

Workgroup: TCPM
Internet-Draft: draft-ietf-tcpm-rfc8312bis-08
Obsoletes: [8312](#) (if approved)
Updates: [5681](#) (if approved)
Published: 30 May 2022
Intended Status: Standards Track
Expires: 1 December 2022
Authors: L. Xu S. Ha I. Rhee V. Goel
 UNL Colorado Bowery Apple Inc.
 L. Eggert, Ed.
 NetApp

CUBIC for Fast and Long-Distance Networks

Abstract

CUBIC is a standard TCP congestion control algorithm that uses a cubic function instead of a linear congestion window increase function to improve scalability and stability over fast and long-distance networks. CUBIC has been adopted as the default TCP congestion control algorithm by the Linux, Windows, and Apple stacks.

This document updates the specification of CUBIC to include algorithmic improvements based on these implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting RFC 8312. This also requires updating RFC 5681, to allow for CUBIC's occasionally more aggressive sending behavior.

About This Document

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

Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-tcpm-rfc8312bis/>.

Discussion of this document takes place on the TCPM Working Group mailing list (<mailto:tcpm@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/tcpm/>.

Source for this draft and an issue tracker can be found at <https://github.com/NTAP/rfc8312bis>.

Note to the RFC Editor

xml2rfc currently renders `` in the XML by surrounding the corresponding text with underscores. This is highly distracting; please manually remove the underscores when doing the final edits to the text version of this document.

(There is an issue open against xml2rfc to stop doing this in the future: <https://trac.tools.ietf.org/tools/xml2rfc/trac/ticket/596>)

Also, please manually change "Figure" to "Equation" for all artwork with anchors beginning with "eq" - xml2rfc doesn't seem to be able to do this.

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 1 December 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

- [1. Introduction](#)
- [2. Conventions](#)
- [3. Design Principles of CUBIC](#)
 - [3.1. Principle 1 for the CUBIC Increase Function](#)
 - [3.2. Principle 2 for Reno-Friendliness](#)
 - [3.3. Principle 3 for RTT Fairness](#)
 - [3.4. Principle 4 for the CUBIC Decrease Factor](#)

[4. CUBIC Congestion Control](#)

[4.1. Definitions](#)

[4.1.1. Constants of Interest](#)

[4.1.2. Variables of Interest](#)

[4.2. Window Increase Function](#)

[4.3. Reno-Friendly Region](#)

[4.4. Concave Region](#)

[4.5. Convex Region](#)

[4.6. Multiplicative Decrease](#)

[4.7. Fast Convergence](#)

[4.8. Timeout](#)

[4.9. Spurious Congestion Events](#)

[4.9.1. Spurious timeout](#)

[4.9.2. Spurious loss detected by acknowledgements](#)

[4.10. Slow Start](#)

[5. Discussion](#)

[5.1. Fairness to Reno](#)

[5.2. Using Spare Capacity](#)

[5.3. Difficult Environments](#)

[5.4. Investigating a Range of Environments](#)

[5.5. Protection against Congestion Collapse](#)

[5.6. Fairness within the Alternative Congestion Control Algorithm](#)

[5.7. Performance with Misbehaving Nodes and Outside Attackers](#)

[5.8. Behavior for Application-Limited Flows](#)

[5.9. Responses to Sudden or Transient Events](#)

[5.10. Incremental Deployment](#)

[6. Security Considerations](#)

[7. IANA Considerations](#)

[8. References](#)

[8.1. Normative References](#)

[8.2. Informative References](#)

[Appendix A. Acknowledgments](#)

[Appendix B. Evolution of CUBIC](#)

[B.1. Since draft-ietf-tcpm-rfc8312bis-07](#)

[B.2. Since draft-ietf-tcpm-rfc8312bis-06](#)

[B.3. Since draft-ietf-tcpm-rfc8312bis-05](#)

[B.4. Since draft-ietf-tcpm-rfc8312bis-04](#)

[B.5. Since draft-ietf-tcpm-rfc8312bis-03](#)

[B.6. Since draft-ietf-tcpm-rfc8312bis-02](#)

[B.7. Since draft-ietf-tcpm-rfc8312bis-01](#)

[B.8. Since draft-ietf-tcpm-rfc8312bis-00](#)

[B.9. Since draft-eggert-tcpm-rfc8312bis-03](#)

[B.10. Since draft-eggert-tcpm-rfc8312bis-02](#)

[B.11. Since draft-eggert-tcpm-rfc8312bis-01](#)

[B.12. Since draft-eggert-tcpm-rfc8312bis-00](#)

[B.13. Since RFC8312](#)

[B.14. Since the Original Paper](#)

[Authors' Addresses](#)

1. Introduction

CUBIC has been adopted as the default TCP congestion control algorithm in the Linux, Windows, and Apple stacks, and has been used and deployed globally. Extensive, decade-long deployment experience in vastly different Internet scenarios has convincingly demonstrated that CUBIC is safe for deployment on the global Internet and delivers substantial benefits over classical Reno congestion control [RFC5681]. It is therefore to be regarded as the currently most widely deployed standard for TCP congestion control. CUBIC can also be used for other transport protocols such as QUIC [RFC9000] and SCTP [RFC4960] as a default congestion controller.

The design of CUBIC was motivated by the well-documented problem classical Reno TCP has with low utilization over fast and long-distance networks [K03][RFC3649]. This problem arises from a slow increase of the congestion window following a congestion event in a network with a large bandwidth-delay product (BDP). [HLRX07] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets. This problem is equally applicable to all Reno-style standards and their variants, including TCP-Reno [RFC5681], TCP-NewReno [RFC6582] [RFC6675], SCTP [RFC4960], TFRC [RFC5348], and QUIC congestion control [RFC9002], which use the same linear increase function for window growth. We refer to all Reno-style standards and their variants collectively as "Reno" below.

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of classical Reno to remedy this problem. Specifically, CUBIC uses a cubic function instead of the linear window increase function of Reno to improve scalability and stability under fast and long-distance networks.

This document updates the specification of CUBIC to include algorithmic improvements based on the Linux, Windows, and Apple implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting [RFC8312]. This requires an update to [RFC5681], which limits the aggressiveness of Reno TCP implementations in its Section 3. Since CUBIC is occasionally more aggressive than the [RFC5681] algorithms, this document updates [RFC5681] to allow for CUBIC's behavior.

Binary Increase Congestion Control (BIC-TCP) [XHR04], a predecessor of CUBIC, was selected as the default TCP congestion control algorithm by Linux in the year 2005 and had been used for several years by the Internet community at large.

