

Rapid Start

draft-kazuho-ccwg-rapid-start-02

IETF 125

Kazuho Oku @ Fastly

Agenda

- Issues with Slow Start
- Rapid Start's Approach
- Production Experiment Results
- Considerations

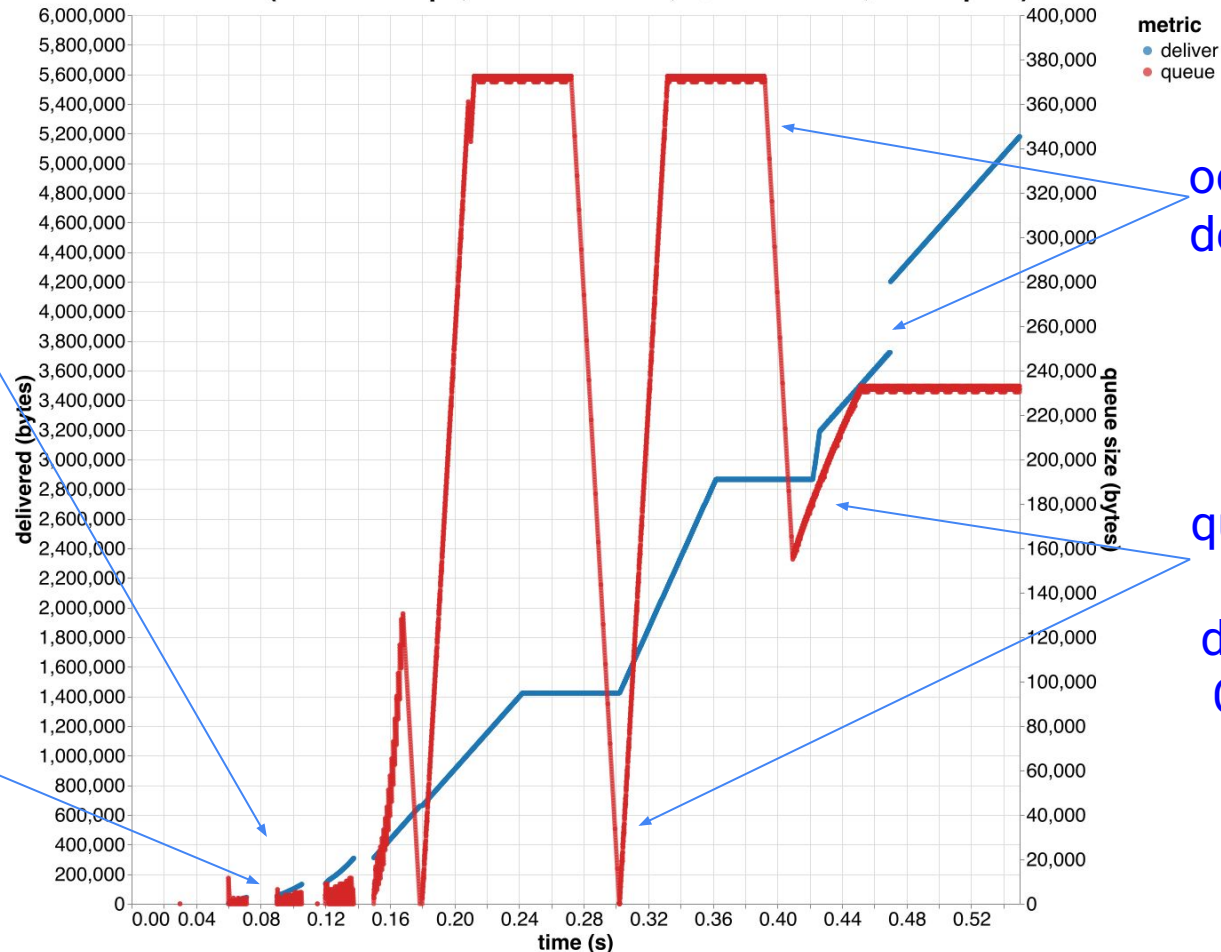
Issues with Slow Start

Issues with Slow Start

- Majority of the HTTP connections never leave slow start
 - Improving the startup phase is crucial for better web performance (esp. TTLB)
- However, with paced slow start:
 - In the 1st RT, packets are sent for only $\frac{1}{2}$ RTT, remaining $\frac{1}{2}$ is idle
 - Ramp-up is only 2x per RT
 - Upon 1st recovery, CWND reduction is rough and remains too large, triggers second recovery event immediately after
 - Even though recoveries need to be avoided as they block data delivery until lost data is retransmitted and recovered

Issues with Slow Start (on simulator)

Slow Start (BW=100Mbps, RTTidle=30ms, Queue=30ms, IW=30pkts)



2x growth is slow

moments of network idle due to pacing over $\frac{1}{2}$ RTT

2nd recovery occurs too soon, delaying delivery once again

queue is drained then filled up, due to reducing CWND by beta

Rapid Start's Approach

Rapid Start in a nutshell

- Full-RTT pacing:
 - In the 1st RTT, pace for the entire RTT (rather than $\frac{1}{2}$ RTT)
- Rapid increase of CWND:
 - Until queue build-up is observed, increase CWND by 3x per RT
 - Once queue build-up is observed, go back to 2x per RT, and then optionally switch to HyStart++, etc.
- Quicker and smoother convergence to CWND comparable to that during Congestion Avoidance:
 - Upon congestion, gradually reduce CWND so that, upon exiting the recovery period, CWND becomes $\beta * fullBDP$

Full-RTT pacing

- To ensure the path's queueing behavior is observed rather than sender-side burstiness, the sender **SHOULD pace the packets over a full RTT**, using the current RTT estimate, **when it first sends more data than classic slow start with pacing would permit.**
- **This send window can be $IW * 2$** , as that yields comparable sending rate to paced slow start.

