ccwg
Internet-Draft
Intended status: Standards Track
Expires: 27 August 2026

奥 一穂 (K. Oku)
Fastly
23 February 2026

Rapid Startup of Congestion Control
draft-kazuho-ccwg-rapid-start-01

Abstract

This document defines Rapid Start, a congestion-control startup algorithm. It starts by pacing the transmission of the initial congestion window over a full RTT, allowing an initial window up to 2× that of classic paced slow start at a comparable sending rate. It then grows the window by 3× per RTT until queue buildup is observed, after which it reverts to classic 2× slow start growth. When congestion is signaled, Rapid Start smoothly converges the window based on delivered data, avoiding bursts and underutilization, before handing over to ordinary congestion avoidance.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Congestion Control Working Group Working Group mailing list (ccwg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/ccwg/.

Source for this draft and an issue tracker can be found at https://github.com/kazuho/draft-kazuho-ccwg-rapid-start.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

New transport connections do not know the available bandwidth or the bandwidth-delay product (BDP) of the path, so TCP and QUIC start from an initial window and use an exponential startup (“slow start” ; Section 3.1 of [RFC5681], Section 7.3.1 of [RFC9002]) to probe for the bottleneck, often paired with pacing to reduce sender-side burstiness. In practice, paced slow start can still leave performance on the table:

* The sender typically starts by pacing packets for half an RTT and then pausing. When the bottleneck bandwidth is higher than the paced rate, the bottleneck can remain idle for the other half of each RTT.

* Even when the bottleneck is being utilized, utilization remains below capacity until queueing begins.

* When the initial window is much smaller than the path BDP, many round-trips are required to ramp up.

These effects are particularly detrimental to short-lived flows, which may only have a few round-trips to send data and therefore suffer disproportionately from underutilization during the startup.

Rapid Start retains the initial-window-based probing model but mitigates these issues. It paces the initial congestion window over the full estimated RTT, allowing an initial window up to 2× that of classic slow start at a comparable pacing rate. It then grows the congestion window by 3× per round-trip until queue buildup is observed, after which it reverts to classic 2× growth. When congestion is signaled, Rapid Start momentarily blocks sending to allow the bottleneck queue to drain slightly; it then resumes sending while reducing the window gradually in proportion to delivered and lost bytes. Doing so avoids burstiness as well as mitigating the risk of the bottleneck buffer becoming empty and the path becoming underutilized during recovery. After recovery, control is handed over to ordinary congestion avoidance, such as that of NewReno ([RFC6582]) and QUIC congestion control (Section 7 of [RFC9002]).

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Algorithm

This section describes the algorithm used by Rapid Start.

3.1. Sending in the First Round Trip

Rapid Start uses a more aggressive growth factor than classic slow start. When such growth is used, sending the initial congestion window as a short burst can make the sender observe a bottleneck overflow earlier than it would under evenly paced transmission. To ensure that Rapid Start observes the path’s queueing behavior rather than sender-side burstiness, the sender SHOULD pace the packets over a full RTT, using the current RTT estimate, when sending the first window’s worth of data.

When pacing over a full RTT, Rapid Start can use an initial window up to 2× that of classic slow start with pacing, because spreading the transmission over a full RTT (rather than half an RTT) yields a comparable pacing rate.

A sender using Careful Resume [CAREFUL-RESUME] satisfies these recommendations, because it requires that all packets sent in its Unvalidated Phase be paced based on current_rtt, regardless of prior knowledge.

3.2. Increasing the Congestion Window

Like Slow Start, Rapid Start increases the congestion window as packets are acknowledged. The difference is that when the path appears not to be building a queue, the sender uses a more aggressive startup increase.

The sender determines if the path is building a queue by comparing the recent minimum RTT (rtt_floor) against a calculated threshold (queue_buildup_thresh).

Let:

* min_rtt be the minimum RTT for the connection so far;

* rtt_floor be the smallest RTT over a recent observation window of approximately one round-trip. For example, an implementation might use a sliding time window of length min_rtt, or simply use currentRoundMinRTT tracked for sequence-based rounds in HyStart++ [RFC9406]; and

* queue_buildup_thresh be min(min_rtt + 4 ms, min_rtt * 1.10), where the additive term (+4 ms) and the multiplicative term (×1.10) are RECOMMENDED defaults.

If rtt_floor is no greater than queue_buildup_thresh, the sender increases the congestion window (cwnd) by 2 bytes for every byte that is newly acknowledged, which results in a 3× growth of cwnd per round-trip.

If rtt_floor is greater than queue_buildup_thresh, the sender SHOULD increase the congestion window as in classic slow start; i.e., by 1 byte for every byte that is newly acknowledged, which results in a 2× growth of cwnd per round-trip.