CUBIC uses a similar window increase function as BIC-TCP and is designed to be less aggressive and fairer to Reno in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and round-trip time (RTT) fairness.

[[RFC5033](#)] documents the IETF's best current practices for specifying new congestion control algorithms, specifically, ones that differ from the general congestion control principles outlined in [[RFC2914](#)]. It describes what type of evaluation is expected by the IETF to understand the suitability of a new congestion control algorithm and the process to enable a specification to be approved for widespread deployment in the global Internet.

There are areas in which CUBIC differs from the congestion control algorithms previously published in standards-track RFCs; those changes are specified in this document. However, it is not obvious that these changes go beyond the general congestion control principles outlined in [[RFC2914](#)], so the process in [[RFC5033](#)] may not apply.

Also, the wide deployment of CUBIC on the Internet was driven by direct adoption in most of the popular operating systems, and did not follow the practices documented in [[RFC5033](#)]. However, due to the resulting Internet-scale deployment experience over a long period of time, the IETF has determined that CUBIC may be published as a standards-track specification. This decision by the IETF does not alter the general guidance in [[RFC2914](#)].

In the following sections, we first briefly explain the design principles of CUBIC, then provide the exact specification of CUBIC, and finally discuss the safety features of CUBIC following the guidelines specified in [[RFC5033](#)].

2. Conventions

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. Design Principles of CUBIC

CUBIC is designed according to the following design principles:

Principle 1: For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function.

Principle 2:

To be Reno-friendly, CUBIC is designed to behave like Reno in networks with short RTTs and small bandwidth where Reno performs well.

Principle 3: For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs.

Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed.

3.1. Principle 1 for the CUBIC Increase Function

For better network utilization and stability, CUBIC [[HRX08](#)] uses a cubic window increase function in terms of the elapsed time from the last congestion event. While most alternative congestion control algorithms to Reno increase the congestion window using convex functions, CUBIC uses both the concave and convex profiles of a cubic function for window growth.

After a window reduction in response to a congestion event detected by duplicate ACKs, Explicit Congestion Notification-Echo (ECN-Echo, ECE) ACKs [[RFC3168](#)], TCP RACK [[RFC8985](#)] or QUIC loss detection [[RFC9002](#)], CUBIC remembers the congestion window size at which it received the congestion event and performs a multiplicative decrease of the congestion window. When CUBIC enters into congestion avoidance, it starts to increase the congestion window using the concave profile of the cubic function. The cubic function is set to have its plateau at the remembered congestion window size, so that the concave window increase continues until then. After that, the cubic function turns into a convex profile and the convex window increase begins.

This style of window adjustment (concave and then convex) improves the algorithm stability while maintaining high network utilization [[CEHRX09](#)]. This is because the window size remains almost constant, forming a plateau around the remembered congestion window size of the last congestion event, where network utilization is deemed highest. Under steady state, most window size samples of CUBIC are close to that remembered congestion window size, thus promoting high network utilization and stability.

Note that congestion control algorithms that only use convex functions to increase the congestion window size have their maximum increments around the remembered congestion window size of the last congestion event, and thus introduce many packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

3.2. Principle 2 for Reno-Friendliness

CUBIC promotes per-flow fairness to Reno. Note that Reno performs well over paths with short RTTs and small bandwidths (or small BDPs). There is only a scalability problem in networks with long RTTs and large bandwidths (or large BDPs).

A congestion control algorithm designed to be friendly to Reno on a per-flow basis must increase its congestion window less aggressively in small BDP networks than in large BDP networks.

The aggressiveness of CUBIC mainly depends on the maximum window size before a window reduction, which is smaller in small-BDP networks than in large-BDP networks. Thus, CUBIC increases its congestion window less aggressively in small-BDP networks than in large-BDP networks.

Furthermore, in cases when the cubic function of CUBIC would increase the congestion window less aggressively than Reno, CUBIC simply follows the window size of Reno to ensure that CUBIC achieves at least the same throughput as Reno in small-BDP networks. We call this region where CUBIC behaves like Reno the "Reno-friendly region".

3.3. Principle 3 for RTT Fairness

Two CUBIC flows with different RTTs have a throughput ratio that is linearly proportional to the inverse of their RTT ratio, where the throughput of a flow is approximately the size of its congestion window divided by its RTT.

Specifically, CUBIC maintains a window increase rate independent of RTTs outside the Reno-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the Reno-friendly region.

This notion of a linear throughput ratio is similar to that of Reno under high statistical multiplexing where packet loss is independent of individual flow rates. However, under low statistical multiplexing, the throughput ratio of Reno flows with different RTTs is quadratically proportional to the inverse of their RTT ratio [[XHR04](#)].

CUBIC always ensures a linear throughput ratio independent of the amount of statistical multiplexing. This is an improvement over Reno. While there is no consensus on particular throughput ratios for different RTT flows, we believe that over wired Internet paths, use of a linear throughput ratio seems more reasonable than equal throughputs (i.e., the same throughput for flows with different RTTs) or a higher-order throughput ratio (e.g., a quadratical

throughput ratio of Reno under low statistical multiplexing environments).

3.4. Principle 4 for the CUBIC Decrease Factor

To balance between scalability and convergence speed, CUBIC sets the multiplicative window decrease factor to 0.7, whereas Reno uses 0.5.

While this improves the scalability of CUBIC, a side effect of this decision is slower convergence, especially under low statistical multiplexing. This design choice is following the observation that HighSpeed TCP (HSTCP) [[RFC3649](#)] and other approaches (e.g., [[GV02](#)]) made: the current Internet becomes more asynchronous with less frequent loss synchronizations under high statistical multiplexing.

In such environments, even strict Multiplicative-Increase Multiplicative-Decrease (MIMD) can converge. CUBIC flows with the same RTT always converge to the same throughput independent of statistical multiplexing, thus achieving intra-algorithm fairness. We also find that in environments with sufficient statistical multiplexing, the convergence speed of CUBIC is reasonable.

4. CUBIC Congestion Control

In this section, we discuss how the congestion window is updated during the different stages of the CUBIC congestion controller.

4.1. Definitions

The unit of all window sizes in this document is segments of the maximum segment size (MSS), and the unit of all times is seconds. Implementations can use bytes to express window sizes, which would require factoring in the maximum segment size wherever necessary and replacing *segments_acked* with the number of bytes acknowledged in [Figure 4](#).