Can be read as: Use the Unvalidated Phase of Careful Resume with
 *$jump_wnd = 2 * IW$*

Rapid Increase of CWND

- $CWND = (\text{queue_buildup}() ? 1 : 2) * \text{bytes_acked}$
 - i.e., 3x per RTT until queue buildup is observed, then use 2x
- `queue_buildup()` being RECOMMENDED:
 - $\text{rtt_floor} > \min(\text{rtt_min} + 4\text{ms}, \text{rtt_min} * 1.1)$
 - `rtt_floor` is smallest RTT observed over the most recent 1RT

Reuses HyStart++'s RTT sampling machinery, but applies a tighter threshold, because Rapid Start uses the signal to allow growth above 2x, whereas HyStart++ uses it to stop 2x growth (paraphrase: Rapid Start is compatible with HyStart++).

Handling Congestion: Problems

- Upon entering recovery after exiting slow start, CWND remains too aggressive unless HyStart++ kicked in early enough.
 - Why: because loss is observed one full RT later, CWND is $2 * \text{fullBDP}$ by the time recovery starts. Therefore, CWND is reduced only to fullBDP, which triggers the next recovery almost immediately.
 - note: $\text{fullBDP} = \text{idle BDP} + \text{bottleneck queue size}$
 - For slow start, reducing to $\beta / 2$ yields a CWND comparable to that during Congestion Avoidance.
- However, just doing $\text{CWND} *= \beta / 2$ is also suboptimal, because the sender remains idle for too long. If $\beta = 0.5$, the sender would remain idle for $\frac{3}{4}$ of a full RT. That might completely drain the queue, leading to underutilization of the bottleneck path, followed by a burst 2x faster than the bottleneck flow rate.

Handling Congestion: Problems (cont'd)

- Rapid Start might observe the first loss with CWND as large as $3 * fullBDP$, and it becomes even more important to avoid the two issues:
 - overshoot of CWND
 - Why: if the overshoot is $2x$, fixing losses takes 1 RT; if the overshoot is $3x$, it takes 2 RT.
 - excessive draining followed by burst
 - Why: because sharper reduction (up to $\frac{5}{6}$ when overshooting by $3x$ and $beta = 0.5$) leads to increased chance of entirely draining the bottleneck queue and underutilization of the bottleneck path, followed by a burst that could be $3x$ as fast than the bottleneck flow rate.

Handling Congestion: The Desired Properties

- Upon entering recovery, the sender should stop and drain the queue, but no more than in Congestion Avoidance; i.e., the sender should remain silent for no longer than $(1 - \beta) * \text{fullRT}$.
- Once the sender resumes sending, it should avoid bursty behavior.
- Upon exiting recovery, CWND should be $\beta * \text{fullBDP}$, as in Congestion Avoidance.

Handling Congestion: Rapid Start's Approach

- Upon entering recovery:

$cwnd *= silence_factor$

- For each ACK received:

$cwnd -= ack_factor * bytes_newly_acked + loss_factor * bytes_newly_lost$

- Once sending resumes, the sending ratio becomes fixed, assuming loss ratio during recovery is uniform and ACKs arrive frequently and regularly ^{note1}.
- Factors are chosen constants so that:
 - In the case of worst overshoot (3x), the sender remains silent for no more than during Congestion Avoidance.
 - Upon exiting recovery, land with $CWND = beta * bytes_acked_in_recovery$ ^{note2}

Note1: For each ACK, inflight is reduced by $byte_newly_acked + bytes_newly_lost$. As a result, for each ACK, the sender sends $(1 - ack_factor) * bytes_newly_acked + (1 - loss_factor) * bytes_newly_lost$ bytes.

