TCP Maintenance and Minor Extensions (TCPM) WG
Internet-Draft
Intended status: Experimental
Expires: October 18, 2015

I. Rhee
NCSU
L. Xu
UNL
S. Ha
NCSU
A. Zimmermann
L. Eggert
R. Scheffenegger
NetApp
April 16, 2015

CUBIC for Fast Long-Distance Networks
draft-zimmermann-tcpm-cubic-01

Abstract

CUBIC is an extension to the current TCP standards. The protocol differs from the current TCP standards only in the congestion window adjustment function in the sender side. In particular, it uses a cubic function instead of a linear window increase of the current TCP standards to improve scalability and stability under fast and long distance networks. BIC-TCP, a predecessor of CUBIC, has been a default TCP adopted by Linux since year 2005 and has already been deployed globally and in use for several years by the Internet community at large. CUBIC is using a similar window growth function as BIC-TCP and is designed to be less aggressive and fairer to TCP in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability and RTT fairness. Through extensive testing in various Internet scenarios, we believe that CUBIC is safe for deployment and testing in the global Internet. The intent of this document is to provide the protocol specification of CUBIC for a third party implementation and solicit the community feedback through experimentation on the performance of CUBIC. We expect this document to be eventually published as an experimental RFC.

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 http://datatracker.ietf.org/drafts/current/.

This Internet-Draft will expire on October 18, 2015.

Table of Contents

1. Introduction 3
2. Conventions 5
3. CUBIC Congestion Control 5
3.1. Window growth function 5
3.2. TCP-friendly region 6
3.3. Concave region 6
3.4. Convex region 7
3.5. Multiplicative decrease 7
3.6. Fast convergence 7
4. Discussion 8
4.1. Fairness to standard TCP 8
4.2. Using Spare Capacity 10
4.3. Difficult Environments 10
4.4. Investigating a Range of Environments 11
4.5. Protection against Congestion Collapse 11
4.6. Fairness within the Alternative Congestion Control Algorithm. 11
4.7. Performance with Misbehaving Nodes and Outside Attackers 11
4.8. Responses to Sudden or Transient Events 11
4.9. Incremental Deployment 11
5. Security Considerations 11
6. IANA Considerations 11
7. Acknowledgements 12
8. References 12

8.1. Normative References 12
8.2. Informative References 12
Authors' Addresses 13

1. Introduction

The low utilization problem of TCP in fast long-distance networks is well documented in [K03][RFC3649]. This problem arises from a slow increase of congestion window following a congestion event in a network with a large bandwidth delay product (BDP). Our experience [HKLRX06] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets (each packet is sized around 1000 bytes) especially under a network path with over 100ms round-trip times (RTTs). This problem is equally applicable to all Reno style TCP standards and their variants, including TCP-RENO [RFC5681], TCP-NewReno [RFC6582], TCP-SACK [RFC2018], SCTP [RFC4960], TFRC [RFC5348] that use the same linear increase function for window growth, which we refer to collectively as Standard TCP below.

CUBIC [HRX08] is a modification to the congestion control mechanism of Standard TCP, in particular, to the window increase function of Standard TCP senders, to remedy this problem. It uses a cubic increase function in terms of the elapsed time from the last congestion event. While most alternative algorithms to Standard TCP uses a convex increase function where after a loss event, the window increment is always increasing, CUBIC uses both the concave and convex profiles of a cubic function for window increase. After a window reduction following a loss event, it registers the window size where it got the loss event as W_max and performs a multiplicative decrease of congestion window and the regular fast recovery and retransmit of Standard TCP. After it enters into congestion avoidance from fast recovery, it starts to increase the window using the concave profile of the cubic function. The cubic function is set to have its plateau at W_max so the concave growth continues until the window size becomes W_max. After that, the cubic function turns into a convex profile and the convex window growth begins. This style of window adjustment (concave and then convex) improves protocol and network stability while maintaining high network utilization [CEHRX07]. This is because the window size remains almost constant, forming a plateau around W_max where network utilization is deemed highest and under steady state, most window size samples of CUBIC are close to W_max, thus promoting high network utilization and protocol stability. Note that protocols with convex increase functions have the maximum increments around W_max and introduces a large number of packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