4.1.1. Constants of Interest

β_{cubic} : CUBIC multiplicative decrease factor as described in [Section 4.6](#).

α_{cubic} : CUBIC additive increase factor used in Reno-friendly region as described in [Section 4.3](#).

C : constant that determines the aggressiveness of CUBIC in competing with other congestion control algorithms in high BDP networks. Please see [Section 5](#) for more explanation on how it is set. The unit for C is

$$\frac{\text{segment}}{\text{second}^3}$$

4.1.2. Variables of Interest

This section defines the variables required to implement CUBIC:

RTT: Smoothed round-trip time in seconds, calculated as described in [\[RFC6298\]](#).

cwnd: Current congestion window in segments.

ssthresh: Current slow start threshold in segments.

W_{max}: Size of *cwnd* in segments just before *cwnd* was reduced in the last congestion event when fast convergence is disabled. However, if fast convergence is enabled, the size may be further reduced based on the current saturation point.

K: The time period in seconds it takes to increase the congestion window size at the beginning of the current congestion avoidance stage to *W_{max}*.

current_time: Current time of the system in seconds.

epoch_{start}: The time in seconds at which the current congestion avoidance stage started.

cwnd_{start}: The *cwnd* at the beginning of the current congestion avoidance stage, i.e., at time *epoch_{start}*.

W_{cubic}(t): The congestion window in segments at time *t* in seconds based on the cubic increase function, as described in [Section 4.2](#).

target: Target value of congestion window in segments after the next RTT, that is, *W_{cubic}(t + RTT)*, as described in [Section 4.2](#).

W_{est}: An estimate for the congestion window in segments in the Reno-friendly region, that is, an estimate for the congestion window of Reno.

segments_acked: Number of MSS-sized segments acked when a "new ACK" is received, i.e., an ACK that cumulatively acknowledges the delivery of new data. This number will be a decimal value when a new ACK acknowledges an amount of data that is not MSS-sized. Specifically, it can be less than 1 when a new ACK acknowledges a segment smaller than the MSS.

4.2. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of Reno by increasing the congestion window only at the reception of a new ACK. It does not make any changes to the TCP Fast Recovery and Fast Retransmit algorithms [RFC6582][RFC6675].

During congestion avoidance, after a congestion event is detected by mechanisms described in [Section 3.1](#), CUBIC uses a window increase function different from Reno.

CUBIC uses the following window increase function:

$$W_{\text{cubic}}(t) = C * (t - K)^3 + W_{\text{max}}$$

Figure 1

where t is the elapsed time in seconds from the beginning of the current congestion avoidance stage, that is,

$$t = \text{current_time} - \text{epoch}_{\text{start}}$$

and where $\text{epoch}_{\text{start}}$ is the time at which the current congestion avoidance stage starts. K is the time period that the above function takes to increase the congestion window size at the beginning of the current congestion avoidance stage to W_{max} if there are no further congestion events and is calculated using the following equation:

$$K = \sqrt[3]{\frac{W_{\text{max}} - \text{cwnd}_{\text{start}}}{C}}$$

Figure 2

where $\text{cwnd}_{\text{start}}$ is the congestion window at the beginning of the current congestion avoidance stage.

Upon receiving a new ACK during congestion avoidance, CUBIC computes the *target* congestion window size after the next *RTT* using [Figure 1](#) as follows, where *RTT* is the smoothed round-trip time. The lower and upper bounds below ensure that CUBIC's congestion window increase rate is non-decreasing and is less than the increase rate of slow start [SXEZ19].

$$\text{target} = \begin{cases} \text{cwnd} & \text{if } W_{\text{cubic}}(t + \text{RTT}) < \text{cwnd} \\ 1.5 * \text{cwnd} & \text{if } W_{\text{cubic}}(t + \text{RTT}) > 1.5 * \text{cwnd} \\ W_{\text{cubic}}(t + \text{RTT}) & \text{otherwise} \end{cases}$$

The elapsed time t in [Figure 1](#) **MUST NOT** include periods during which *cwnd* has not been updated due to application-limited behavior (see [Section 5.8](#)).

Depending on the value of the current congestion window size $cwnd$, CUBIC runs in three different regions:

1. The Reno-friendly region, which ensures that CUBIC achieves at least the same throughput as Reno.
2. The concave region, if CUBIC is not in the Reno-friendly region and $cwnd$ is less than W_{max} .
3. The convex region, if CUBIC is not in the Reno-friendly region and $cwnd$ is greater than W_{max} .

Below, we describe the exact actions taken by CUBIC in each region.

4.3. Reno-Friendly Region

Reno performs well in certain types of networks, for example, under short RTTs and small bandwidths (or small BDPs). In these networks, CUBIC remains in the Reno-friendly region to achieve at least the same throughput as Reno.

The Reno-friendly region is designed according to the analysis in [FHP00], which studies the performance of an AIMD algorithm with an additive factor of α (segments per RTT) and a multiplicative factor of β , denoted by AIMD(α , β). p is the packet loss rate. Specifically, the average congestion window size of AIMD(α , β) can be calculated using Figure 3.

$$AVG_AIMD(\alpha, \beta) = \sqrt{\frac{\alpha * (1 + \beta)}{2 * (1 - \beta) * p}}$$

Figure 3

By the same analysis, to achieve the same average window size as Reno that uses AIMD(1, 0.5), α must be equal to,

$$3 * \frac{1 - \beta}{1 + \beta}$$

Thus, CUBIC uses Figure 4 to estimate the window size W_{est} in the Reno-friendly region with

$$\alpha_{cubic} = 3 * \frac{1 - \beta_{cubic}}{1 + \beta_{cubic}}$$

which achieves the same average window size as Reno. When receiving a new ACK in congestion avoidance (where $cwnd$ could be greater than or less than W_{max}), CUBIC checks whether $W_{cubic}(t)$ is less than W_{est} .

If so, CUBIC is in the Reno-friendly region and *cwnd* **SHOULD** be set to W_{est} at each reception of a new ACK.

W_{est} is set equal to $cwnd_{start}$ at the start of the congestion avoidance stage. After that, on every new ACK, W_{est} is updated using [Figure 4](#). Note that this equation is for a connection where Appropriate Byte Counting (ABC) [[RFC3465](#)] is disabled. For a connection with ABC enabled, this equation **SHOULD** be adjusted by using the number of acknowledged bytes instead of acknowledged segments. Also note that this equation works for connections with enabled or disabled Delayed ACKs [[RFC5681](#)], as *segments_acked* will be different based on the segments actually acknowledged by a new ACK.

$$W_{est} = W_{est} + \alpha_{cubic} * \frac{segments_acked}{cwnd}$$

Figure 4

Note that once W_{est} reaches W_{max} , that is, $W_{est} \geq W_{max}$, CUBIC needs to start probing to determine the new value of W_{max} . At this point, α_{cubic} **SHOULD** be set to 1 to ensure that CUBIC can achieve the same congestion window increment as Reno, which uses AIMD(1, 0.5).