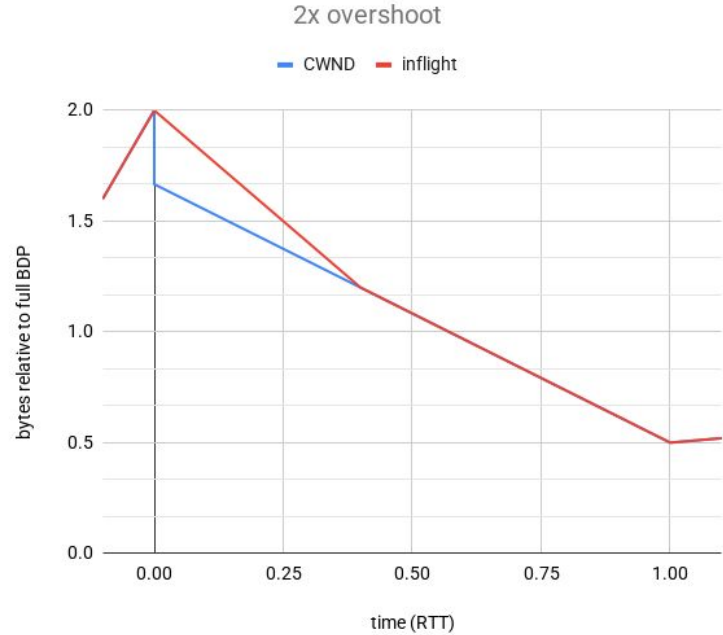
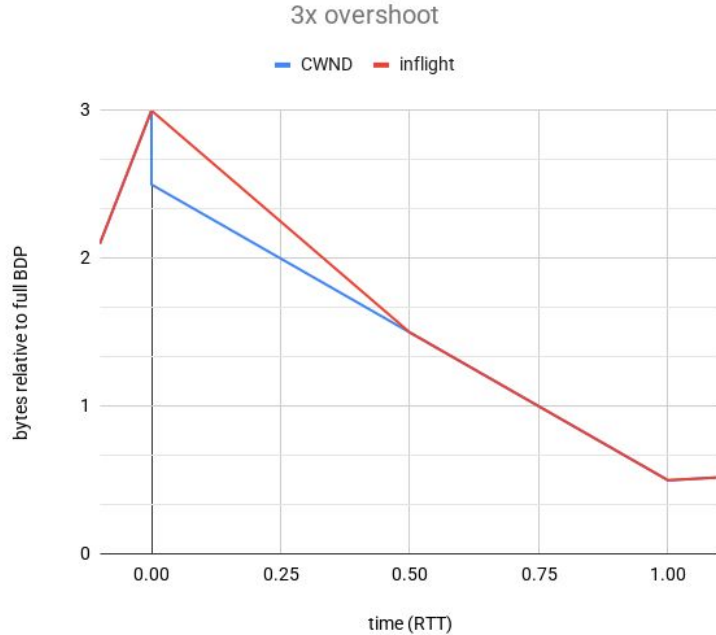
Note2: Assuming that all losses are due to queue overflows, fullBDP equals $bytes_acked$ during recovery.

Handling Congestion: The Factors

- If $\beta = 0.5$:
 - $\text{silence_factor} = 5/6$
 - $\text{Ack_factor} = 1/3$
 - $\text{loss_factor} = 5/6$
- If $\beta = 0.7$:
 - $\text{silence_factor} = 9/10$
 - $\text{ack_factor} = 1/5$
 - $\text{loss_factor} = 9/10$
- The derivation process is explained in Appendix. A.

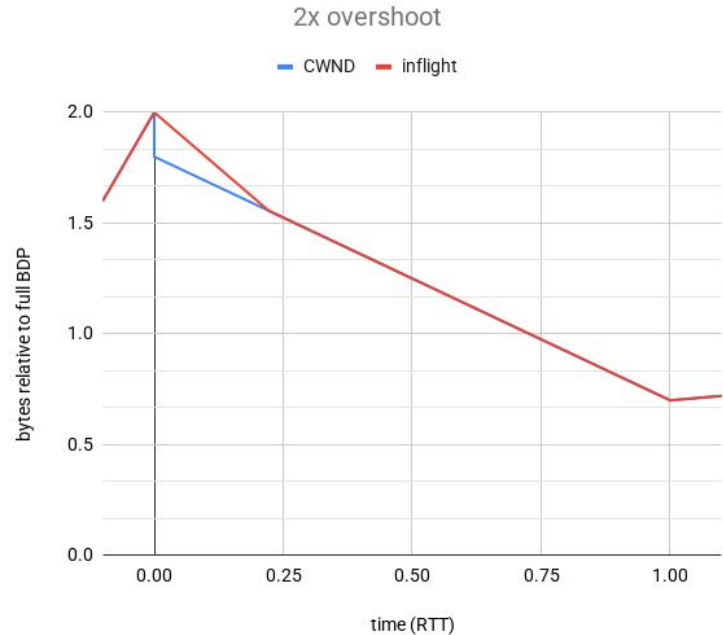
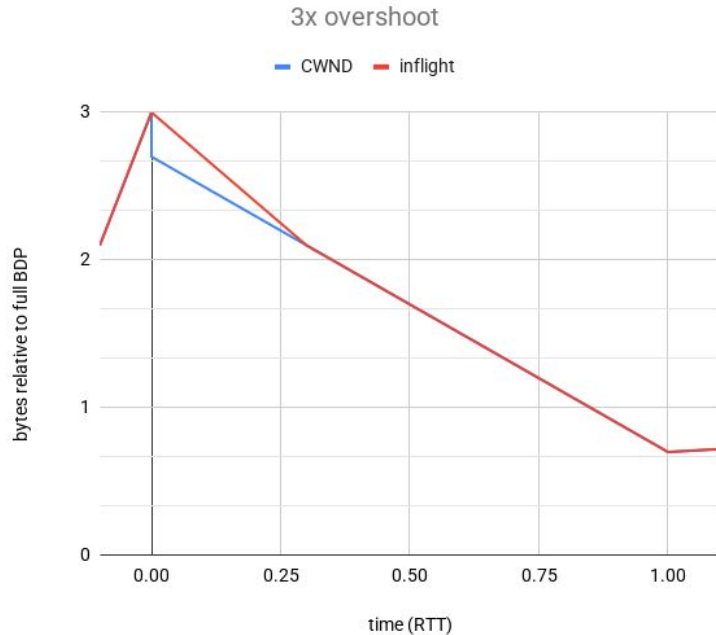
Handling Congestion: How It Looks

