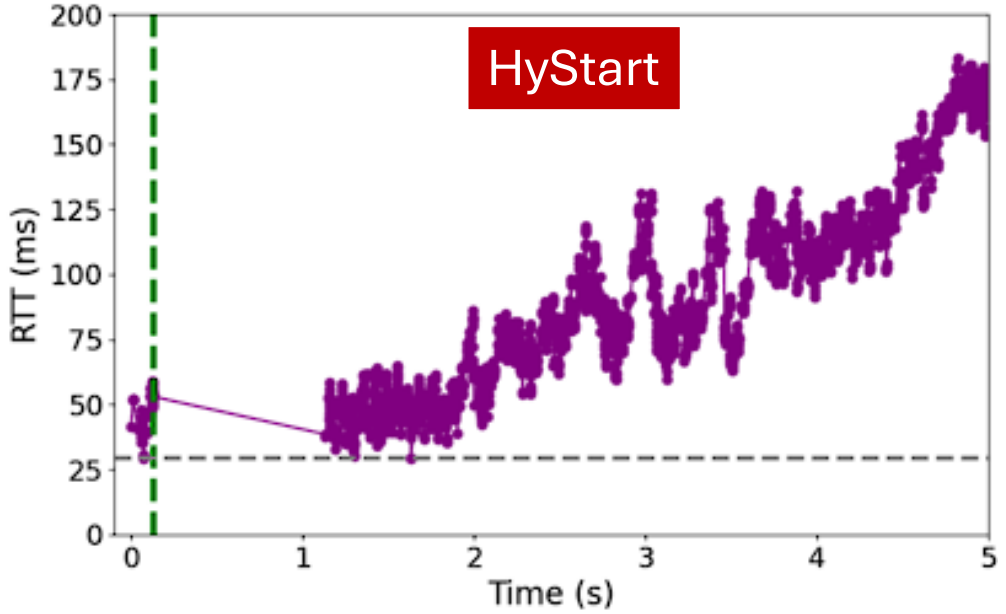


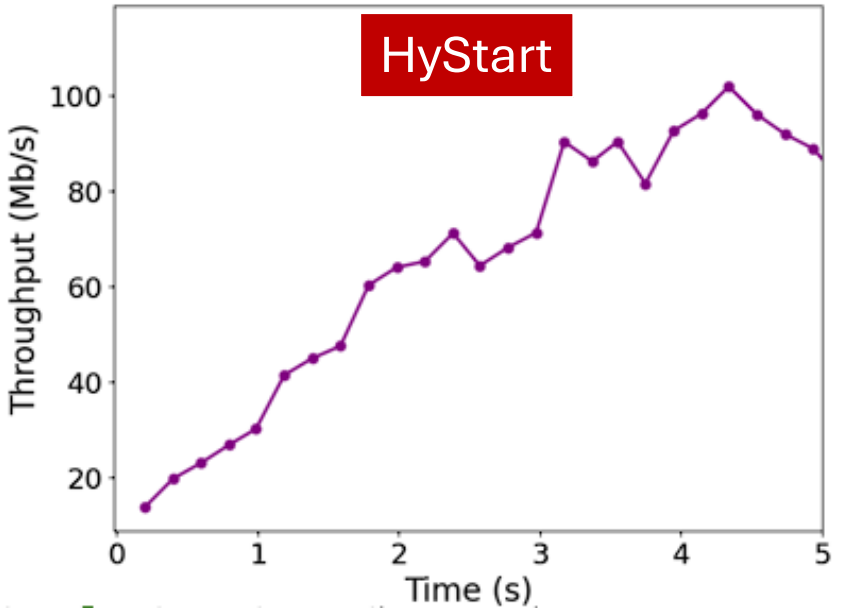
4G LTE – Same old story

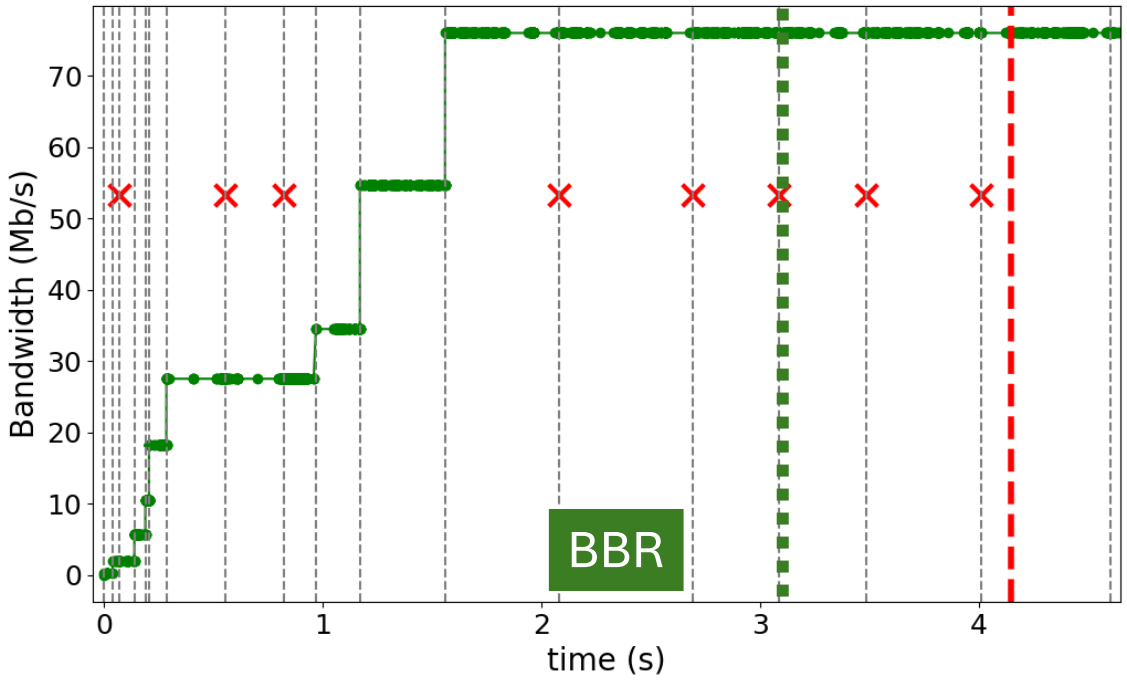


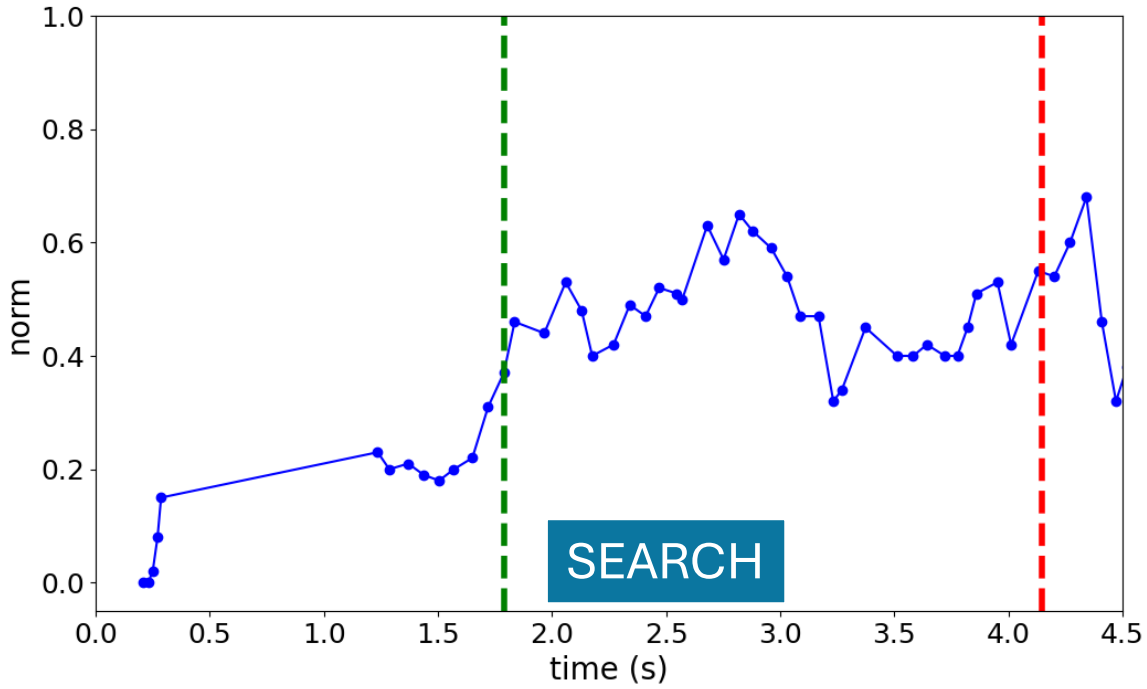


4G LTE


HyStart
RTT (ms)
Time (s)


HyStart
Throughput (Mb/s)
Time (s)


BBR
Bandwidth (Mb/s)
time (s)

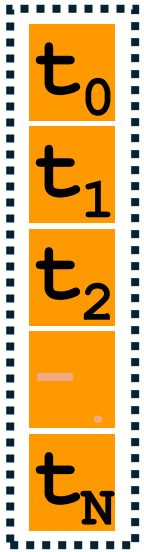

SEARCH
norm
time (s)

Outline

- Place SEARCH in slow start space (done)
  - Delay-based: HyStart, HyStart++
  - Delivered Bytes-based: BBR, SEARCH
- Memory use delivered byte history (next)
- Test framework
- Summary

Memory Use for Delivered Bytes

- BBR
  - Time when every segment sent
  - When segment delivered, compute delivery rate
    - bytes delivered from sent to now / time
  - Storage overhead – time size x cwnd size
- SEARCH
  - Sequence number each bin (~½ RTT)
  - When bin ends, compute delivered
    - seq now – seq then
  - Storage overhead – sequence x window



100 MB/s, 100 ms RTT

= 83 total

Window 10 + History 15

= 25 total

Outline

- Place SEARCH in slow start space (done)
  - Delay-based: HyStart, HyStart++
  - Delivered Bytes-based: BBR, SEARCH
- Memory use delivered byte history (done)
- Test framework (next)
- Summary

Kernel code
(tcp_cubic_search.c)

```
// SEARCH_defs_begin

// SEARCH_defs_end
```

```
// SEARCH_begin

// SEARCH_end
```

Challenges:

1. Verify implementations of SEARCH
2. Compare performance for parameter tuning

Source Code

Code Extraction

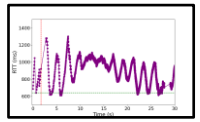
Setup Support Files

keyword_defs.h | keyword_mod.c | tcp.h | cc_helper_func.h | test_keyword.c

Input
CSV file
t_1 s_1
t_2 s_2
...
t_n s_n

Input csv

Compilation & Execution



Output file

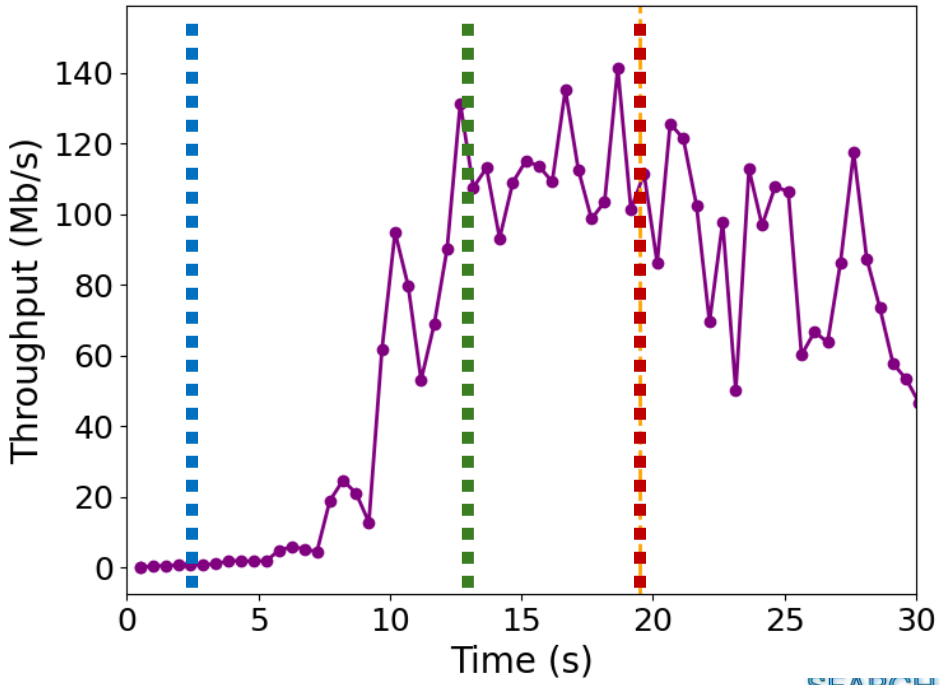
Output
Log file
Pass
Pass
Fail
Pass

Server Client

Using the Framework

- Verify and validate SEARCH implementations
  - Linux, versions
  - FreeBSD
  - QUIC
- Compare performance
  - Traditional, HyStart, SEARCH
  - BBR, HyStart++
- Traces!
  - Normal, edge cases
  - WiFi, 4g/5g, GEO, LEO, Wired

Output
Log file
Pass
Pass
Fail
Pass



Summary

SEARCH

https://search-ss.wpi.edu/

- SEARCH
  - Determines “choke point” from expected delivered bytes
  - Exits slow start after congestion point, before loss
- Framework to verify implementations
- Looking for volunteers to try it out!

claypool@wpi.edu

Thank-you for your attention!

SEARCH – a New Slow Start Algorithm for TCP and QUIC

Jae Chung
Feng Li
Maryam Ataei Kachooei
Mark Claypool

IETF CCWG
Bangkok, Thailand
March 2025

Viasat

WPI

References

- Improving TCP Slow Start Performance in Wireless Networks with SEARCH
  - *IEEE World of Wireless, Mobile and Multimedia Networks (WoWMoM)*
  - Perth, Australia, June 2024
- Improving QUIC Slow Start Behavior in Wireless Networks with SEARCH
  - *IEEE Local and Metropolitan Area Networks (LANMAN)*
  - Boston, Massachusetts, USA, July 2024
- Implementation of the SEARCH Slow Start Algorithm in the Linux Kernel
  - *0x18 NetDev Conference*
  - Santa Clara, California, USA, July 2024
- Reducing Per-flow Memory Use in TCP SEARCH
  - *IEEE World of Wireless, Mobile and Multimedia Networks (WoWMoM)*
  - Fort Worth, TX, USA, May 2025

TCP Congestion Control Test Framework

1. **ss_extract.py** - Extracts Relevant Code
   - **Usage:** `ss_extract.py -f original_file.c -k keyword`
   - **Functionality**:
     - Extracts slow start-related code using labeled sections (`keyword_begin`, `keyword_end`, `keyword_defs.begin`, `keyword_defs.end`).
     - Generates three key files in `test_dir`:

       1. `keyword_modules.c`
       2. `keyword_defs.h`
       - `tcp.h` (based on extracted modules)

TCP Congestion Control Test Framework

2. **ss_setup.py** - Sets Up Testing Environment

- **Usage:** `ss_setup.py -k keyword`
- **Functionality:**
  - Determines the appropriate test file for the specified protocol.
  - If a predefined `test_keyword.c` exists for the keyword (e.g., SEARCH, HyStart, BBR), it is placed in `test_dir`.
  - If no predefined test file is available, `test_base.c` is provided for user modification.
  - Places `cc_helper_functions.h`, which includes essential congestion control functions, in `test_dir`. Users can extend this file by extracting additional functions from the kernel if needed.

TCP Congestion Control Test Framework

3. **ss_run.py** - Executes the Tests
   - **Usage:** `ss_run.py -k keyword -i input -o output`
   - **Functionality:**
     - Runs the test with specified input CSV files.
     - Runs test_keyword:
       - Reads the input CSV file with test parameters.
       - Applies the congestion control logic.
     - Outputs test results to output.txt.

Per-Flow Memory Use