4.4. Concave Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and *cwnd* is less than W_{max} , then CUBIC is in the concave region. In this region, *cwnd* **MUST** be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where *target* is calculated as described in [Section 4.2](#).

4.5. Convex Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and *cwnd* is larger than or equal to W_{max} , then CUBIC is in the convex region.

The convex region indicates that the network conditions might have changed since the last congestion event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, some amount of perturbation is always possible without causing a major change in available bandwidth.

Unless it is overridden by the AIMD window increase, CUBIC is very careful in this region. The convex profile aims to increase the

window very slowly at the beginning when $cwnd$ is around W_{max} and then gradually increases its rate of increase. We also call this region the "maximum probing phase", since CUBIC is searching for a new W_{max} . In this region, $cwnd$ **MUST** be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where $target$ is calculated as described in [Section 4.2](#).

4.6. Multiplicative Decrease

When a congestion event is detected by mechanisms described in [Section 3.1](#), CUBIC updates W_{max} and reduces $cwnd$ and $ssthresh$ immediately as described below. In case of packet loss, the sender **MUST** reduce $cwnd$ and $ssthresh$ immediately upon entering loss recovery, similar to [\[RFC5681\]](#) (and [\[RFC6675\]](#)). Note that other mechanisms, such as Proportional Rate Reduction [\[RFC6937\]](#), can be used to reduce the sending rate during loss recovery more gradually. The parameter β_{cubic} **SHOULD** be set to 0.7, which is different from the multiplicative decrease factor used in [\[RFC5681\]](#) (and [\[RFC6675\]](#)) during fast recovery.

In [Figure 5](#), $flight_size$ is the amount of outstanding data in the network, as defined in [\[RFC5681\]](#). Note that a rate-limited application with idle periods or periods when unable to send at the full rate permitted by $cwnd$ may easily encounter notable variations in the volume of data sent from one RTT to another, resulting in $flight_size$ that is significantly less than $cwnd$ on a congestion event. This may decrease $cwnd$ to a much lower value than necessary. To avoid suboptimal performance with such applications, the mechanisms described in [\[RFC7661\]](#) can be used to mitigate this issue as it would allow using a value between $cwnd$ and $flight_size$ to calculate the new $ssthresh$ in [Figure 5](#). The congestion window growth mechanism defined in [\[RFC7661\]](#) is safe to use even when $cwnd$ is greater than the receive window as it validates $cwnd$ based on the amount of data acknowledged by the network in an RTT which implicitly accounts for the allowed receive window. Some implementations of CUBIC currently use $cwnd$ instead of $flight_size$ when calculating a new $ssthresh$ using [Figure 5](#).

```

ssthresh = flight_size *  $\beta_{cubic}$  // new ssthresh
cwnd =    { max(ssthresh, 2) // reduction on packet loss, cwnd is at least 2 MSS
           { max(ssthresh, 1) // reduction on ECE, cwnd is at least 1 MSS
ssthresh = max(ssthresh, 2) // ssthresh is at least 2 MSS

```

Figure 5

A side effect of setting β_{cubic} to a value bigger than 0.5 is slower convergence. We believe that while a more adaptive setting of β_{cubic} could result in faster convergence, it will make the analysis of CUBIC much harder.

Note that CUBIC **MUST** continue to reduce $cwnd$ in response to congestion events due to ECN-Echo ACKs until it reaches a value of 1 MSS. If congestion events indicated by ECN-Echo ACKs persist, a sender with a $cwnd$ of 1 MSS **MUST** reduce its sending rate even further. It can achieve that by using a retransmission timer with exponential backoff, as described in [RFC3168].

4.7. Fast Convergence

To improve convergence speed, CUBIC uses a heuristic. When a new flow joins the network, existing flows need to give up some of their bandwidth to allow the new flow some room for growth, if the existing flows have been using all the network bandwidth. To speed up this bandwidth release by existing flows, the following "Fast Convergence" mechanism **SHOULD** be implemented.

With Fast Convergence, when a congestion event occurs, we update W_{max} as follows, before the window reduction as described in [Section 4.6](#).

$$W_{max} = \begin{cases} cwnd * \frac{1+\beta_{cubic}}{2} & \text{if } cwnd < W_{max} \text{ and fast convergence is enabled,} \\ & \text{further reduce } W_{max} \\ cwnd & \text{otherwise, remember cwnd before reduction} \end{cases}$$

At a congestion event, if the current $cwnd$ is less than W_{max} , this indicates that the saturation point experienced by this flow is getting reduced because of a change in available bandwidth. Then we allow this flow to release more bandwidth by reducing W_{max} further. This action effectively lengthens the time for this flow to increase its congestion window, because the reduced W_{max} forces the flow to plateau earlier. This allows more time for the new flow to catch up to its congestion window size.

Fast Convergence is designed for network environments with multiple CUBIC flows. In network environments with only a single CUBIC flow and without any other traffic, Fast Convergence **SHOULD** be disabled.

4.8. Timeout

In case of a timeout, CUBIC follows Reno to reduce $cwnd$ [RFC5681], but sets $ssthresh$ using β_{cubic} (same as in [Section 4.6](#)) in a way that is different from Reno TCP [RFC5681].

During the first congestion avoidance stage after a timeout, CUBIC increases its congestion window size using [Figure 1](#), where t is the

elapsed time since the beginning of the current congestion avoidance, K is set to 0, and W_{max} is set to the congestion window size at the beginning of the current congestion avoidance stage. In addition, for the Reno-friendly region, W_{est} **SHOULD** be set to the congestion window size at the beginning of the current congestion avoidance.

4.9. Spurious Congestion Events

In cases where CUBIC reduces its congestion window in response to having detected packet loss via duplicate ACKs or timeouts, there is a possibility that the missing ACK would arrive after the congestion window reduction and a corresponding packet retransmission. For example, packet reordering could trigger this behavior. A high degree of packet reordering could cause multiple congestion window reduction events, where spurious losses are incorrectly interpreted as congestion signals, thus degrading CUBIC's performance significantly.