- $\beta = 0.5$:



Handling Congestion: How It Looks

- $\beta = 0.7$:

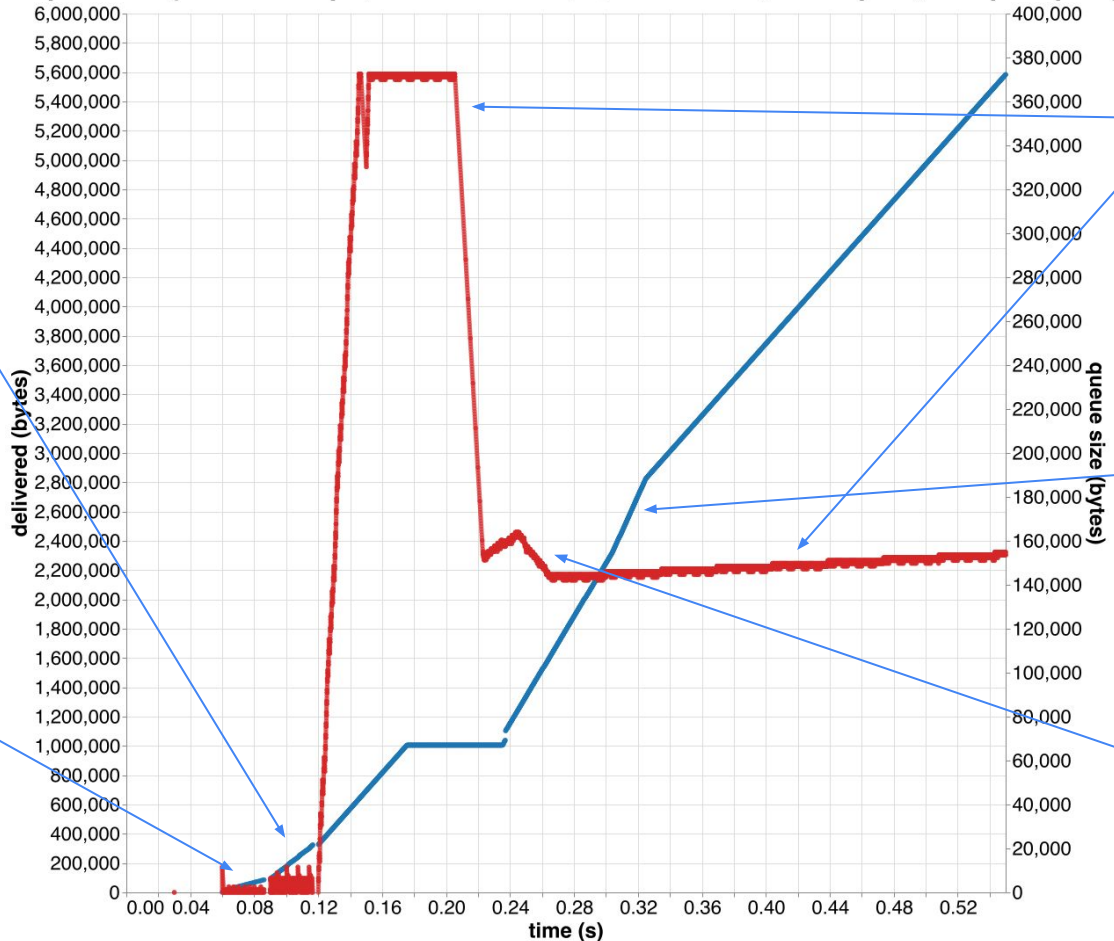


Handling Congestion: Safeguards

- SHOULD NOT reduce CWND below $pre_recovery_cwnd * beta / 3$
 - Rationale: $pre_recovery_cwnd$ would be no greater than $3 * fullBDP$. If the ack ratio is lower than $1/3$, some of the losses are not due to bottleneck queue overflow.
- MAY stop reducing below $beta * IW$
 - Rationale: on terribly slow paths, play on par with slow start.

