Workgroup: TCPM
Internet-Draft:
draft-eggert-tcpm-rfc8312bis-03
Obsoletes: 8312 (if approved)
Published: 10 March 2021
Intended Status: Standards Track
Expires: 11 September 2021
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 an extension to the traditional TCP standards. It differs from the traditional TCP standards only in the congestion control algorithm on the sender side. In particular, it uses a cubic function instead of the linear window increase function of the traditional TCP standards to improve scalability and stability under 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 [RFC8312].

Note to Readers

Discussion of this draft takes place on the TCPM working group mailing list, which is archived at https://mailarchive.ietf.org/arch/browse/tcpm/.

Working Group information can be found at https://datatracker.ietf.org/wg/tcpm/; source code and issues list for this draft can be found at https://github.com/NTAP/rfc8312bis.

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 11 September 2021.

Copyright Notice

Table of Contents

1. Introduction

The low utilization problem of traditional TCP in fast and long-distance networks is well documented in [K03] and [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). [HKLRX06] 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 TCP standards and their variants, including TCP-Reno [RFC5681], TCP-NewReno [RFC6582][RFC6675], SCTP [RFC4960], and TFRC [RFC5348], which use the same linear increase function for window growth. We refer to all Reno-style TCP standards and their variants collectively as "AIMD TCP" below because they use the Additive Increase and Multiplicative Decrease algorithm (AIMD).

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of traditional AIMD TCP to remedy this problem. This document describes the most recent specification of CUBIC. Specifically, CUBIC uses a cubic function instead of the linear window increase function of AIMD TCP to improve scalability and stability under fast and long-distance networks.

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 AIMD TCP in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and round-trip time (RTT) fairness. 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 traditional AIMD congestion control. It is therefore to be regarded as the current standard for TCP congestion control.

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 AIMD-friendly, CUBIC is designed to behave like AIMD TCP in networks with short RTTs and small bandwidth where AIMD TCP 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 AIMD TCP 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 is detected by duplicate ACKs or Explicit Congestion Notification-Echo (ECN-Echo, ECE) ACKs [RFC3168], CUBIC remembers the congestion window size where 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 [CEHRX07]. 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 a large number of packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

3.2. Principle 2 for AIMD Friendliness

CUBIC promotes per-flow fairness to AIMD TCP. Note that AIMD TCP 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 AIMD TCP 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 AIMD TCP, CUBIC simply follows the window size of AIMD TCP to ensure that CUBIC achieves at least the same throughput as AIMD TCP in small-BDP networks. We call this region where CUBIC behaves like AIMD TCP the "AIMD-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 of the AIMD-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the AIMD-friendly region.

This notion of a linear throughput ratio is similar to that of AIMD TCP under high statistical multiplexing where packet loss is independent of individual flow rates. However, under low statistical multiplexing, the throughput ratio of AIMD TCP 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 AIMD TCP. 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 AIMD TCP 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 AIMD TCP 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.

4.1.1. Constants of Interest

β_{cubic}: CUBIC multiplication decrease factor as described in Section 4.6.

α_{aimd}: CUBIC additive increase factor used in AIMD-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

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

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.

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 AIMD-friendly region, that is, an estimate for the congestion window of AIMD TCP.

segments_acked: Number of segments acked when an ACK is received.

4.2. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of AIMD TCP by increasing the congestion window only at the reception of an 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 where a packet loss is detected by duplicate ACKs or by receiving packets carrying ECE flags [RFC3168], CUBIC changes the window increase function of AIMD TCP.

CUBIC uses the following window increase function:

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

Figure 1

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

$$t = current\_time - epoch_{start}$$

and where $epoch_{start}$ 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:

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

Figure 2

where $cwnd_{start}$ is the congestion window at the beginning of the current congestion avoidance stage. For example, right after a congestion event, $cwnd_{start}$ is equal to the new cwnd calculated as described in Section 4.6.

Upon receiving an 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.

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

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

1. The AIMD-friendly region, which ensures that CUBIC achieves at least the same throughput as AIMD TCP.

2. The concave region, if CUBIC is not in the AIMD-friendly region and *cwnd* is less than W_{max}.

3. The convex region, if CUBIC is not in the AIMD-friendly region and *cwnd* is greater than W_{max}.

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

4.3. AIMD-Friendly Region

AIMD TCP 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 AIMD-friendly region to achieve at least the same throughput as AIMD TCP.