Another notable feature of CUBIC is that its window increase rate is mostly independent of RTT, and follows a (cubic) function of the elapsed time since the last loss event. This feature promotes per-flow fairness to Standard TCP as well as RTT-fairness. Note that Standard TCP performs well under short RTT and small bandwidth (or small BDP) networks. Only in a large long RTT and large bandwidth (or large BDP) networks, it has the scalability problem. An alternative protocol to Standard TCP designed to be friendly to Standard TCP at a per-flow basis must operate must increase its window much less aggressively in small BDP networks than in large BDP networks. In CUBIC, its window growth rate is slowest around the inflection point of the cubic function and this function does not depend on RTT. In a smaller BDP network where Standard TCP flows are working well, the absolute amount of the window decrease at a loss event is always smaller because of the multiplicative decrease. Therefore, in CUBIC, the starting window size after a loss event from which the window starts to increase, is smaller in a smaller BDP network, thus falling nearer to the plateau of the cubic function where the growth rate is slowest. By setting appropriate values of the cubic function parameters, CUBIC sets its growth rate always no faster than Standard TCP around its inflection point. When the cubic function grows slower than the window of Standard TCP, CUBIC simply follows the window size of Standard TCP to ensure fairness to Standard TCP in a small BDP network. We call this region where CUBIC behaves like Standard TCP, the TCP-friendly region.

CUBIC maintains the same window growth rate independent of RTTs outside of the TCP-friendly region, and flows with different RTTs have the similar window sizes under steady state when they operate outside the TCP-friendly region. This ensures CUBIC flows with different RTTs to have their bandwidth shares linearly proportional to the inverse of their RTT ratio (the longer RTT, the smaller the share). This behavior is the same as that of Standard TCP under high statistical multiplexing environments where packet losses are independent of individual flow rates. However, under low statistical multiplexing environments, the bandwidth share ratio of Standard TCP flows with different RTTs is squarely proportional to the inverse of their RTT ratio [XHR04]. CUBIC always ensures the linear ratio independent of the levels of statistical multiplexing. This is an improvement over Standard TCP. While there is no consensus on a particular bandwidth share ratios of different RTT flows, we believe that under wired Internet, use of the linear share notion seems more reasonable than equal share or a higher order shares. HTCP [LS08] currently uses the equal share.

CUBIC sets the multiplicative window decrease factor to 0.2 while Standard TCP uses 0.5. While this improves the scalability of the protocol, a side effect of this decision is slower convergence

especially under low statistical multiplexing environments. This design choice is following the observation that the author of HSTCP [RFC3649] has made along with other researchers (e.g., [GV02]): the current Internet becomes more asynchronous with less frequent loss synchronizations with high statistical multiplexing. Under this environment, even strict MIMD can converge. CUBIC flows with the same RTT always converge to the same share of bandwidth independent of statistical multiplexing, thus achieving intra-protocol fairness. We also find that under the environments with sufficient statistical multiplexing, the convergence speed of CUBIC flows is reasonable.

In the ensuing sections, we provide the exact specification of CUBIC and 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. CUBIC Congestion Control

3.1. Window growth function

CUBIC maintains the acknowledgment (ACK) clocking of Standard TCP by increasing congestion window only at the reception of ACK. The protocol does not make any change to the fast recovery and retransmit of TCP-NewReno [RFC6582] and TCP-SACK [RFC2018]. During congestion avoidance after fast recovery, CUBIC changes the window update algorithm of Standard TCP. Suppose that W_max is the window size before the window is reduced in the last fast retransmit and recovery.