Rapid Start (on simulator)

Rapid Start (BW=100Mbps, RTTidle=30ms, Queue=30ms, IW=30pkts, Jump=60pkts)



3x growth is faster

full-RTT pacing eliminates moments of network in idle

CWND lands at the correct value after only one recovery event

takes longer to fix losses, because more packets are lost

no excessive draining or bursts

Production Experiment

Production Experiment: Setup

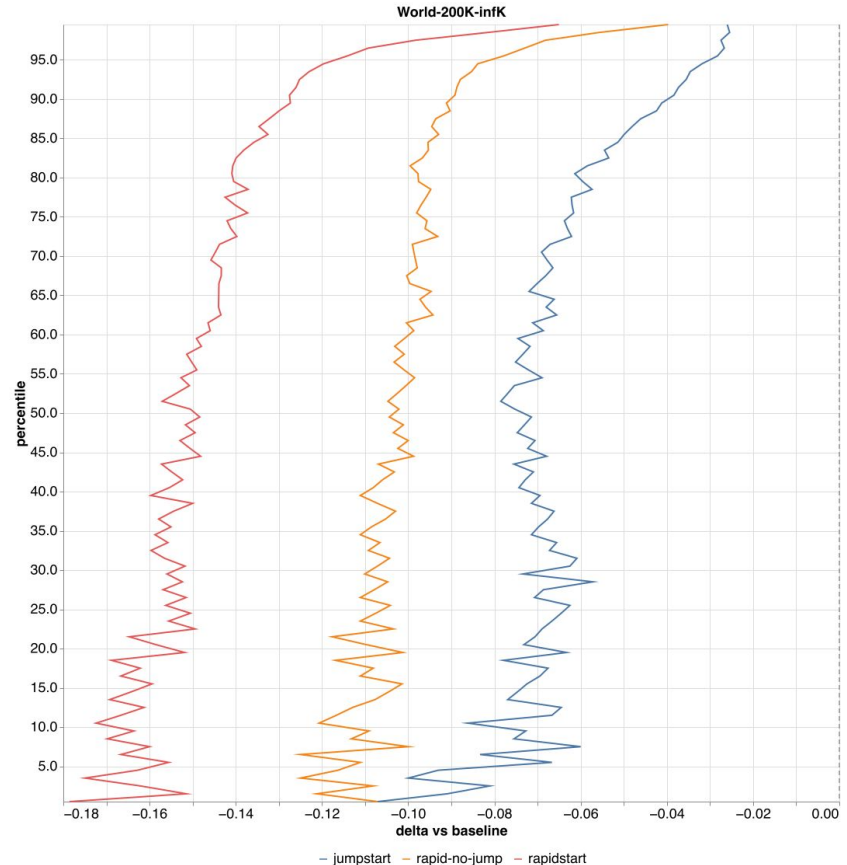
- Incoming connections divided into 4 groups:
 - Baseline: paced slow start (IW=30mtu)
 - Jumpstart: full-RT pacing (jump window=60mtu) but no 3x growth or smooth reduction^{note}
 - Rapid-wo-jump: Rapid Start without the first full-RT pacing
 - Rapidstart
- For HTTP/3 connections serving cached objects $\geq 200\text{KB}$ as their first request, measure TTLB of the cached object (i.e., when all bytes of the object are ACKed).
- Used 7 POPs for approx. one week:
 - East and South-East Asia, Western and Eastern Europe, Africa, North and South America

Note: The “jumpstart” group uses Careful Resume draft-04 with the jump window fixed to 60mtu; therefore, if packets sent during the full-RTT pacing stage (i.e., unvalidated phase) is lost, CWND is reduced to 0.5x to 0.7x of PipeSize, rather than applying beta.

