Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 18 December 2022

P. Balasubramanian
Confluent
Y. Huang
M. Olson
Microsoft

16 June 2022

HyStart++: Modified Slow Start for TCP
draft-ietf-tcpm-hystartplusplus-05

Abstract

This doument describes HyStart++, a simple modification to the slow start phase of congestion control algorithms. Traditional slow start can overshoot the ideal send rate in many cases, causing high packet loss and poor performance. HyStart++ uses a delay increase 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 Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 18 December 2022.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Definitions 3
4. HyStart++ Algorithm 3
4.1. Summary 3
4.2. Algorithm Details 4
4.3. Tuning constants and other considerations 6
5. Deployments and Performance Evaluations 7
6. Security Considerations 7
7. IANA Considerations 7
8. References 7
8.1. Normative References 7
8.2. Informative References 8
Authors' Addresses 8

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 absence of packet loss signals, TCP increases cwnd exponentially to probe the network capacity. This fast growth can overshoot the ideal sending rate and cause significant packet loss which cannot always be recovered efficiently.

HyStart++ uses delay increase 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]. After the slow start exit, a novel 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 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 which use slow start such as QUIC [RFC9002].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Definitions

We repeat here some definition from [RFC5681] to aid the reader.

SENDER MAXIMUM SEGMENT SIZE (SMSS): The SMSS is 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 [RFC1191, RFC4821] algorithm, RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

RECEIVER MAXIMUM SEGMENT SIZE (RMSS): The RMSS is the size of the largest segment 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 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 traditional 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

like in 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

For the pseudocode, we assume that Appropriate Byte Counting (as described in [RFC3465]) is in use and L is the cwnd increase limit as discussed in RFC 3465.

lastRoundMinRTT and currentRoundMinRTT are initialized to infinity at the initialization time

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:

> 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 minimum observed RTT

- currentRoundMinRTT = min(currentRoundMinRTT, currRTT)

- where currRTT is the RTT sampled from the latest incoming ACK

- rttSampleCount += 1

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

- if (rttSampleCount >= N_RTT_SAMPLE AND currentRoundMinRTT != infinity AND lastRoundMinRTT != infinity)

  - RttThresh = clamp(MIN_RTT_THRESH, lastRoundMinRTT / 8, MAX_RTT_THRESH)

  - if (currentRoundMinRTT >= (lastRoundMinRTT + RttThresh))

    + cssBaselineMinRtt = currentRoundMinRTT

    + exit slow start and enter CSS

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.

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 minimum observed RTT

- currentRoundMinRTT = min(currentRoundMinRTT, currRTT)

- where currRTT is the sampled RTT from the incoming ACK

- rttSampleCount += 1

For CSS rounds where N_RTT_SAMPLE RTT samples have been obtained, check if current round's minRTT drops below baseline indicating that HyStart exit was spurious.

- if (currentRoundMinRTT < cssBaselineMinRtt)

  - cssBaselineMinRtt = infinity

  - resume slow start including HyStart++

If CSS_ROUNDS rounds are complete, enter congestion avoidance.

* ssthresh = cwnd

If loss or ECN-marking is observed anytime during standard slow start or CSS, enter congestion avoidance.

* 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

* N_RTT_SAMPLE = 8

* CSS_GROWTH_DIVISOR = 4

* CSS_ROUNDS = 5

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.

A TCP implementation is required to take at least one RTT sample each round. 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.

An implementation SHOULD use HyStart++ only for the initial slow start (when ssthresh is at its initial value of arbitrarily high per [RFC5681]) and fall back to using traditional 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 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

As of the time of writing, HyStart++ as described in draft versions 01 through 04 was default enabled for all TCP connections in the Windows operating system for over three years with an actual L = 8. The original Hystart has been default-enabled for all TCP connections in the Linux operating system using the default congestion control module CUBIC ([RFC8312]) for a decade with an infinite L.

In lab measurements with Windows TCP, HyStart++ shows both goodput improvements as well as reductions in packet loss and retransmissions. 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 by 36%.

In an A/B test for HyStart++ draft 01 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% 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].

7. IANA Considerations

This document has no actions for IANA.

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>.

[RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <https://www.rfc-editor.org/info/rfc3465>.

[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>.

8.2. Informative References

[HyStart] Ha, S. and I. Ree, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", DOI 10.1145/1851182.1851192, International Workshop on Protocols for Fast Long-Distance Networks, 2008, <https://pdfs.semanticscholar.org/25e9/ef3f03315782c7f1cbcd31b587857adae7d1.pdf>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

[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>.

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 94052
United States of America
Phone: +1 425 703 0447
Email: huanyi@microsoft.com

Matt Olson
Microsoft
Phone: +1 425 538 8598
Email: maolson@microsoft.com