The window growth function of CUBIC uses the following function:

$$W(t) = C*(t-K)^3 + W_{max} \quad \text{(Eq. 1)}$$

where C is a constant fixed to determine the aggressiveness of window growth in high BDP networks, t is the elapsed time from the last window reduction,and K is the time period that the above function takes to increase W to W_max when there is no further loss event and is calculated by using the following equation:

$$K = \text{cubic\_root}(W_{max}*beta/C) \quad \text{(Eq. 2)}$$

where beta is the multiplication decrease factor. We discuss how we set C in the next Section in more details.

Upon receiving an ACK during congestion avoidance, CUBIC computes the window growth rate during the next RTT period using Eq. 1. It sets W(t+RTT) as the candidate target value of congestion window. Suppose that the current window size is cwnd. Depending on the value of cwnd, CUBIC runs in three different modes. First, if cwnd is less than the window size that Standard TCP would reach at time t after the last loss event, then CUBIC is in the TCP friendly region (we describe below how to determine this window size of Standard TCP in term of time t). Otherwise, if cwnd is less than W_max, then CUBIC is the concave region, and if cwnd is larger than W_max, CUBIC is in the convex region. Below, we describe the exact actions taken by CUBIC in each region.

3.2. TCP-friendly region

When receiving an ACK in congestion avoidance, we first check whether the protocol is in the TCP region or not. This is done as follows. We can analyze the window size of Standard TCP in terms of the elapsed time t. Using a simple analysis in [FHP00], we can analyze the average window size of additive increase and multiplicative decrease (AIMD) with an additive factor alpha and a multiplicative factor beta to be the following function:

$$(\alpha/2 * (2-\beta)/\beta * 1/p)^{0.5} \quad \text{(Eq. 3)}$$

By the same analysis, the average window size of Standard TCP with alpha 1 and beta 0.5 is $(3/2 * 1/p)^{0.5}$. Thus, for Eq. 3 to be the same as that of Standard TCP, alpha must be equal to $3*\beta/(2-\beta)$. As Standard TCP increases its window by alpha per RTT, we can get the window size of Standard TCP in terms of the elapsed time t as follows:

$$W_{tcp}(t) = W_{max}*(1-\beta) + 3*\beta/(2-\beta)* t/RTT \quad \text{(Eq. 4)}$$

If cwnd is less than W_tcp(t), then the protocol is in the TCP friendly region and cwnd SHOULD be set to W_tcp(t) at each reception of ACK.

3.3. Concave region

When receiving an ACK in congestion avoidance, if the protocol is not in the TCP-friendly region and cwnd is less than W_max, then the protocol is in the concave region. In this region, cwnd MUST be incremented by (W(t+RTT) - cwnd)/cwnd.

3.4. Convex region

When the window size of CUBIC is larger than W_max, it passes the plateau of the cubic function after which CUBIC follows the convex profile of the cubic function. Since cwnd is larger than the previous saturation point W_max, this indicates that the network conditions might have been perturbed since the last loss 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 phase, CUBIC is being very careful by very slowly increasing its window size. The convex profile ensures that the window increases very slowly at the beginning and gradually increases its growth rate. We also call this phase as the maximum probing phase since CUBIC is searching for a new W_max. In this region, cwnd MUST be incremented by (W(t+RTT) - cwnd)/cwnd for each received ACK.

3.5. Multiplicative decrease

When a packet loss occurs, CUBIC reduces its window size by a factor of beta. Parameter beta SHOULD be set to 0.2.

```
W_max = cwnd;              // save window size before reduction
cwnd = cwnd * (1-beta);    // window reduction
```

A side effect of setting beta to a smaller value than 0.5 is slower convergence. We believe that while a more adaptive setting of beta could result in faster convergence, it will make the analysis of the protocol much harder. This adaptive adjustment of beta is an item for the next version of CUBIC.

3.6. Fast convergence

To improve the convergence speed of CUBIC, we add a heuristic in the protocol. When a new flow joins the network, existing flows in the network need to give up their bandwidth shares to allow the flow some room for growth if the existing flows have been using all the bandwidth of the network. To increase this release of bandwidth by existing flows, the following mechanism called fast convergence SHOULD be implemented.