The construction of queue_buildup_thresh follows HyStart++’s bounded RTT-inflation approach, but uses a tighter RECOMMENDED threshold because the threshold is used to enable a more aggressive startup increase when queue buildup is unlikely, whereas HyStart++ uses RTT inflation to reduce growth by exiting slow start. Consequently, HyStart++ can be used in conjunction with Rapid Start.

3.3. Congestion Handling

When Rapid Start observes the first packet loss or an explicit congestion signal (e.g., ECN-CE), the sender enters the first recovery period (TCP: Section 3.2 of [RFC5681]; QUIC: Section 7.3.2 of [RFC9002]), but adjusts the congestion window in an alternative manner to smoothly converge after the more aggressive startup.

When entering the recovery period, the sender slightly scales down the current congestion window using a silence factor. As a result of this reduction, sending is momentarily blocked until bytes-in-flight is no greater than the reduced congestion window, allowing the bottleneck queue to be drained by a controlled amount.

```
cwnd *= silence_factor
```

Then, for each ACK that results in an update of acknowledged or lost bytes while in the first recovery period, the sender reduces the congestion window in proportion to newly acknowledged or newly declared lost bytes:

```
cwnd -= ack_factor * bytes_newly_acked
cwnd -= loss_factor * bytes_newly_lost
```

This approach ensures that, upon exiting the recovery period, the congestion window becomes a fraction of the full BDP (the sum of the idle BDP and the bottleneck queue size). At the same time, it keeps the silence period short enough that the sender is likely to resume transmission before the bottleneck is fully drained, even when the congestion window must be reduced significantly to compensate for the aggressive ramp-up.

To avoid overly sharp reduction caused by losses other than tail drops, the sender SHOULD NOT reduce the congestion window below

```
pre_recovery_cwnd * (silence_factor - 1/3 * ack_factor - 2/3 * loss_factor)
```

where pre_recovery_cwnd is the congestion window immediately before entering the recovery period. The coefficients are chosen to be consistent with the tail-drop model, which yields a loss ratio of 1 - 1 / G where G is the growth factor, using G = 3 (the largest growth factor used by Rapid Start). With the reduction factors defined in Section 3.3.1, this lower bound simplifies to pre_recovery_cwnd * beta / 3.

Separately, the sender MUST NOT reduce the congestion window below the minima specified by [RFC5681] or [RFC9002].

The sender MAY stop reducing the congestion window once it reaches the initial window multiplied by the window decrease factor. This allows the sender to keep the congestion window at least as large as classic slow start on paths with very small BDPs when transitioning to congestion avoidance.

Upon exiting the first recovery period, Rapid Start ends; thereafter, the congestion window is governed by the underlying congestion controller’s ordinary rules.

3.3.1. Deriving the Reduction Factors

The reduction factors are constants derived from the multiplicative window decrease factor (denoted beta) used by the congestion avoidance algorithm. They are chosen so that the recovery behavior described in Section 3.3 has the following properties:

* When the loss ratio is 2/3, the duration of the silence period is 1 - beta as a fraction of the full BDP, the same as during the congestion avoidance phase.

* Upon exiting the recovery period, the congestion window becomes the full BDP multiplied by beta, the same as during the congestion avoidance phase. This holds independent of the loss ratio during the recovery period, unless limited by the lower bounds on the congestion window.

Using a single constant K to distribute the window reduction across the send-blocking step and the per-ACK reductions, the factors are calculated as:

```
K                = 11/18
silence_factor  = beta + K * (1 - beta)
ack_factor      = K * (1 - beta)
loss_factor     = beta + K * (1 - beta)
```

Specifically, when beta is 0.5, the values are:

```
silence_factor  = 29/36
ack_factor      = 11/36
loss_factor     = 29/36
```

When beta is 0.7 (i.e., that of CUBIC [RFC9438]), the values are:

```
silence_factor  = 53/60
ack_factor      = 11/60
loss_factor     = 53/60
```

3.3.2. Interaction with ECN

Section 3.3.1 provides the rationale for the recovery behavior in terms of the full BDP (which, under loss-based detection, is estimated by probing until the bottleneck queue overflows and packets are dropped). However, when congestion happens on an ECN-capable path, it can be reported via CE marks without requiring packet loss. If Rapid Start enters a recovery period due to a CE mark but no packets are lost, then it exits recovery with a congestion window that is beta times its size immediately before entering recovery.

If the growth factor in the last round-trip was 3×, the congestion window upon entering recovery can be larger than with 2×, and therefore the congestion window at the end of recovery (beta times the entry size) can also be larger. This makes the next recovery period start sooner, but otherwise does not change the flow's behavior under ECN-signaled congestion pressure.

