INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

CUBIC High-Speed Algorithms Implemented In Linux-2.6.31

Dr. Sarika Agarwal
Dronacharya Group of Institutions, Greater Noida, India
Ms.Aarshi Jain
Dronacharya Group of Institutions, Greater Noida, India
Mr. Ankit Chadha
Dronacharya Group of Institutions, Greater Noida, India

Abstract:

This paper investigates the congestion control capabilities of TCP-CUBIC introduced in Linux Kernel 2.6 for streamlining the kernel deploying CUBIC Congestion Control Protocol and similar High Speed variants . The end-to-end nature of the congestion control module is investigated to counter the trafficCUBIC is an enhanced version of BIC and a high speed variant for Transmission Control Protocol (TCP) in which the size of the window is a cubic function of time since the last lost event. It modifies the linear growth function of the window of the existing TCP standards into a cubic function such that the scalability of TCP would improve over the fast and long distance networks. CUBIC enables the window size to be increased aggressively when the window is far enough from the saturation point, and lets it to become slower when the window is close to the saturation point. This feature allows CUBIC to be very scalable when the bandwidth and delay product of the network is large, and at the same time, be highly stable and also fair to standard TCP flows. The implementation of CUBIC in Linux has gone through several upgrades. In this paper we present an initial design, implementation and performance.

Key words: *AIMD, CUBIC, BIC*

1.Introduction

The transmission control protocol (TCP) is one of thecore protocols of the Internet protocol (IP) suite. It provides reliable end-to-end connections in the Internet. The TCP congestion control mechanism enables the sender to adjust the transmission rate (or equivalently the congestion window size) according to the network conditions dynamically.As the Internet evolves to include many very high speed and long distance network paths, the performance of TCPwas challenged. These networks are characterized by large bandwidth and delay product (BDP) which represents the total number of packets needed in flight while keeping the bandwidth fully utilized, in other words, the size of the congestion window.There are many variations of TCP congestion control mechanisms proposed in the literature.Some of the TCP congestion control protocols which have been deployed in the current Internet include TCP Reno, New Reno [1], and SACK. In these protocols, the window size is reduced if there is a loss event (e.g., three duplicate acknowledgement (ACK), timeout).There are TCP protocols which are designed specifically for long-distance, high latency links. Examples include Fast TCP [2],BIC (Binary Increase Congestion control) TCP[3], and TCP CUBIC [4].Since the release of the algorithm to the Linux community in 2006, apart from the work in [5], CUBIC TCP has not undergone substantial experimental evaluation. A key observation of [5], through experiments, was that CUBIC TCP suffers from slow convergence which would also imply prolonged unfairness between CUBIC TCP flows.

In this paper, our goal is to propose an analyticalmodel to analyze the performance of TCP CUBIC in wireless networks.Then CUBIC enhances the fairness properties of BIC while retaining its scalability and stability. The main feature of CUBIC is that its window growth function is defined in real-time so that its growth will be independent of RTT.Since the first release of CUBIC to the Linux community in 2006, CUBIC has gone through several upgrades.

2.System Model For TCP Cubic

In this section, we first present the network model and state the assumptions of the system model. We then describe CUBIC is an enhanced version of BIC. It simplifies the BIC window control and improves its TCP-friendliness and RTT-fairness.

2.1.Congestion Loss And Random Packet Loss

Consider the network where the bottleneck link is the last hop wireless link. This wireless bottleneck link has a capacity of C bits/sec and is smaller than the capacities of other intermediate links between the source and destination pair. This scenario is applicable to the scenario as in 3GPP (Third Generation Partnership Project) LTE (Long Term Evolution) or WiMAX (Worldwide Interoperability for Microwave Access). The source has a large file to send to the destination. We assume that packet losses are caused by two factors:congestion loss and random packet loss.Congestionlosshappens when the transmission rate attains the maximum capacity C of the bottleneck link. We assume that the average RTT is a constant, which is a common assumption in loss-based TCP analytical modeling (e.g., [7]).