For TCP, there are two types of spurious events - spurious timeouts and spurious fast retransmits. In case of QUIC, there are no spurious timeouts as the loss is only detected after receiving an ACK.

4.9.1. Spurious timeout

An implementation **MAY** detect spurious timeouts based on the mechanisms described in Forward RTO-Recovery [[RFC5682](#)]. Experimental alternatives include Eifel [[RFC3522](#)]. When a spurious timeout is detected, a TCP implementation **MAY** follow the response algorithm described in [[RFC4015](#)] to restore the congestion control state and adapt the retransmission timer to avoid further spurious timeouts.

4.9.2. Spurious loss detected by acknowledgements

Upon receiving an ACK, a TCP implementation **MAY** detect spurious losses either using TCP Timestamps or via D-SACK [[RFC2883](#)]. Experimental alternatives include Eifel detection algorithm [[RFC3522](#)] which uses TCP Timestamps and DSACK based detection [[RFC3708](#)] which uses DSACK information. A QUIC implementation can easily determine a spurious loss if a QUIC packet is acknowledged after it has been marked as lost and the original data has been retransmitted with a new QUIC packet.

In this section, we specify a simple response algorithm when a spurious loss is detected by acknowledgements. Implementations would need to carefully evaluate the impact of using this algorithm in different environments that may experience sudden change in available capacity (e.g., due to variable radio capacity, a routing change, or a mobility event).

When a packet loss is detected via acknowledgements, a CUBIC implementation **MAY** save the current value of the following variables before the congestion window is reduced.

```

prior_cwnd = cwnd
prior_ssthresh = ssthresh
prior_Wmax = Wmax
prior_K = K
prior_epochstart = epochstart
prior_West = West

```

Once the previously declared packet loss is confirmed to be spurious, CUBIC **MAY** restore the original values of the above-mentioned variables as follows if the current *cwnd* is lower than *prior_cwnd*. Restoring the original values ensures that CUBIC's performance is similar to what it would be without spurious losses.

```

cwnd = prior_cwnd
ssthresh = prior_ssthresh
Wmax = prior_Wmax
K = prior_K
epochstart = prior_epochstart
West = prior_West

```

} if *cwnd* < *prior_cwnd*

In rare cases, when the detection happens long after a spurious loss event and the current *cwnd* is already higher than *prior_cwnd*, CUBIC **SHOULD** continue to use the current and the most recent values of these variables.

4.10. Slow Start

CUBIC **MUST** employ a slow-start algorithm, when *cwnd* is no more than *ssthresh*. In general, CUBIC **SHOULD** use the HyStart++ slow start algorithm [[I-D.ietf-tcpm-hystartplusplus](#)], or **MAY** use the Reno TCP slow start algorithm [[RFC5681](#)] in the rare cases when HyStart++ is not suitable. Experimental alternatives include hybrid slow start [[HR11](#)], a predecessor to HyStart++ that some CUBIC implementations have used as the default for the last decade, and limited slow start [[RFC3742](#)]. Whichever start-up algorithm is used, work might be needed to ensure that the end of slow start and the first multiplicative decrease of congestion avoidance work well together.

When CUBIC uses HyStart++ [[I-D.ietf-tcpm-hystartplusplus](#)], it may exit the first slow start without incurring any packet loss and thus *W_{max}* is undefined. In this special case, CUBIC switches to congestion avoidance and increases its congestion window size using [Figure 1](#), where *t* is the elapsed time since the beginning of the current congestion avoidance, *K* is set to 0, and *W_{max}* is set to the

congestion window size at the beginning of the current congestion avoidance stage.

5. Discussion

In this section, we further discuss the safety features of CUBIC following the guidelines specified in [\[RFC5033\]](#).

With a deterministic loss model where the number of packets between two successive packet losses is always $1/p$, CUBIC always operates with the concave window profile, which greatly simplifies the performance analysis of CUBIC. The average window size of CUBIC can be obtained by the following function:

$$AVG_W_{cubic} = \sqrt[4]{\frac{C * (3 + \beta_{cubic})}{4 * (1 - \beta_{cubic})}} * \frac{\sqrt[4]{RTT^3}}{\sqrt[4]{p^3}}$$

Figure 6

With β_{cubic} set to 0.7, the above formula reduces to:

$$AVG_W_{cubic} = \sqrt[4]{\frac{C * 3.7}{1.2}} * \frac{\sqrt[4]{RTT^3}}{\sqrt[4]{p^3}}$$

Figure 7

We will determine the value of C in the following subsection using [Figure 7](#).

5.1. Fairness to Reno

In environments where Reno is able to make reasonable use of the available bandwidth, CUBIC does not significantly change this state.

Reno performs well in the following two types of networks:

1. networks with a small bandwidth-delay product (BDP)
2. networks with a short RTTs, but not necessarily a small BDP

CUBIC is designed to behave very similarly to Reno in the above two types of networks. The following two tables show the average window sizes of Reno TCP, HSTCP, and CUBIC TCP. The average window sizes of Reno TCP and HSTCP are from [\[RFC3649\]](#). The average window size of CUBIC is calculated using [Figure 7](#) and the CUBIC Reno-friendly region for three different values of C .

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	59
1.0e-04	120	263	120	187	333
1.0e-05	379	1795	593	1054	1874
1.0e-06	1200	12280	3332	5926	10538
1.0e-07	3795	83981	18740	33325	59261
1.0e-08	12000	574356	105383	187400	333250

Table 1: Reno TCP, HSTCP, and CUBIC with $RTT = 0.1$ seconds

[Table 1](#) describes the response function of Reno TCP, HSTCP, and CUBIC in networks with $RTT = 0.1$ seconds. The average window size is in MSS-sized segments.

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	38
1.0e-04	120	263	120	120	120
1.0e-05	379	1795	379	379	379
1.0e-06	1200	12280	1200	1200	1874
1.0e-07	3795	83981	3795	5926	10538
1.0e-08	12000	574356	18740	33325	59261

Table 2: Reno TCP, HSTCP, and CUBIC with $RTT = 0.01$ seconds

[Table 2](#) describes the response function of Reno TCP, HSTCP, and CUBIC in networks with $RTT = 0.01$ seconds. The average window size is in MSS-sized segments.