The other concern is buffer overflow before CE feedback is observed. Under 3× growth, the sender might build up a bottleneck queue that is twice as large as under 2× growth. However, even in the extreme case where a network's buffering margin is tightly provisioned for a target maximum RTT under conventional slow start (i.e., 2× growth), this larger queue buildup under 3× growth simply halves the loss-free RTT range: only connections with RTTs above half of that target maximum would be affected. In practice, networks do not generally provision buffers that tightly; with ECN, they can signal congestion without relying on drop, so leaving extra buffering margin typically has little downside. For these reasons, this overflow risk is limited in practice.

On loss-based paths, a more aggressive startup increases the likelihood of overflowing the bottleneck buffer and triggering packet drops, which delays delivery to the application due to retransmission. In contrast, on ECN-capable paths, congestion is typically signaled without relying on packet drops, so this loss-induced delivery delay mode is largely avoided. The benefits of faster growth of the congestion window are thus more reliable.

4. Considerations

Rapid Start’s startup and recovery behavior is driven by feedback from ACKs and loss detection. In practice, packet transmission and ACK reception can be affected by scheduling delays and buffering within the host network stack and along the path, which can make observed RTT signals noisier and reduce the smoothness of the algorithm’s response compared to an idealized per-packet model.

4.1. Considerations for TCP

Rapid Start’s recovery behavior is based on the QUIC-style model of tracking newly delivered and newly declared lost bytes as ACKs are processed. In QUIC, these quantities can be computed directly from acknowledged packet ranges and loss declarations over packet numbers. TCP implementations vary in how delivery and loss information is represented and exposed to congestion control; loss may be declared in multiple waves as the SACK scoreboard evolves, and accurately accounting newly declared lost bytes can be implementation-dependent (e.g., avoiding double-counting across reordering and retransmission heuristics); RTO-driven recovery can further reduce the timeliness and fidelity of these signals. As a result, TCP implementations might not be able to produce a reliable estimate of delivered and newly declared lost bytes during the first recovery period, especially when loss is high.

Therefore, it is up to each TCP implementation to determine whether and how the required delivered/lost byte accounting can be approximated robustly.

5. Security Considerations

TODO Security

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/rfc/rfc5681>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

7.2. Informative References

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP’s Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/rfc/rfc6582>.

[CAREFUL-RESUME]
Kuhn, N., Stephan, E., Fairhurst, G., Secchi, R., and C. Huitema, "Convergence of Congestion Control from Retained State", Work in Progress, Internet-Draft, draft-ietf-tsvwg-careful-resume-24, 1 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-careful-resume-24>.

[RFC9406] Balasubramanian, P., Huang, Y., and M. Olson, "HyStart++: Modified Slow Start for TCP", RFC 9406, DOI 10.17487/RFC9406, May 2023, <https://www.rfc-editor.org/rfc/rfc9406>.

[RFC9438] Xu, L., Ha, S., Rhee, I., Goel, V., and L. Eggert, Ed., "CUBIC for Fast and Long-Distance Networks", RFC 9438, DOI 10.17487/RFC9438, August 2023, <https://www.rfc-editor.org/rfc/rfc9438>.

[RFC9937] Mathis, M., Cardwell, N., Cheng, Y., and N. Dukkipati, "Proportional Rate Reduction (PRR)", RFC 9937, DOI 10.17487/RFC9937, December 2025, <https://www.rfc-editor.org/rfc/rfc9937>.

Acknowledgments

Rapid Start combines three ideas: (1) pacing the initial window over a full RTT, (2) a more aggressive startup increase when queue buildup is not observed, and (3) a recovery behavior that smoothly converges the congestion window.

Careful Resume [CAREFUL-RESUME] provides a predecessor for (1): it paces an initial window over a full RTT (based on a current RTT estimate) to avoid bursts when (re)starting. Rapid Start applies the same full-RTT pacing principle during starting.

"SUSS: Improving TCP Performance by Speeding Up Slow-Start" (Mahdi Arghavani et al.) advocates a similar approach for (2), built on top of HyStart that increases the congestion window by up to 4× per round-trip based on ACK dispersal and RTT.

Compared to SUSS, Rapid Start bases the (2) decision solely on RTT-based queue buildup detection, making it easier to integrate with other mechanisms and specifications such as HyStart++ [RFC9406].

For (3), Proportional Rate Reduction [RFC9937] is related work in that it regulates sending during recovery to avoid bursts and underutilization. Rapid Start differs by defining a startup-specific recovery behavior, allowing the congestion window to smoothly converge before handing over to congestion avoidance.

Author’s Address

Kazuho Oku
Fastly
Email: kazuhooku@gmail.com

Additional contact information:

奥 一穂
Fastly