With fast convergence, when a loss event occurs, before a window reduction of congestion window, a flow remembers the last value of W_max before it updates W_max for the current loss event. Let us call the last value of W_max to be W_last_max.

```
if (W_max < W_last_max){        // check downward trend
    W_last_max = W_max;         // remember the last W_max
    W_max = W_max*(2-beta)/2;   // further reduce W_max
} else {                        // check upward trend
    W_last_max = W_max          // remember the last W_max
}
```

This allows W_max to be slightly less than the original W_max. Since flows spend most of time around their W_max, flows with larger bandwidth shares tend to spend more time around the plateau allowing more time for flows with smaller shares to increase their windows.

4. Discussion

With a deterministic loss model where the number of packets between two successive lost events 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:

$$(C*(4-beta)/4/beta)^{0.25} * RTT^{0.75} / p^{0.75} \quad \text{(Eq. 5)}$$

With beta set to 0.2, the above formula is reduced to:

$$(C*3.8/0.8)^{0.25} * RTT^{0.75} / p^{0.75} \quad \text{(Eq. 6)}$$

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

4.1. Fairness to standard TCP

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

Standard TCP performs well in the following two types of networks:

1. networks with a small bandwidth-delay product (BDP)

2. networks with a short RTT, but not necessarily a small BDP

CUBIC is designed to behave very similarly to standard TCP in the above two types of networks. The following two tables show the average window size of standard TCP, HSTCP, and CUBIC. The average window size of standard TCP and HSTCP is from [RFC3649]. The average window size of CUBIC is calculated by using Eq. 6 and CUBIC TCP friendly mode for three different values of C.

Loss Rate P	TCP	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
10^-2	12	12	12	12	12
10^-3	38	38	38	38	66
10^-4	120	263	120	209	371
10^-5	379	1795	660	1174	2087
10^-6	1200	12279	3713	6602	11740
10^-7	3795	83981	20878	37126	66022
10^-8	12000	574356	117405	208780	371269

Response function of standard TCP, HSTCP, and CUBIC in networks with RTT = 100ms. The average window size W is in MSS-sized segments.

Table 1

Loss Rate P	Average TCP W	Average HSTCP W	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
10^-2	12	12	12	12	12
10^-3	38	38	38	38	38
10^-4	120	263	120	120	120
10^-5	379	1795	379	379	379
10^-6	1200	12279	1200	1200	2087
10^-7	3795	83981	3795	6603	11740
10^-8	12000	574356	20878	37126	66022

Response function of standard TCP, HSTCP, and CUBIC in networks with RTT = 10ms. The average window size W is in MSS-sized segments.

Table 2

Both tables show that CUBIC with any of these three C values is more friendly to TCP than HSTCP, especially in networks with a short RTT where TCP performs reasonably well. For example, in a network with RTT = 10ms and p=10^-6, TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then 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 Standard TCP, whereas HSTCP is about ten times more aggressive than Standard TCP.

We can see that C determines the aggressiveness of CUBIC in competing with other protocols for the bandwidth. CUBIC is more friendly to the Standard TCP, if the value of C is lower. However, we do not

recommend to set C to a very low value like 0.04, since CUBIC with a low C cannot efficiently use the bandwidth in long RTT and high bandwidth networks. Based on these observations, we find C=0.4 gives a good balance between TCP-friendliness and aggressiveness of window growth. Therefore, C SHOULD be set to 0.4. With C set to 0.4, Eq. 6 is reduced to:

$$1.17 * RTT^{0.75} / p^{0.75} \quad \text{(Eq. 7)}$$

Eq. 7 is then used in the next subsection to show the scalability of CUBIC.

4.2. Using Spare Capacity

CUBIC uses a more aggressive window growth function than Standard TCP under long RTT and high bandwidth networks.