Both tables show that CUBIC with any of these three C values is more friendly to Reno TCP than HSTCP, especially in networks with a short RTT where Reno TCP performs reasonably well. For example, in a network with $RTT = 0.01$ seconds and $p=10^{-6}$, Reno TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then Reno TCP can achieve an average rate of 1.44 Gbps. In this case, CUBIC with $C=0.04$ or $C=0.4$ achieves exactly the same rate as Reno TCP, whereas HSTCP is about ten times more aggressive than Reno TCP.

We can see that C determines the aggressiveness of CUBIC in competing with other congestion control algorithms for bandwidth. CUBIC is more friendly to Reno TCP, if the value of C is lower. However, we do not recommend setting C to a very low value like 0.04, since CUBIC with a low C cannot efficiently use the bandwidth in fast and long-distance networks. Based on these observations and extensive deployment experience, we find $C=0.4$ gives a good balance between Reno-friendliness and aggressiveness of window increase.

Therefore, C **SHOULD** be set to 0.4. With C set to 0.4, [Figure 7](#) is reduced to:

$$AVG_{W_{cubic}} = 1.054 * \frac{\sqrt[4]{RTT^3}}{\sqrt[4]{p^3}}$$

Figure 8

[Figure 8](#) is then used in the next subsection to show the scalability of CUBIC.

5.2. Using Spare Capacity

CUBIC uses a more aggressive window increase function than Reno for fast and long-distance networks.

The following table shows that to achieve the 10 Gbps rate, Reno TCP requires a packet loss rate of $2.0e-10$, while CUBIC TCP requires a packet loss rate of $2.9e-8$.

Throughput (Mbps)	Average W	Reno P	HSTCP P	CUBIC P
1	8.3	$2.0e-2$	$2.0e-2$	$2.0e-2$
10	83.3	$2.0e-4$	$3.9e-4$	$2.9e-4$
100	833.3	$2.0e-6$	$2.5e-5$	$1.4e-5$
1000	8333.3	$2.0e-8$	$1.5e-6$	$6.3e-7$
10000	83333.3	$2.0e-10$	$1.0e-7$	$2.9e-8$

Table 3: Required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput

[Table 3](#) describes the required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput. We use 1500-byte packets and an RTT of 0.1 seconds.

Our test results in [\[HLRX07\]](#) indicate that CUBIC uses the spare bandwidth left unused by existing Reno TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

5.3. Difficult Environments

CUBIC is designed to remedy the poor performance of Reno in fast and long-distance networks.

5.4. Investigating a Range of Environments

CUBIC has been extensively studied using simulations, testbed emulations, Internet experiments, and Internet measurements, covering a wide range of network environments [\[HLRX07\]](#)[\[H16\]](#)[\[CEHRX09\]](#)

[[HR11](#)][[BSCLU13](#)][[LBEWK16](#)]. They have convincingly demonstrated that CUBIC delivers substantial benefits over classical Reno congestion control [[RFC5681](#)].

Same as Reno, CUBIC is a loss-based congestion control algorithm. Because CUBIC is designed to be more aggressive (due to a faster window increase function and bigger multiplicative decrease factor) than Reno in fast and long-distance networks, it can fill large drop-tail buffers more quickly than Reno and increases the risk of a standing queue [[RFC8511](#)]. In this case, proper queue sizing and management [[RFC7567](#)] could be used to mitigate the risk to some extent and reduce the packet queuing delay. Also, in large-BDP networks after a congestion event, CUBIC, due its cubic window increase function, recovers quickly to the highest link utilization point. This means that link utilization is less sensitive to an active queue management (AQM) target that is lower than the amplitude of the whole sawtooth.

Similar to Reno, the performance of CUBIC as a loss-based congestion control algorithm suffers in networks where a packet loss is not a good indication of bandwidth utilization, such as wireless or mobile networks [[LIU16](#)].

5.5. Protection against Congestion Collapse

With regard to the potential of causing congestion collapse, CUBIC behaves like Reno, since CUBIC modifies only the window adjustment algorithm of Reno. Thus, it does not modify the ACK clocking and timeout behaviors of Reno.

CUBIC also satisfies the "full backoff" requirement as described in [[RFC5033](#)]. After reducing the sending rate to one packet per RTT in response to congestion events due to ECN-Echo ACKs, CUBIC then exponentially increases the transmission timer for each packet retransmission while congestion persists.

5.6. Fairness within the Alternative Congestion Control Algorithm

CUBIC ensures convergence of competing CUBIC flows with the same RTT in the same bottleneck links to an equal throughput. When competing flows have different RTT values, their throughput ratio is linearly proportional to the inverse of their RTT ratios. This is true independently of the level of statistical multiplexing on the link. The convergence time depends on the network environments (e.g., bandwidth, RTT) and the level of statistical multiplexing, as mentioned in [Section 3.4](#).

5.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC design.

5.8. Behavior for Application-Limited Flows

A flow is application-limited if it is currently sending less than what is allowed by the congestion window. This can happen if the flow is limited by either the sender application or the receiver application (via the receiver advertised window) and thus sends less data than what is allowed by the sender's congestion window.

CUBIC does not increase its congestion window if a flow is application-limited. [Section 4.2](#) requires that t in [Figure 1](#) does not include application-limited periods, such as idle periods, otherwise $W_{\text{cubic}}(t)$ might be very high after restarting from these periods.

5.9. Responses to Sudden or Transient Events

If there is a sudden increase in capacity, e.g., due to variable radio capacity, a routing change, or a mobility event, CUBIC is designed to utilize the newly available capacity faster than Reno.

On the other hand, if there is a sudden decrease in capacity, CUBIC reduces more slowly than Reno. This remains true whether or not CUBIC is in Reno-friendly mode and whether or not fast convergence is enabled.

5.10. Incremental Deployment

CUBIC requires only changes to the congestion control at the sender, and it does not require any changes at receivers. That is, a CUBIC sender works correctly with Reno receivers. In addition, CUBIC does not require any changes to routers and does not require any assistance from routers.

6. Security Considerations

CUBIC makes no changes to the underlying security of TCP. More information about TCP security concerns can be found in [\[RFC5681\]](#).

7. IANA Considerations

This document does not require any IANA actions.

8. References

8.1. Normative References

[I-D.ietf-tcpm-hystartplusplus] Balasubramanian, P., Huang, Y., and M. Olson, "HyStart++: Modified Slow Start for TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-hystartplusplus-04, 23 January 2022, <<https://>

datatracker.ietf.org/doc/html/draft-ietf-tcpm-hystartplusplus-04>.