Thus, the maximize congestion window size W is

$\Omega = X \cdot PT\ T.$ (1)

Equivalently, congestion loss happens when window size attains the maximum window size W. Random packet loss is caused by fading or interference in the wireless link. We assume that random packet loss experiences a random Poisson process with rate λ. This assumption has also been made in [8]. Given a time instant t0 , the time duration τloss from time t0 to the next loss event is a random variable with an exponential distribution. The probability density function (pdf) of τloss is

$f(\tau loss) + \lambda \exp(-\lambda \tau loss),\quad \tau loss > 0$ (2)

Given the time instant t0 , the probability that the next loss event happens within the time interval (t0 + T1 , t0 + T2] is

$P(T1 < \tau loss \leq T2) = \exp(-\lambda T1) - \exp(-\lambda T2)$ (3)

2.2.Congestion Control For TCP CUBIC

We now introduce some notations to model TCP CUBICcongestion control. Let τ denote the elapsed time from the last window reduction. The window size just before the last window reduction is denoted by x. The constant α denotes the window growth factor. A large value of α implies faster window growth rate. The constant β represents the multiplicative decrease factor. The window reduces to βx at the time of the last reduction.

In TCP CUBIC, the window size is a cubic function of time since the last loss event. Let $w(x, \tau)$ denote the window size as a function of x and τ . The congestion window of CUBIC is determined.

$$w(x, \tau) = \alpha(\tau - 3\sqrt{(1-\beta)x/\alpha}) + x \qquad (4)$$

The congestion window reduction occurs due to either congestion loss or random packet loss event. When window reduction happens, the window size reduces to β times the window size just before the loss event. After that, it grows according to (4). Let D(x, y) denote the time duration in which the window size grows from βx to

y without encountering another loss event, after the last window reduction happened at the value of x. We have

$$D(x, y) = 3\sqrt{((y-x)/\alpha)} + \sqrt{((1-\beta)x/\alpha)} \qquad (5)$$

2.3. BIC Window Growth Function

Before delving into CUBIC, let us examine the features of BIC. The main feature of BIC is its unique window growth function.

Fig. A shows the growth function of BIC. When it gets a packet loss event, BIC reduces its window by a multiplicative factor β. The window size just before the reduction is set to themaximum Wmax and the window size just after the reduction is set to the minimum Wmin. Then, BIC performs a binary search using these two parameters – by jumping to the "midpoint" between Wmax and Wmin. Since packet losses have occurred at Wmax, the window size that the network can currently handle without loss must be somewhere between these two numbers.

However, jumping to the midpoint could be too much increase within one RTT, so if the distance between the midpoint and the current minimum is larger than a fixed constant, called Smax, BIC increments the current window size by Smax (linear increase). If BIC does not get packet losses at the

updated window size, that window size becomes the new minimum. If it gets a packet loss, that window size becomes the new maximum. This process continues until the window increment is less than some small constant called Smin at whichpoint, the window is set to the current maximum. So the growing function after a window reduction will be most likely to be a linear one followed by a logarithmic one (marked as "additive increase" and "binary search" respectively in Fig. A).

If the window grows past the maximum, the equilibrium window size must be larger than the current maximum and a new maximum must be found. BIC enters a new phase called

"max probing." Max probing uses a window growth function exactly symmetric to those used in additive increase and binary search – only in a different order: it uses the inverse of binary search (which is logarithmic; its reciprocal will be exponential) and then additive increase. Fig. A shows the growth function during max probing. During max probing, the window grows slowly initially to find the new maximum nearby, and after some time of slow growth, if it does not find the new maximum (i.e., packet losses), then it guesses the new maximum is further away so it switches to a faster increase by switching to additive increase where the window size is incremented by a large fixed increment.The good performance of BIC comes from the slow increase around Wmax and linear increase during additive increase and max probing.