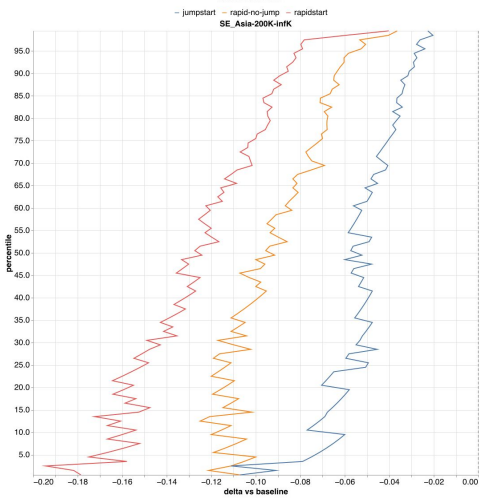
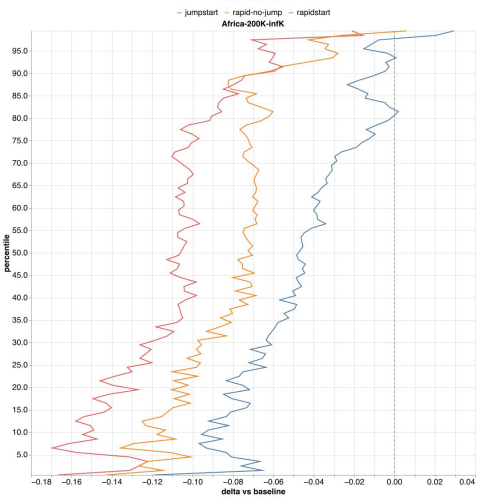
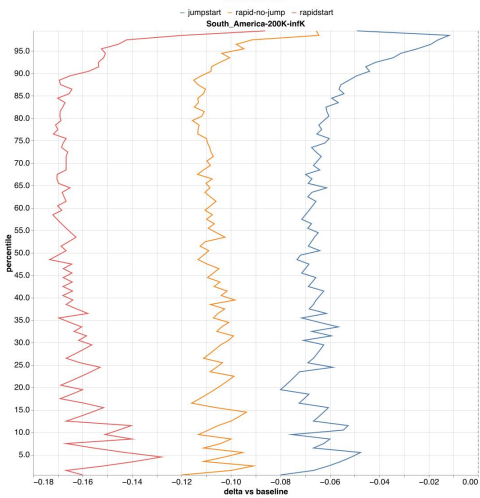
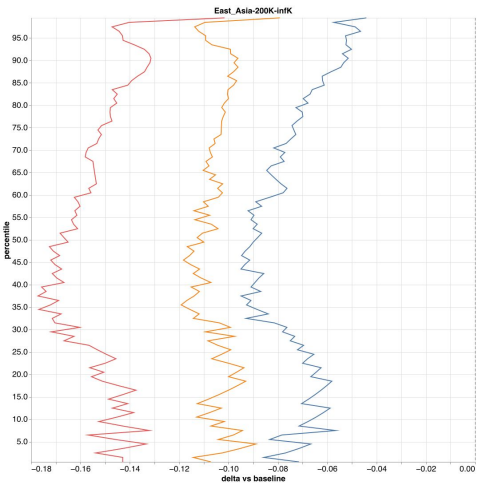
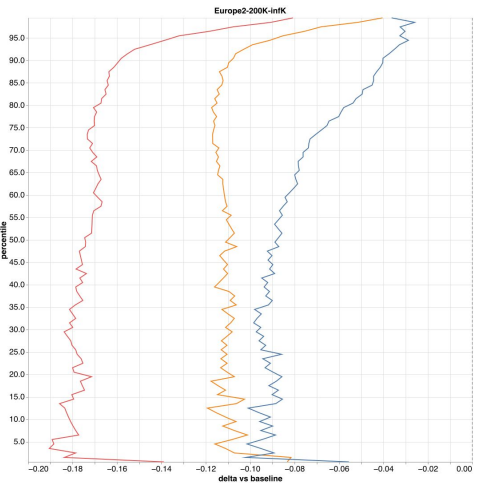
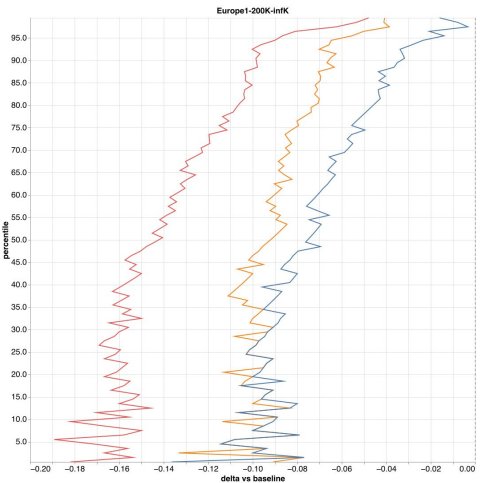
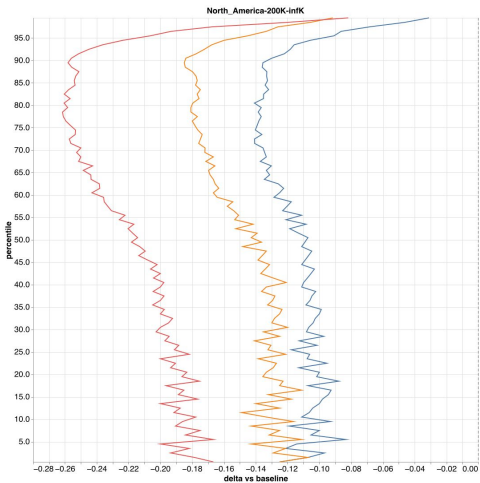
TTLB Reduction: Global

- All POPs
- All objects $\geq 200\text{KB}$
- With Rapid Start, TTLB is reduced by 14.7%

Note: thawtooth at the lower percentiles are due to the clock granularity being 1ms