- [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>>.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, <<https://www.rfc-editor.org/rfc/rfc2883>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/rfc/rfc2914>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/rfc/rfc3168>>.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, DOI 10.17487/RFC4015, February 2005, <<https://www.rfc-editor.org/rfc/rfc4015>>.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/rfc/rfc5033>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/rfc/rfc5348>>.
- [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>>.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, DOI 10.17487/RFC5682, September 2009, <<https://www.rfc-editor.org/rfc/rfc5682>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/rfc/rfc6298>>.

[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>>.

[RFC6675]

Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <<https://www.rfc-editor.org/rfc/rfc6675>>.

[RFC7567]

Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/rfc/rfc7567>>.

[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>>.

[RFC8985]

Cheng, Y., Cardwell, N., Dukkupati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <<https://www.rfc-editor.org/rfc/rfc8985>>.

[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>>.

8.2. Informative References

[BSCLU13]

Belhareth, S., Sassatelli, L., Collange, D., Lopez-Pacheco, D., and G. Urvoy-Keller, "Understanding TCP cubic performance in the cloud: A mean-field approach", 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), DOI 10.1109/cloudnet.2013.6710576, November 2013, <<https://doi.org/10.1109/cloudnet.2013.6710576>>.

[CEHRX09]

Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, "Stochastic convex ordering for multiplicative decrease internet congestion control", Computer Networks Vol. 53, pp. 365-381, DOI 10.1016/j.comnet.2008.10.012, February 2009, <<https://doi.org/10.1016/j.comnet.2008.10.012>>.

[FHP00]

Floyd, S., Handley, M., and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control", May 2000, <<https://www.icir.org/tfrc/aimd.pdf>>.

[GV02]

Gorinsky, S. and H. Vin, "Extended Analysis of Binary Adjustment Algorithms", Technical Report TR2002-29, Department of Computer Sciences, The University of Texas at Austin, 11 August 2002, <<https://www.cs.utexas.edu/ftp/techreports/tr02-39.ps.gz>>.

[H16]

Sangtae Ha, "Simulation, Testbed, and Deployment Testing Results of CUBIC", 3 November 2016, <https://web.archive.org/web/20161118125842/http://netsrv.csc.ncsu.edu/wiki/index.php/TCP_Testing>.

[HLRX07]

Ha, S., Le, L., Rhee, I., and L. Xu, "Impact of background traffic on performance of high-speed TCP variant protocols", Computer Networks Vol. 51, pp. 1748-1762, DOI 10.1016/j.comnet.2006.11.005, May 2007, <<https://doi.org/10.1016/j.comnet.2006.11.005>>.

[HR11]

Ha, S. and I. Rhee, "Taming the elephants: New TCP slow start", Computer Networks Vol. 55, pp. 2092-2110, DOI 10.1016/j.comnet.2011.01.014, June 2011, <<https://doi.org/10.1016/j.comnet.2011.01.014>>.

[HRX08]

Ha, S., Rhee, I., and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant", ACM SIGOPS Operating Systems Review Vol. 42, pp. 64-74, DOI 10.1145/1400097.1400105, July 2008, <<https://doi.org/10.1145/1400097.1400105>>.

[K03]

Kelly, T., "Scalable TCP: improving performance in highspeed wide area networks", ACM SIGCOMM Computer Communication Review Vol. 33, pp. 83-91, DOI 10.1145/956981.956989, April 2003, <<https://doi.org/10.1145/956981.956989>>.

[LBEWK16]

Lukaseder, T., Bradatsch, L., Erb, B., Van Der Heijden, R., and F. Kargl, "A Comparison of TCP Congestion Control Algorithms in 10G Networks", 2016 IEEE 41st Conference on Local Computer Networks (LCN), DOI 10.1109/lcn.2016.121, November 2016, <<https://doi.org/10.1109/lcn.2016.121>>.

[LIU16]

Liu, K. and J. Lee, "On Improving TCP Performance over Mobile Data Networks", IEEE Transactions on Mobile Computing Vol. 15, pp. 2522-2536, DOI 10.1109/tmc.2015.2500227, October 2016, <<https://doi.org/10.1109/tmc.2015.2500227>>.

[RFC3465]

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

[RFC3522]

Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, DOI 10.17487/RFC3522, April 2003, <<https://www.rfc-editor.org/rfc/rfc3522>>.

[RFC3649]

Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <<https://www.rfc-editor.org/rfc/rfc3649>>.

[RFC3708]

Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, DOI 10.17487/RFC3708, February 2004, <<https://www.rfc-editor.org/rfc/rfc3708>>.

[RFC3742]

Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <<https://www.rfc-editor.org/rfc/rfc3742>>.

[RFC4960]

Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/rfc/rfc4960>>.

[RFC6937]

Mathis, M., Dukkkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <<https://www.rfc-editor.org/rfc/rfc6937>>.

[RFC7661]

Fairhurst, G., Sathiaselalan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <<https://www.rfc-editor.org/rfc/rfc7661>>.

[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/rfc/rfc8312>>.

[RFC8511]

Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/rfc/rfc8511>>.

[RFC9000]

Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/rfc/rfc9000>>.

[SXEZ19]

Sun, W., Xu, L., Elbaum, S., and D. Zhao, "Model-Agnostic and Efficient Exploration of Numerical Congestion Control

State Space of Real-World TCP Implementations", IEEE/ACM Transactions on Networking Vol. 29, pp. 1990-2004, DOI 10.1109/tnet.2021.3078161, October 2021, <<https://doi.org/10.1109/tnet.2021.3078161>>.

[XHR04] Xu, L., Harfoush, K., and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks", IEEE INFOCOM 2004, DOI 10.1109/infcom.2004.1354672, n.d., <<https://doi.org/10.1109/infcom.2004.1354672>>.

Appendix A. Acknowledgments

Richard Scheffenegger and Alexander Zimmermann originally co-authored [[RFC8312](#)].