The AIMD-friendly region is designed according to the analysis in [FHP00], which studies the performance of an AIMD algorithm with an additive factor of α_{aimd} (segments per *RTT*) and a multiplicative factor of β_{aimd}, denoted by AIMD(α_{aimd}, β_{aimd}). Specifically, the average congestion window size of AIMD(α_{aimd}, β_{aimd}) can be calculated using Figure 3. The analysis shows that AIMD(α_{aimd}, β_{aimd}) with

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

achieves the same average window size as AIMD TCP that uses AIMD(1, 0.5).

$$\text{AVG\_AIMD}(\alpha_{aimd}, \beta_{aimd}) = \sqrt{\frac{\alpha_{aimd} * (1 + \beta_{aimd})}{2 * (1 - \beta_{aimd}) * p}}$$

Figure 3

Based on the above analysis, CUBIC uses Figure 4 to estimate the window size W_{est} of AIMD(α_{aimd}, β_{aimd}) with

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

which achieves the same average window size as AIMD TCP. When receiving an 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 AIMD-friendly region and *cwnd* SHOULD be set to W_{est} at each reception of an ACK.

W_{est} is set equal to $cwnd_{start}$ at the start of the congestion avoidance stage. After that, on every ACK, W_{est} is updated using Figure 4.

$$W_{est} = W_{est} + \alpha_{aimd} * \frac{segments\_acked}{cwnd}$$

Figure 4

Note that once W_{est} reaches W_{max}, that is, W_{est} >= W_{max}, α_{aimd} SHOULD be set to 1 to achieve the same congestion window increment as AIMD TCP, which uses AIMD(1, 0.5).

4.4. Concave Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the AIMD-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 ACK, where *target* is calculated as described in Section 4.2.

4.5. Convex Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the AIMD-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.

In this region, CUBIC is very careful. The convex profile ensures that the window increases very slowly at the beginning and gradually increases its increase rate. 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 ACK, where *target* is calculated as described in Section 4.2.

4.6. Multiplicative Decrease

When a packet loss is detected by duplicate ACKs or by receiving packets carrying ECE flags, CUBIC updates W_{max} and reduces *cwnd* and *ssthresh* immediately as described below. An implementation MAY set a smaller *ssthresh* than suggested below to accomodate rate-limited applications as described in [RFC7661]. For both packet loss and congestion detection through ECN, the sender MAY employ a Fast Recovery algorithm to gradually adjust the congestion window to its new reduced *ssthresh* value. The parameter β_{cubic} SHOULD be set to 0.7.

$$\begin{aligned} ssthresh &= cwnd * \beta_{cubic} && \text{// new slow-start threshold} \\ ssthresh &= \max(ssthresh, 2) && \text{// threshold is at least 2 MSS} \\ cwnd &= ssthresh && \text{// window reduction} \end{aligned}$$

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.

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 AIMD TCP to reduce *cwnd* [RFC5681], but sets *ssthresh* using β_{cubic} (same as in Section 4.6) in a way that is different from AIMD 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 AIMD-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.

When there is a congestion event, a CUBIC implementation SHOULD save the current value of the following variables before the congestion window reduction.

$$
\begin{aligned}
prior\_cwnd &= cwnd \\
prior\_ssthresh &= ssthresh \\
prior\_W_{max} &= W_{max} \\
prior\_K &= K \\
prior\_epoch_{start} &= epoch_{start} \\
prior\_W\_est &= W_{est}
\end{aligned}
$$

CUBIC MAY implement an algorithm to detect spurious retransmissions, such as DSACK [RFC3708], Forward RTO-Recovery [RFC5682] or Eifel [RFC3522]. Once a spurious congestion event is detected, CUBIC SHOULD restore the original values of 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.

$$
\left.
\begin{aligned}
cwnd &= prior\_cwnd \\
ssthresh &= prior\_ssthresh \\
W_{max} &= prior\_W_{max} \\
K &= prior\_K \\
epoch_{start} &= prior\_epoch_{start} \\
W_{est} &= prior\_W_{est}
\end{aligned}
\right\} \text{if } cwnd < prior\_cwnd
$$

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*. Among the slow-start algorithms, CUBIC MAY choose the AIMD TCP slow start [RFC5681] in general networks, or the limited slow start [RFC3742] or hybrid slow start [HR08] for fast and long-distance networks.

When CUBIC uses hybrid slow start [HR08], 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:

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

Figure 5

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

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

Figure 6

We will determine the value of *C* in the following subsection using Figure 6.

5.1. Fairness to AIMD TCP

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

AIMD TCP 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 AIMD TCP in the above two types of networks. The following two tables show the average window sizes of AIMD TCP, HSTCP, and CUBIC. The average window sizes of AIMD TCP and HSTCP are from [RFC3649]. The average window size of CUBIC is calculated using Figure 6 and the CUBIC AIMD-friendly region for three different values of *C*.