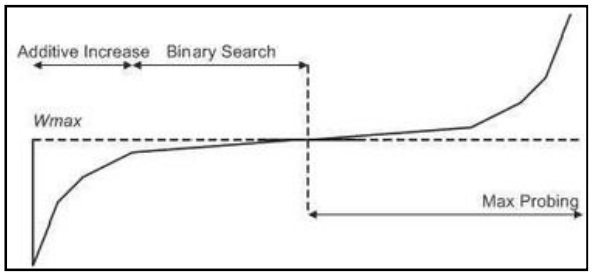

Figure 1: The Window Growth Function of BIC[11]

2.4. CUBIC Window Growth Function

Although BIC achieves pretty good scalability, fairness, and stability during the current high speed environments, the BIC's growth function can still be too aggressive for TCP, especially under short RTT or low speed networks. Furthermore, the several different phases of window control add a lot of complexity in analyzing the protocol. We have been searching for a new window growth function that while retaining most of strengths of BIC (especially, its stability and scalability), simplifies the window control and enhances its TCP friendliness.

In this paper, we introduce a new high-speed TCP variant: CUBIC. As the name of the new protocol represents, the window growth function of CUBIC is a cubic function, whose shape is very similar to the growth function of BIC. CUBIC is designed to simplify and enhance the window control of BIC.

More specifically , the congestion window of CUBIC is determined by the following function:

$$W_{cubic} = C (t - K)^3 + W_{max} \quad (6)$$

where C is a scaling factor, t is the elapsed time from the last window reduction, Wmax is the window size just before the last window reduction, and $K = 3\sqrt{W \max \beta / C}$, where β is a constantmultiplication decrease factor applied for window reduction atthe time of loss event (i.e., the window reduces to βWmax at thetime of the last reduction).

Fig. B shows the growth function of CUBIC with the originat Wmax. The window grows very fast upon a window reduction,but as it gets closer to Wmax, it slows down its growth. AroundWmax, the window increment becomes almost zero. Above that,CUBIC starts probing for more bandwidth in which thewindow grows slowly initially, accelerating its growth as itmoves away from Wmax. This slow growth around Wmaxenhances the stability of the protocol, and increases theutilization of the network while the fast growth away from Wmaxensures the scalability of the protocol.

The cubic function ensures the intra-protocol fairness amongthe competing flows of the same protocol. To see this, supposethat two flows are competing on the same end-to-end path. Thetwo flows converge to a fair share since they drop by the same multiplicative factor β – so a flow with larger Wmax will reduce more, and the growth function allows the flow with larger Wmax will increase more slowly – K is larger as Wmax is larger. Thus,
the two flows eventually converge to the same window size.

The function also offers a good RTT fairness property because the window growth rate is dominated by t, the elapsed time. This ensures linear RTT fairness since any competing flows with different RTT will have the same t after a synchronized packet loss (note that TCP and BIC offer square RTT fairness in terms of throughput ratio).

To further enhance the fairness and stability, we clamp the window increment to be no more than Smax per second. This feature keeps the window to grow linearly when it is far away from Wmax, making the growth function very much in line with

BIC's as BIC increases the window additively when the window increment per RTT becomes larger than some constant.The difference is that we ensure this linear increase of the CUBIC window to be real-time dependent— when under short RTTs, the linear increment per RTT is smaller although stays
constant in real time.

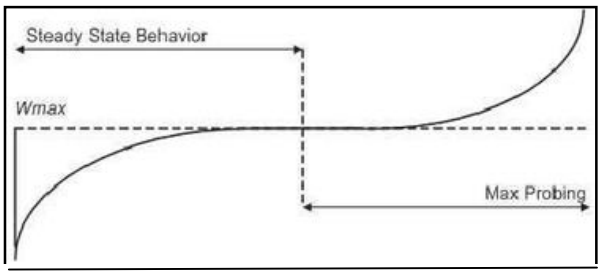

Figure 2: The Window Growth Function of CUBIC[11]

3.Cubic TCP Algorithm

A salient feature of CUBIC TCP [3] is that it uses a cubic window growth function of the elapsed time since the last loss event; the functional form of the window growth . The algorithm makes use of both the concave and the convex profile of the growth function. The window grows very quickly after there is a reduction in the window size. However, as the window size gets closer to Wmax the growth rate slows down. As a consequence of the algorithm behavior, the window spends a lot of time close to Wmax . This is where CUBIC TCP tries to stabilize itself. Once the window is

atWmax , the algorithm again starts probing for more bandwidth; initially, the window grows slowly, and then increases its rate of growth as it moves away from Wmax.