These individuals suggested improvements to this document:

*Bob Briscoe

*Christian Huitema

*Gorry Fairhurst

*Jonathan Morton

*Juhamatti Kuusisaari

*Junho Choi

*Markku Kojo

*Martin Thomson

*Matt Mathis

*Matt Olson

*Michael Welzl

*Mirja Kuehlewind

*Mohit P. Tahiliani

*Neal Cardwell

*Praveen Balasubramanian

*Randall Stewart

*Richard Scheffenegger

*Rod Grimes

*Tom Henderson

*Tom Petch

*Wesley Rosenblum

*Yoshifumi Nishida

*Yuchung Cheng

Appendix B. Evolution of CUBIC

B.1. Since draft-ietf-tcpm-rfc8312bis-07

*Document the WG discussion and decision around [[RFC5033](#)] and [[RFC2914](#)] ([#145](#))

B.2. Since draft-ietf-tcpm-rfc8312bis-06

*RFC7661 is safe even when cwnd grows beyond rwnd ([#143](#))

B.3. Since draft-ietf-tcpm-rfc8312bis-05

*Clarify meaning of "application-limited" in Section 5.8 ([#137](#))

*Create new subsections for spurious timeouts and spurious loss via ACK ([#90](#))

*Brief discussion of convergence in Section 5.6 ([#96](#))

*Add more test results to Section 5 and update some references ([#91](#))

*Change wording around setting ssthresh ([#131](#))

B.4. Since draft-ietf-tcpm-rfc8312bis-04

*Fix incorrect math ([#106](#))

*Update RFC5681 ([#99](#))

*Rephrase text around algorithmic alternatives, add HyStart++ ([#85](#), [#86](#), [#90](#))

*Clarify what we mean by "new ACK" and use it in the text in more places. ([#101](#))

*Rewrite the Responses to Sudden or Transient Events section ([#98](#))

- *Remove confusing text about $cwnd_{start}$ in Section 4.2 ([#100](#))
- *Change terminology from "AIMD" to "Reno" ([#108](#))
- *Moved **MUST NOT** from app-limited section to main cubic AI section ([#97](#))
- *Clarify cwnd decrease during multiplicative decrease ([#102](#))
- *Clarify text around queuing and slow adaptation of CUBIC in wireless environments ([#94](#))
- *Set lower bound of cwnd to 1 MSS and use retransmit timer thereafter ([#83](#))
- *Use FlightSize instead of cwnd to update ssthresh ([#114](#))

B.5. Since draft-ietf-tcpm-rfc8312bis-03

- *Remove reference from abstract ([#82](#))

B.6. Since draft-ietf-tcpm-rfc8312bis-02

- *Description of packet loss rate p ([#65](#))
- *Clarification of TCP Friendly Equation for ABC and Delayed ACK ([#66](#))
- *add applicability to QUIC and SCTP ([#61](#))
- *clarity on setting α_{aimd} to 1 ([#68](#))
- *introduce α_{cubic} ([#64](#))
- *clarify cwnd growth in convex region ([#69](#))
- *add guidance for using bytes and mention that segments count is decimal ([#67](#))
- *add loss events detected by RACK and QUIC loss detection ([#62](#))

B.7. Since draft-ietf-tcpm-rfc8312bis-01

- *address Michael Scharf's editorial suggestions. ([#59](#))
- *add "Note to the RFC Editor" about removing underscores

B.8. Since draft-ietf-tcpm-rfc8312bis-00

- *use updated xml2rfc with better text rendering of subscripts

B.9. Since draft-eggert-tcpm-rfc8312bis-03

- *fix spelling nits
- *rename to draft-ietf
- *define W_{max} more clearly

B.10. Since draft-eggert-tcpm-rfc8312bis-02

- *add definition for `segments_acked` and α_{aimd} . ([#47](#))
- *fix a mistake in W_{max} calculation in the fast convergence section. ([#51](#))
- *clarity on setting *ssthresh* and $cwnd_{start}$ during multiplicative decrease. ([#53](#))

B.11. Since draft-eggert-tcpm-rfc8312bis-01

- *rename TCP-Friendly to AIMD-Friendly and rename Standard TCP to AIMD TCP to avoid confusion as CUBIC has been widely used on the Internet. ([#38](#))
- *change introductory text to reflect the significant broader deployment of CUBIC on the Internet. ([#39](#))
- *rephrase introduction to avoid referring to variables that have not been defined yet.

B.12. Since draft-eggert-tcpm-rfc8312bis-00

- *acknowledge former co-authors ([#15](#))
- *prevent *cwnd* from becoming less than two ([#7](#))
- *add list of variables and constants ([#5](#), [#6](#))
- *update *K*'s definition and add bounds for CUBIC *target cwnd* [[SXEZ19](#)] ([#1](#), [#14](#))
- *update W_{est} to use AIMD approach ([#20](#))
- *set α_{aimd} to 1 once W_{est} reaches W_{max} ([#2](#))
- *add Vidhi as co-author ([#17](#))
- *note for Fast Recovery during *cwnd* decrease due to congestion event ([#11](#))
- *add section for spurious congestion events ([#23](#))

*initialize W_{est} after timeout and remove variable W_{last_max} ([#28](#))

B.13. Since RFC8312

*converted to Markdown and xml2rfc v3

*updated references (as part of the conversion)

*updated author information

*various formatting changes

*move to Standards Track

B.14. Since the Original Paper

CUBIC has gone through a few changes since the initial release [[HRX08](#)] of its algorithm and implementation. Below we highlight the differences between its original paper and [[RFC8312](#)].

*The original paper [[HRX08](#)] includes the pseudocode of CUBIC implementation using Linux's pluggable congestion control framework, which excludes system-specific optimizations. The simplified pseudocode might be a good source to start with and understand CUBIC.

*[[HRX08](#)] also includes experimental results showing its performance and fairness.

*The definition of β_{cubic} constant was changed in [[RFC8312](#)]. For example, β_{cubic} in the original paper was the window decrease constant while [[RFC8312](#)] changed it to CUBIC multiplication decrease factor. With this change, the current congestion window size after a congestion event in [[RFC8312](#)] was $\beta_{cubic} * W_{max}$ while it was $(1 - \beta_{cubic}) * W_{max}$ in the original paper.

*Its pseudocode used W_{last_max} while [[RFC8312](#)] used W_{max} .

*Its AIMD-friendly window was W_{tcp} while [[RFC8312](#)] used W_{est} .

Authors' Addresses

Lisong Xu
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588-0115
United States of America

Email: xu@unl.edu

URI: <https://cse.unl.edu/~xu/>

Sangtae Ha
University of Colorado at Boulder
Department of Computer Science
Boulder, CO 80309-0430
United States of America

Email: sangtae.ha@colorado.edu
URI: <https://netstech.org/sangtaeha/>

Injong Rhee
Bowery Farming
151 W 26TH Street, 12TH Floor
New York, NY 10001
United States of America

Email: injongrhee@gmail.com

Vidhi Goel
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: vidhi_goel@apple.com

Lars Eggert (editor)
NetApp
Stenbergintie 12 B
FI-02700 Kauniainen
Finland

Email: lars@eggert.org
URI: <https://eggert.org/>