ccwg
Internet-Draft
Intended status: Standards Track
Expires: 17 September 2026

奥 一穂 (K. Oku)
Fastly

16 March 2026

Rapid Startup of Congestion Control
draft-kazuho-ccwg-rapid-start-02

Abstract

This document defines Rapid Start, a congestion-control startup algorithm. It starts by pacing first full flight 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

Table of Contents

1. Introduction 2
2. Conventions and Definitions 3
3. Algorithm 3
3.1. Full-RTT Pacing 4
3.2. Increasing the Congestion Window 4
3.3. Congestion Handling 5
3.3.1. Reduction Factors 6
3.3.2. Interaction with ECN 7
4. Considerations 8
4.1. Considerations for TCP 8
5. Security Considerations 9
6. IANA Considerations 9
7. References 9
7.1. Normative References 9
7.2. Informative References 9
Appendix A. Deriving the Reduction Factors 10
Acknowledgments 12
Author's Address 13

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 first full flight over a 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. Full-RTT Pacing

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 it first sends more data than classic slow start with pacing would permit.

By pacing the packets over a full RTT, Rapid Start can use an initial window up to 2× that of classic slow start with pacing; spreading the transmission over a full RTT (rather than half an RTT) yields a comparable pacing rate. If this more aggressive transmission overshoots and congestion is signaled, Rapid Start compensates by reducing the congestion window as specified in Section 3.3.

Careful Resume [CAREFUL-RESUME] provides a compatible way to realize these recommendations: it can defer entry to its Unvalidated Phase until the sender first sends more data than normal congestion control would permit, and it requires packets sent in that phase to be paced based on the current RTT.

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. Specifically, it briefly pauses sending to allow the bottleneck queue to drain slightly, then gradually reduces the congestion window during recovery.

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 is designed so that, upon exiting the recovery period, the congestion window becomes the full BDP (the sum of the idle BDP and the bottleneck queue size) multiplied by beta, assuming that packets are dropped only due to bottleneck queue overflow; see

Appendix A. At the same time, it limits the silence period so that no more is drained from the bottleneck queue than under congestion avoidance, reducing the risk of underutilizing the link even when the congestion window must be reduced significantly to compensate for the aggressive ramp-up. The gradual reduction based on newly acknowledged and newly declared lost bytes also avoids burstiness when transmission resumes.

To avoid overly sharp reduction caused by losses other than those due to bottleneck queue overflow, the sender SHOULD NOT reduce the congestion window below

pre_recovery_cwnd * beta / 3

where pre_recovery_cwnd is the congestion window immediately before entering the recovery period. This lower bound corresponds to the target window size under Rapid Start's most aggressive growth factor (3×), for which the pre-recovery congestion window is three times the full BDP.

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. Reduction Factors

Assuming that losses are caused only by overflow of the bottleneck queue, the recovery algorithm described in Section 3.3 requires the following:

* The post-recovery congestion window becomes the full BDP multiplied by beta, independent of the loss ratio.

* The silence period drains at most 1 - beta times the full BDP from the bottleneck queue, matching congestion avoidance.

These requirements make Rapid Start match congestion avoidance in the post-recovery congestion window and in the maximum amount drained from the bottleneck queue.

The first requirement implies:

loss_factor = silence_factor = beta + ack_factor

Taken together, these requirements yield:

```
K               = 2/3
silence_factor  = beta + K * (1 - beta)
ack_factor      = K * (1 - beta)
loss_factor     = beta + K * (1 - beta)
```

When beta is 0.5, the values are:

```
silence_factor  = 5/6
ack_factor      = 1/3
loss_factor     = 5/6
```

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

```
silence_factor  = 9/10
ack_factor      = 1/5
loss_factor     = 9/10
```

Appendix A derives these formulas.

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>.

Appendix A. Deriving the Reduction Factors

This appendix derives the reduction factors specified in Section 3.3.1 and used by the recovery algorithm in Section 3.3.