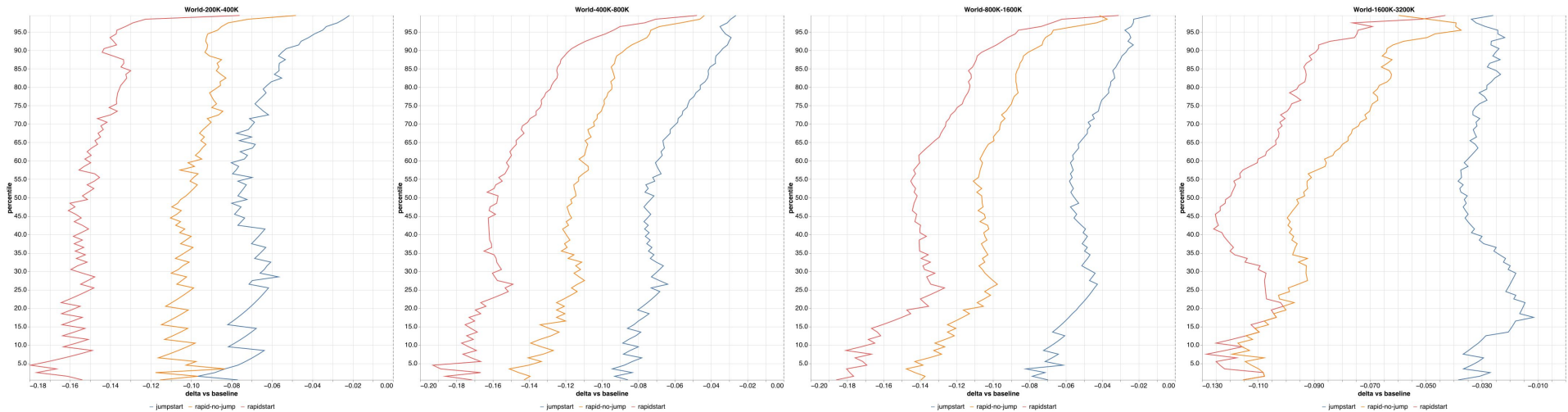


TTLB Reduction: per-POP



● TTLB reduction:
10.8% ~ 21.5%

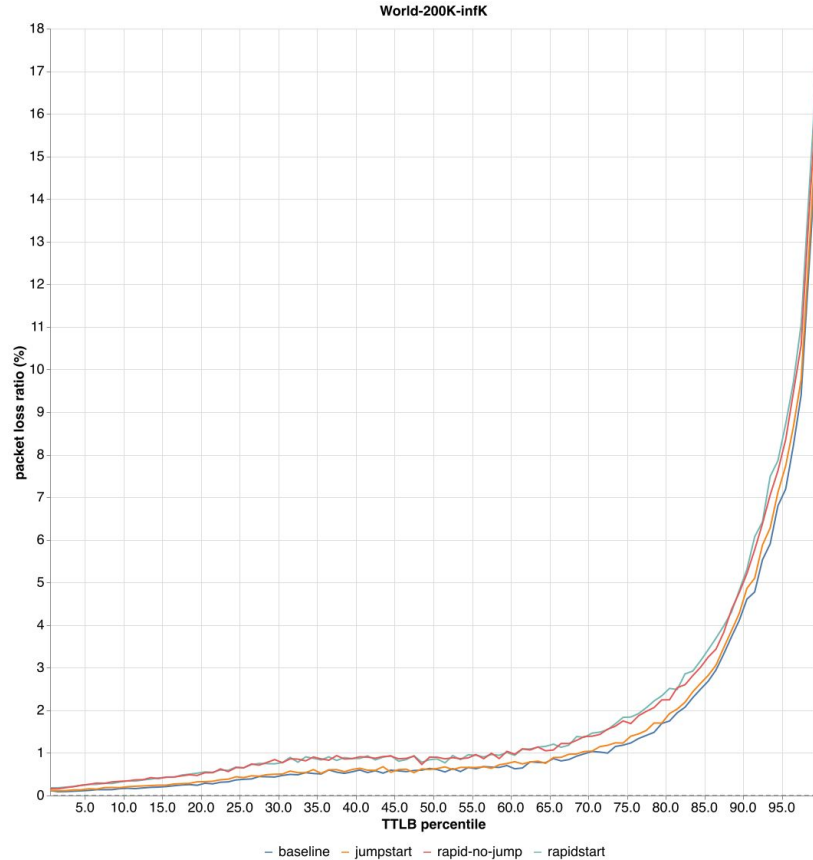
TTLB Reduction: by Object Size Bin



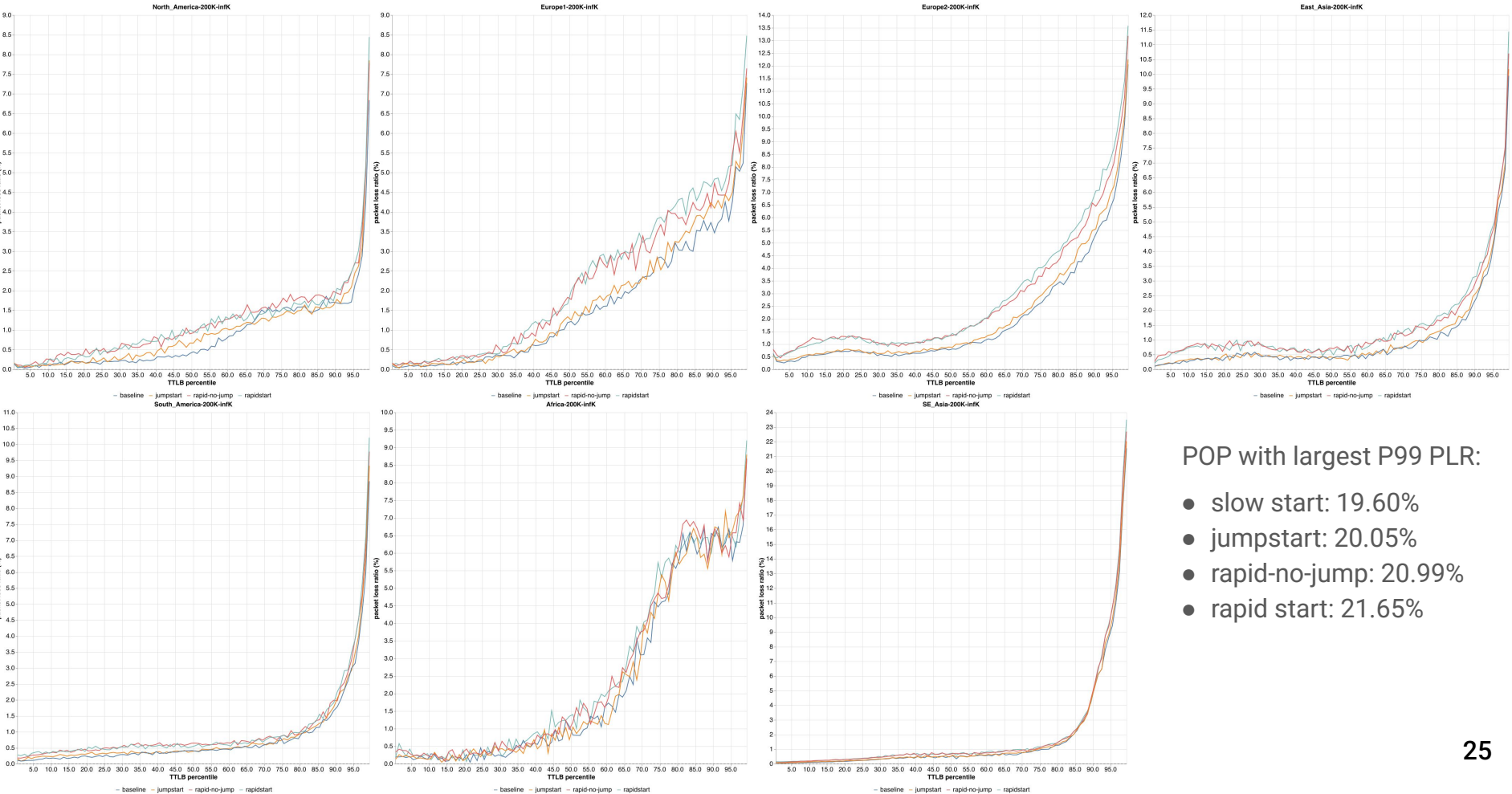
- Global data for different size bins:
200KB - 400KB / 400KB - 800KB / 800KB - 1.6MB / 1.6MB - 3.2MB
- TTLB reduction: 10.6% (1.6MB - 3.2MB) ~ 14.9% (200KB - 400KB)

Packet Loss Ratio: Global

	slow start (baseline)	jumpstart	rapid- no-jump	rapidstart
avg.	1.52%	1.61%	1.92%	1.98%
P50	0.62%	0.62%	0.90%	0.85%
P90	4.36%	4.57%	4.99%	5.06%
P99	13.80%	14.22%	14.97%	15.55%



Packet Loss Ratio: per-POP



POP with largest P99 PLR:

- slow start: 19.60%
- jumpstart: 20.05%
- rapid-no-jump: 20.99%
- rapid start: 21.65%

Production Experiment: Outcome

- TTLB reduction for objects $\geq 200\text{KB}$:
 - 14.7% globally
 - 10.8% ~ 21.5% depending on the location
 - 10.6% ~ 14.9% depending on size
- Full-RT pacing and 3x growth combined provides the best outcome.
- Packet loss ratio:
 - 1.98% globally (+0.46% from slow start)
 - P99 of the worst POP is 21.65%, +1.05% from slow start; i.e., on the slowest paths, Rapid Start remains comparably non-aggressive to slow start

Considerations

Interaction with ECN

- If congestion is indicated by ECN, no packets may be lost. This can cause CWND to remain as high as $\beta * cwnd_before_recovery$ and immediately trigger a second recovery period.
