Stream: Internet Engineering Task Force (IETF)
RFC: 9406
Category: Standards Track
Published: May 2023
ISSN: 2070-1721
Authors: P. Balasubramanian *Confluent* Y. Huang *Microsoft* M. Olson *Microsoft*

RFC 9406
HyStart++: Modified Slow Start for TCP

Abstract

This document describes HyStart++, a simple modification to the slow start phase of congestion control algorithms. Slow start can overshoot the ideal send rate in many cases, causing high packet loss and poor performance. HyStart++ uses increase in round-trip delay as a heuristic to find an exit point before possible overshoot. It also adds a mitigation to prevent jitter from causing premature slow start exit.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9406.

Copyright Notice

Table of Contents

1. Introduction

[RFC5681] describes the slow start congestion control algorithm for TCP. The slow start algorithm is used when the congestion window (cwnd) is less than the slow start threshold (ssthresh). During slow start, in the absence of packet loss signals, TCP increases the cwnd exponentially to probe the network capacity. This fast growth can overshoot the ideal sending rate and cause significant packet loss that cannot always be recovered efficiently.

HyStart++ builds upon Hybrid Start (HyStart), originally described in [HyStart]. HyStart++ uses increase in round-trip delay as a signal to exit slow start before potential packet loss occurs as a result of overshoot. This is one of two algorithms specified in [HyStart] for finding a safe exit point for slow start. After the slow start exit, a new Conservative Slow Start (CSS) phase is used to determine whether the slow start exit was premature and to resume slow start. This mitigation improves performance in the presence of jitter. HyStart++ reduces packet loss and retransmissions, and improves goodput in lab measurements and real-world deployments.

While this document describes HyStart++ for TCP, it can also be used for other transport protocols that use slow start, such as QUIC [RFC9002] or the Stream Control Transmission Protocol (SCTP) [RFC9260].

2. Terminology

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. Definitions

To aid the reader, we repeat some definitions from [RFC5681]:

SENDER MAXIMUM SEGMENT SIZE (SMSS): The size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the Path MTU Discovery algorithm [RFC1191] [RFC4821], RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

RECEIVER MAXIMUM SEGMENT SIZE (RMSS): The size of the largest segment that the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup. Or, if the MSS option is not used, it is 536 bytes [RFC1122]. The size does not include the TCP/IP headers and options.

RECEIVER WINDOW (rwnd): The most recently advertised receiver window.

CONGESTION WINDOW (cwnd): A TCP state variable that limits the amount of data a TCP can send. At any given time, a TCP **MUST NOT** send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of the cwnd and rwnd.

4. HyStart++ Algorithm

4.1. Summary

[HyStart] specifies two algorithms (a "Delay Increase" algorithm and an "Inter-Packet Arrival" algorithm) to be run in parallel to detect that the sending rate has reached capacity. In practice, the Inter-Packet Arrival algorithm does not perform well and is not able to detect congestion early, primarily due to ACK compression. The idea of the Delay Increase algorithm is to look for spikes in RTT (round-trip time), which suggest that the bottleneck buffer is filling up.

In HyStart++, a TCP sender uses standard slow start and then uses the Delay Increase algorithm to trigger an exit from slow start. But instead of going straight from slow start to congestion avoidance, the sender spends a number of RTTs in a Conservative Slow Start (CSS) phase to determine whether the exit from slow start was premature. During CSS, the congestion window is grown exponentially in a fashion similar to regular slow start, but with a smaller exponential base, resulting in less aggressive growth. If the RTT reduces during CSS, it's concluded that the RTT spike was not related to congestion caused by the connection sending at a rate greater than the ideal send rate, and the connection resumes slow start. If the RTT inflation persists throughout CSS, the connection enters congestion avoidance.

4.2. Algorithm Details

The following pseudocode uses a limit, L, to control the aggressiveness of the cwnd increase during both standard slow start and CSS. While an arriving ACK may newly acknowledge an arbitrary number of bytes, the HyStart++ algorithm limits the number of those bytes applied to increase the cwnd to L*SMSS bytes.

lastRoundMinRTT and currentRoundMinRTT are initialized to infinity at the initialization time. currRTT is the RTT sampled from the latest incoming ACK and initialized to infinity.

```
lastRoundMinRTT = infinity
currentRoundMinRTT = infinity
currRTT = infinity
```

HyStart++ measures rounds using sequence numbers, as follows:

- Define windowEnd as a sequence number initialized to SND.NXT.
- When windowEnd is ACKed, the current round ends and windowEnd is set to SND.NXT.