For the derivation, consider a tail-drop model in which packets are lost only due to overflow of the bottleneck queue, and assume that the sender is congestion-window-limited when entering recovery. Let full_bdp be the path's full bandwidth-delay product. Let bytes_acked and bytes_lost denote the number of bytes newly acknowledged and newly declared lost during a recovery period.

Immediately before entering the recovery period, the congestion window equals the bytes in flight. By the end of the recovery period, those bytes are either newly acknowledged or newly declared lost. Therefore:

```
pre_recovery_cwnd = bytes_acked + bytes_lost
```

The congestion window after processing all acknowledgements and losses in the recovery period is:

```
post_recovery_cwnd = pre_recovery_cwnd * silence_factor
                     - bytes_acked * ack_factor
                     - bytes_lost * loss_factor
```

The first requirement of Section 3.3.1 is that the post-recovery congestion window becomes full_bdp * beta, independent of the loss ratio. Under the derivation assumptions, the bytes newly acknowledged during recovery equal one full BDP, i.e., bytes_acked = full_bdp. Substituting these relations into the expression for post_recovery_cwnd gives:

```
full_bdp * beta
    = pre_recovery_cwnd * silence_factor
        - bytes_acked * ack_factor
        - bytes_lost * loss_factor
    = bytes_acked * (silence_factor - ack_factor)
        + bytes_lost * (silence_factor - loss_factor)
```

For this expression to equal full_bdp * beta independent of the loss ratio, the coefficient of bytes_lost must be zero, and the coefficient of bytes_acked must equal beta. Therefore:

```
silence_factor - loss_factor = 0
silence_factor - ack_factor = beta
```

or equivalently:

```
loss_factor = silence_factor = beta + ack_factor
```

To derive factors satisfying the second requirement — that the silence period drains at most 1 - beta times the full BDP from the bottleneck queue, matching congestion avoidance — it is sufficient to consider the case that yields the longest silence period. In the recovery algorithm of Section 3.3, the sender resumes transmission when the congestion window catches up with bytes in flight. Under this recovery rule, the silence period becomes longer as the loss ratio increases. Because Rapid Start uses at most a growth factor of 3, the largest possible loss ratio is 2/3, at which point the silence period is the longest.

Because the loss ratio is 2/3, the congestion window immediately before entering recovery is three times the full BDP:

```
pre_recovery_cwnd = 3 * full_bdp
```

Let x be the number of newly acknowledged bytes, normalized by the full BDP, measured from the start of the recovery period. Because two bytes are newly declared lost for every byte that is newly acknowledged, after x * full_bdp bytes are acknowledged, bytes in flight is:

```
bytes_in_flight = 3 * (1 - x) * full_bdp
```

Over the same interval, the congestion window is:

```
cwnd = 3 * full_bdp * silence_factor
       - x * full_bdp * ack_factor
       - 2 * x * full_bdp * loss_factor
```

The sender resumes transmission once the congestion window catches up with bytes in flight. Satisfying the second requirement is therefore equivalent to requiring that transmission resumes when:

```
x = 1 - beta
```

At that point, cwnd = bytes_in_flight, therefore:

```
3 * (1 - x) * full_bdp
    = 3 * full_bdp * silence_factor
        - x * full_bdp * ack_factor
        - 2 * x * full_bdp * loss_factor
```

Dividing by full_bdp and substituting x = 1 - beta:

```
3 * beta
    = 3 * silence_factor
        - (1 - beta) * ack_factor
        - 2 * (1 - beta) * loss_factor
```

Substituting:

```
loss_factor = silence_factor = beta + ack_factor
```

into the previous equation yields:

```
3 * beta
    = 3 * (beta + ack_factor)
       - (1 - beta) * ack_factor
       - 2 * (1 - beta) * (beta + ack_factor)
```

which simplifies to:

```
ack_factor = (2 / 3) * (1 - beta)
```

Equivalently, using a constant K, the resulting formulas are:

```
K               = 2/3
silence_factor  = beta + K * (1 - beta)
ack_factor      = K * (1 - beta)
loss_factor     = beta + K * (1 - beta)
```

Acknowledgments

Rapid Start combines three ideas: (1) pacing the first full flight 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 the first flight 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 when 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