The following table shows that to achieve 10Gbps rate, standard TCP requires a packet loss rate of 2.0e-10, while CUBIC requires a packet loss rate of 3.4e-8.

Throughput(Mbps)	Average W	TCP 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	3.3e-4
100	833.3	2.0e-6	2.5e-5	1.6e-5
1000	8333.3	2.0e-8	1.5e-6	7.3e-7
10000	83333.3	2.0e-10	1.0e-7	3.4e-8

Required packet loss rate for Standard TCP, HSTCP, and CUBIC to achieve a certain throughput. We use 1500-byte packets and an RTT of 0.1 seconds.

Table 3

Our test results in [HKLRX06] indicate that CUBIC uses the spare bandwidth left unused by existing Standard TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

4.3. Difficult Environments

CUBIC is designed to remedy the poor performance of TCP in fast long-distance networks. It is not designed for wireless networks.

4.4. Investigating a Range of Environments

CUBIC has been extensively studied by using both NS-2 simulation and test-bed experiments covering a wide range of network environments. More information can be found in [HKLRX06].

4.5. Protection against Congestion Collapse

In case that there is congestion collapse, CUBIC behaves likely standard TCP since CUBIC modifies only the window adjustment algorithm of TCP. Thus, it does not modify the ACK clocking and Timeout behaviors of Standard TCP.

4.6. 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 bandwidth share. When competing flows have different RTTs, their bandwidth shares are linearly proportional to the inverse of their RTT ratios. This is true independent of the level of statistical multiplexing in the link.

4.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC.

4.8. Responses to Sudden or Transient Events

In case that there is a sudden congestion, a routing change, or a mobility event, CUBIC behaves the same as Standard TCP.

4.9. Incremental Deployment

CUBIC requires only the change of TCP senders, and does not require any assistant of routers.

5. Security Considerations

This proposal makes no changes to the underlying security of TCP.

6. IANA Considerations

There are no IANA considerations regarding this document.

7. Acknowledgements

Alexander Zimmermann and Lars Eggert have received funding from the European Union's Horizon 2020 research and innovation program 2014-2018 under grant agreement No. 644866 (SSICLOPS). This document reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

8. References

8.1. Normative References

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, December 2003.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, August 2007.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, April 2012.

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", In Proceedings of IEEE INFOCOM , May 2007.

[FHP00] Floyd, S., Handley, M., and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control", May 2000.

[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 , August 2002.

[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.

[HRX08] Ha, S., Rhee, I., and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", ACM SIGOPS Operating System Review , 2008.

[K03] Kelly, T., "Scalable TCP: Improving Performance in HighSpeed Wide Area Networks", ACM SIGCOMM Computer Communication Review , April 2003.

[LS08] Leith, D. and R. Shorten, "H-TCP: TCP Congestion Control for High Bandwidth-Delay Product Paths", Internet-draft draft-leith-tcp-htcp-06 , April 2008.

[XHR04] Xu, L., Harfoush, K., and I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks", In Proceedings of IEEE INFOCOM , March 2004.

Authors' Addresses

Injong Rhee
North Carolina State University
Department of Computer Science
Raleigh, NC 27695-7534
US

Email: rhee@ncsu.edu

Lisong Xu
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588-01150
US

Email: xu@unl.edu

Sangtae Ha
University of Colorado at Boulder
Department of Computer Science
Boulder, CO 80309-0430
US

Email: sangtae.ha@colorado.edu

Alexander Zimmermann
NetApp
Sonnenallee 1
Kirchheim 85551
Germany

Phone: +49 89 900594712
Email: alexander.zimmermann@netapp.com

Lars Eggert
NetApp
Sonnenallee 1
Kirchheim 85551
Germany

Phone: +49 151 12055791
Email: lars@netapp.com

Richard Scheffenegger
NetApp
Am Euro Platz 2
Vienna 1120
Austria

Phone: +43 1 3676811 3146
Email: rs@netapp.com