At the start of each round during standard slow start [RFC5681] and CSS, initialize the variables used to compute the last round's and current round's minimum RTT:

```
lastRoundMinRTT = currentRoundMinRTT
currentRoundMinRTT = infinity
rttSampleCount = 0
```

For each arriving ACK in slow start, where N is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

Update the cwnd:

```
  cwnd = cwnd + min(N, L * SMSS)
```

Keep track of the minimum observed RTT:

```
  currentRoundMinRTT = min(currentRoundMinRTT, currRTT)
  rttSampleCount += 1
```

For rounds where at least N_RTT_SAMPLE RTT samples have been obtained and currentRoundMinRTT and lastRoundMinRTT are valid, check to see if delay increase triggers slow start exit:

```
if ((rttSampleCount >= N_RTT_SAMPLE) AND
    (currentRoundMinRTT != infinity) AND
    (lastRoundMinRTT != infinity))
  RttThresh = max(MIN_RTT_THRESH,
    min(lastRoundMinRTT / MIN_RTT_DIVISOR, MAX_RTT_THRESH))
  if (currentRoundMinRTT >= (lastRoundMinRTT + RttThresh))
    cssBaselineMinRtt = currentRoundMinRTT
    exit slow start and enter CSS
```

For each arriving ACK in CSS, where N is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

Update the cwnd:

```
cwnd = cwnd + (min(N, L * SMSS) / CSS_GROWTH_DIVISOR)
```

Keep track of the minimum observed RTT:

```
currentRoundMinRTT = min(currentRoundMinRTT, currRTT)
rttSampleCount += 1
```

For CSS rounds where at least N_RTT_SAMPLE RTT samples have been obtained, check to see if the current round's minRTT drops below baseline (cssBaselineMinRtt) indicating that slow start exit was spurious:

```
if (currentRoundMinRTT < cssBaselineMinRtt)
  cssBaselineMinRtt = infinity
  resume slow start including HyStart++
```

CSS lasts at most CSS_ROUNDS rounds. If the transition into CSS happens in the middle of a round, that partial round counts towards the limit.

If CSS_ROUNDS rounds are complete, enter congestion avoidance by setting the ssthresh to the current cwnd.

```
ssthresh = cwnd
```

If loss or Explicit Congestion Notification (ECN) marking is observed at any time during standard slow start or CSS, enter congestion avoidance by setting the ssthresh to the current cwnd.

```
ssthresh = cwnd
```

4.3. Tuning Constants and Other Considerations

It is **RECOMMENDED** that a HyStart++ implementation use the following constants:

```
MIN_RTT_THRESH = 4 msec
MAX_RTT_THRESH = 16 msec
MIN_RTT_DIVISOR = 8
N_RTT_SAMPLE = 8
CSS_GROWTH_DIVISOR = 4
CSS_ROUNDS = 5
L = infinity if paced, L = 8 if non-paced
```

These constants have been determined with lab measurements and real-world deployments. An implementation **MAY** tune them for different network characteristics.

The delay increase sensitivity is determined by MIN_RTT_THRESH and MAX_RTT_THRESH. Smaller values of MIN_RTT_THRESH may cause spurious exits from slow start. Larger values of MAX_RTT_THRESH may result in slow start not exiting until loss is encountered for connections on large RTT paths.

MIN_RTT_DIVISOR is a fraction of RTT to compute the delay threshold. A smaller value would mean a larger threshold and thus less sensitivity to delay increase, and vice versa.

While all TCP implementations are **REQUIRED** to take at least one RTT sample each round, implementations of HyStart++ are **RECOMMENDED** to take at least N_RTT_SAMPLE RTT samples. Using lower values of N_RTT_SAMPLE will lower the accuracy of the measured RTT for the round; higher values will improve accuracy at the cost of more processing.

The minimum value of CSS_GROWTH_DIVISOR **MUST** be at least 2. A value of 1 results in the same aggressive behavior as regular slow start. Values larger than 4 will cause the algorithm to be less aggressive and maybe less performant.

Smaller values of CSS_ROUNDS may miss detecting jitter, and larger values may limit performance.

Packet pacing [ASA00] is a possible mechanism to avoid large bursts and their associated harm. A paced TCP implementation **SHOULD** use L = infinity. Burst concerns are mitigated by pacing, and this setting allows for optimal cwnd growth on modern networks.