- However, if no packets are dropped, receivers do not need to wait for losses to be fixed. **Full-RTT pacing with 2x IW and 3x CWND increase of Rapid Start therefore provide reliable benefits when ECN is used.**

TCP

- Rapid Start requires the sender to determine bytes newly acked or lost for each ACK received.
- QUIC Loss Recovery naturally provides the required property.
- TCP stacks may not have the capability to expose the newly acked or lost bytes, which could limit Rapid Start's deployment on TCP.

Summary

Summary

- Rapid Start is a comprehensive approach, consisting of:
 - Full-RT pacing
 - 3x growth until queue build up
 - Quicker and smoother convergence of CWND comparable to that achieved by Congestion Avoidance
- Strong production results:
 - 14.7% average TTLB reduction for objects \geq 200KB
 - 10.8% ~ 21.5% depending on the location
 - 10.6% ~ 14.9% depending on size
 - The cost is a slight increase of average PLR from 1.52% to 1.98%

Next Steps

- Comments or Suggestions?
- WG Adoption?
- Our minimum objective: make the transport protocol community aware that a startup phase above 2x per RT is viable and will be used.
 - Because people occasionally try to build flow control under the assumption that the sender will increase its sending rate only as much as 2x per RT.
 - If such restrictive flow control becomes prevalent, any startup algorithm above 2x per RT becomes useless.