Loss Rate P	AIMD	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

Loss Rate P	AIMD	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
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: AIMD TCP, HSTCP, and CUBIC with RTT = 0.1 seconds

Table 1 describes the response function of AIMD TCP, HSTCP, and CUBIC in networks with *RTT* = 0.1 seconds. The average window size is in MSS-sized segments.

Loss Rate P	AIMD	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: AIMD TCP, HSTCP, and CUBIC with RTT = 0.01 seconds

Table 2 describes the response function of AIMD 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 AIMD TCP than HSTCP, especially in networks with a short *RTT* where AIMD TCP performs reasonably well. For example, in a network with *RTT* = 0.01 seconds and p=10^-6, AIMD TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then AIMD 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 AIMD TCP, whereas HSTCP is about ten times more aggressive than AIMD 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 AIMD 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 AIMD- friendliness and aggressiveness of window increase. Therefore, *C* SHOULD be set to 0.4. With *C* set to 0.4, Figure 6 is reduced to:

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

Figure 7

Figure 7 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 AIMD TCP for fast and long-distance networks.

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

Throughput (Mbps)	Average W	AIMD 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 AIMD TCP, HSTCP, and CUBIC to achieve a certain throughput

Table 3 describes the required packet loss rate for AIMD 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 [HKLRX06] indicate that CUBIC uses the spare bandwidth left unused by existing AIMD 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 AIMD TCP in fast and long-distance networks.

5.4. Investigating a Range of Environments

CUBIC has been extensively studied by using both NS-2 simulation and testbed experiments, covering a wide range of network environments. More information can be found in [HKLRX06]. Additionally, there is decade-long deployment experience with CUBIC on the Internet.

Same as AIMD TCP, 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 AIMD TCP in fast and long-distance networks, it can fill large drop-tail buffers more quickly than AIMD TCP and increases the risk of a standing queue [RFC8511]. In this case, proper queue sizing and management [RFC7567] could be used to reduce the packet queuing delay.

5.5. Protection against Congestion Collapse

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

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.

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

CUBIC does not increase its congestion window size if a flow is currently limited by the application instead of the congestion window. In case of long periods during which *cwnd* has not been updated due to such an application limit, such as idle periods, t in Figure 1 MUST NOT include these periods; otherwise, $W_{cubic}(t)$ might be very high after restarting from these periods.

5.9. Responses to Sudden or Transient Events

If there is a sudden congestion, a routing change, or a mobility event, CUBIC behaves the same as AIMD TCP.

5.10. Incremental Deployment

CUBIC requires only changes to TCP senders, and it does not require any changes at TCP receivers. That is, a CUBIC sender works correctly with the AIMD TCP 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

[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>.

[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>.

[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>.

[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>.

8.2. Informative References

[CEHRX07] Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, "Stochastic Ordering for Internet Congestion Control and its Applications", IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications, DOI 10.1109/infcom.2007.111, 2007, <https://doi.org/10.1109/infcom.2007.111>.

[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, <http://www.cs.utexas.edu/ftp/techreports/tr02-39.ps.gz>.

[HKLRX06] Ha, S., Kim, Y., Le, L., Rhee, I., and L. Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP Variants", International Workshop on Protocols for Fast Long-Distance Networks, February 2006, <https://pfld.net/2006/paper/s2_03.pdf>.

[HR08] Ha, S. and I. Rhee, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", International Workshop on Protocols for Fast Long-Distance Networks, March 2008, <http://www.hep.man.ac.uk/g/GDARN-IT/pfldnet2008/paper/Sangate_Ha%20Final.pdf>.

[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>.

[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>.

[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>.

[RFC7661] Fairhurst, G., Sathiaseelan, 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>.

[SXEZ19] Sun, W., Xu, L., Elbaum, S., and D. Zhao, "Model-Agnostic and Efficient Exploration of Numerical State Space of Real-World TCP Congestion Control Implementations", USENIX NSDI 2019, February 2019, <https://www.usenix.org/system/files/nsdi19-sun.pdf>.

[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, March 2004, <https://doi.org/10.1109/infcom.2004.1354672>.

Appendix A. Acknowledgements

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

Appendix B. Evolution of CUBIC

B.1. Since draft-eggert-tcpm-rfc8312bis-02

*add definition for segments_acked and $alpha_{aimd}$. (#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.2. 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 in the Internet. (#38)

*change introductory text to reflect the significant broader deployment of CUBIC in the Internet. (#39)

*rephrase introduction to avoid referring to variables that have not been defined yet.

B.3. 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 $alpha_{aimd}$ 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.4. 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.5. 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 $beta_{cubic}$ constant was changed in [RFC8312]. For example, $beta_{cubic}$ 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/