For TCP implementations that pace to mitigate burst concerns, L values smaller than infinity may suffer performance problems due to slow cwnd growth in high-speed networks. For non-paced TCP implementations, L values smaller than 8 may suffer performance problems due to slow cwnd growth in high-speed networks; L values larger than 8 may cause an increase in burstiness and thereby loss rates, and result in poor performance.

An implementation **SHOULD** use HyStart++ only for the initial slow start (when the ssthresh is at its initial value of arbitrarily high per [RFC5681]) and fall back to using standard slow start for the remainder of the connection lifetime. This is acceptable because subsequent slow starts will use the discovered ssthresh value to exit slow start and avoid the overshoot problem. An implementation **MAY** use HyStart++ to grow the restart window [RFC5681] after a long idle period.

In application-limited scenarios, the amount of data in flight could fall below the bandwidth-delay product (BDP) and result in smaller RTT samples, which can trigger an exit back to slow start. It is expected that a connection might oscillate between CSS and slow start in such scenarios. But this behavior will neither result in a connection prematurely entering congestion avoidance nor cause overshooting compared to slow start.

5. Deployments and Performance Evaluations

At the time of this writing, HyStart++ as described in this document has been default enabled for all TCP connections in the Windows operating system for over two years with pacing disabled and an actual L = 8.

In lab measurements with Windows TCP, HyStart++ shows goodput improvements as well as reductions in packet loss and retransmissions compared to standard slow start. For example, across a variety of tests on a 100 Mbps link with a bottleneck buffer size of bandwidth-delay product, HyStart++ reduces bytes retransmitted by 50% and retransmission timeouts (RTOs) by 36%.

In an A/B test where we compared an implementation of HyStart++ (based on an earlier draft version of this document) to standard slow start across a large Windows device population, out of 52 billion TCP connections, 0.7% of connections move from 1 RTO to 0 RTOs and another 0.7% of connections move from 2 RTOs to 1 RTO with HyStart++. This test did not focus on send-heavy connections, and the impact on send-heavy connections is likely much higher. We plan to conduct more such production experiments to gather more data in the future.

6. Security Considerations

HyStart++ enhances slow start and inherits the general security considerations discussed in [RFC5681].

An attacker can cause HyStart++ to exit slow start prematurely and impair the performance of a TCP connection by, for example, dropping data packets or their acknowledgments.

The ACK division attack outlined in [SCWA99] does not affect HyStart++ because the congestion window increase in HyStart++ is based on the number of bytes newly acknowledged in each arriving ACK rather than by a particular constant on each arriving ACK.

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[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/info/rfc2119>.

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

[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/info/rfc8174>.

8.2. Informative References

[ASA00] Aggarwal, A., Savage, S., and T. Anderson, "Understanding the performance of TCP pacing", Proceedings IEEE INFOCOM 2000, DOI 10.1109/INFCOM.2000.832483, March 2000, <https://doi.org/10.1109/INFCOM.2000.832483>.

[HyStart] Ha, S. and I. Rhee, "Taming the elephants: New TCP slow start", Computer Networks vol. 55, no. 9, pp. 2092-2110, DOI 10.1016/j.comnet.2011.01.014, June 2011, <https://doi.org/10.1016/j.comnet.2011.01.014>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/info/rfc4821>.

[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/info/rfc9002>.

[RFC9260] Stewart, R., Tüxen, M., and K. Nielsen, "Stream Control Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260, June 2022, <https://www.rfc-editor.org/info/rfc9260>.

[SCWA99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", ACM SIGCOMM Computer Communication Review, vol. 29, issue 5, pp. 71-78, DOI 10.1145/505696.505704, October 1999, <https://doi.org/10.1145/505696.505704>.

Acknowledgments

During the discussions of this work on the TCPM mailing list and in working group meetings, helpful comments, critiques, and reviews were received from (listed alphabetically by last name) Mark Allman, Bob Briscoe, Neal Cardwell, Yuchung Cheng, Junho Choi, Martin Duke, Reese Enghardt, Christian Huitema, Ilpo Järvinen, Yoshifumi Nishida, Randall Stewart, and Michael Tüxen.

Authors' Addresses

Praveen Balasubramanian
Confluent
899 West Evelyn Ave
Mountain View, CA 94041
United States of America
Email: pravb.ietf@gmail.com

Yi Huang
Microsoft
One Microsoft Way
Redmond, WA 98052
United States of America
Phone: +1 425 703 0447
Email: huanyi@microsoft.com

Matt Olson
Microsoft
One Microsoft Way
Redmond, WA 98052
United States of America
Phone: +1 425 538 8598
Email: maolson@microsoft.com