We now outline, in detail, the functioning of the CUBIC algorithm. Upon detecting the loss of apacket,the congestion window is reduced by a multiplicative factor β, where β is a window decrease constant. The window size prior to thereduction is set as Wmax and the current window is increased using the following cubic window growth function

$$W(t) = C(t - K)3 + Wmax, \quad (7)$$

where C is a parameter called a scaling factor, t is the elapsed time since the last window reduction, and K is the time period the above function takes to increase from W to Wmax when no loss is detected. The form of K is given by

$$K=3\sqrt{(Wmax\ \beta/C)}$$

If standard TCP increases its window size by α per round trip time (RTT), its window size in terms of elapsed time is given by

$$Wtcp(t) = Wmax\ (1 - \beta)+(3\beta/2-\beta)(t/RTT)$$

Where$\alpha=(3\beta/2-\beta)$

Depending on the value of the current window size (cwnd),CUBIC operates in the following three different regimes:

IF

cwnd<Wtcp(t) then cwnd =Wtcp(t)

cwnd<Wmax then

cwnd =(cwnd+W(t+RT T)−cwnd)/cwnd

cwnd>Wmax then cwnd = probe for new Wmax

For cwnd<Wtcp(t) , the protocol is in TCP mode, i.e. it behaves similar to the standard TCP Reno. The protocol enters the concave region if cwnd<Wmax . If the window size is greater than Wmax , it indicates the availability of more bandwidth. In this region of concave growth, the window increments are initially slow followed by a gradual increase in its rate, in order to search for a new Wmax . The increased growth rate helps to achieve scalability, whereas the fairness and stability is maintained by forcing an almost linear growth when the window size is far from Wmax .

4.Cubic In Linux Kernel

4.1.Evolution of CUBIC in Linux

It summarizes important updates [9] on the implementation of CUBIC in Linux since its first introduction in Linux 2.6.13. The most updates on CUBIC are focussed on performance and implementation efficiency improvements. One of notable optimizations is the improvement on cubic root calculation. The implementation of CUBIC requires solving a cubic root calculation. The initial implementation of CUBIC [10] in Linux uses the bisection method. But the Linux developer community worked together to replace it with the Newton-Rhaphson method which improves the running time by more than 10 times on average (1032 clocks vs. 79 clocks) and reduces the variance in running times. CUBIC also went through several algorithmic changes to have its current form to enhance its scalability, fairness and convergence speed.

4.2.Pluggable Congestion Module

More inclusions of TCP variants to the Linux kernel has substantially increased the complexity of the TCP code in the kernel. Even though a new TCP algorithm comes with a patch for the kernel, this process requires frequent kernel recompilations and exacerbates the stability of the TCP code.

To eliminate the need of kernel recompilation and help experimenting with a new TCP algorithm with Linux, Stephen Hemminger introduces a new architecture [23, 6], called

pluggable congestion module, in Linux 2.6.13. It is dynamically loadable and allows switching between different congestion control algorithm modules on the fly without recompilation. It shows the interface to this module, named tcp congestion ops. Each method in tcp congestion ops is a hook in the TCP code that provides access to the TCP code. A new congestion control algorithm requires to define cong avoid and ssthresh, but the other methods are optional.

5.Result Analysis Of TCP CUBIC

In this section, we compare the performance of Linux CUBIC TCP w.r.t. AIMD. In the analysis of CUBIC,we use Linux hosts as communication end points communicating over 100Mbps link with MTU of 1500 bytes. The RTT of each background traffic is random. The socket buffer size of some client machines is fixed to default 64KB.We evaluate CUBIC-TCP and AIMD for the bandwidth utilization and RTT

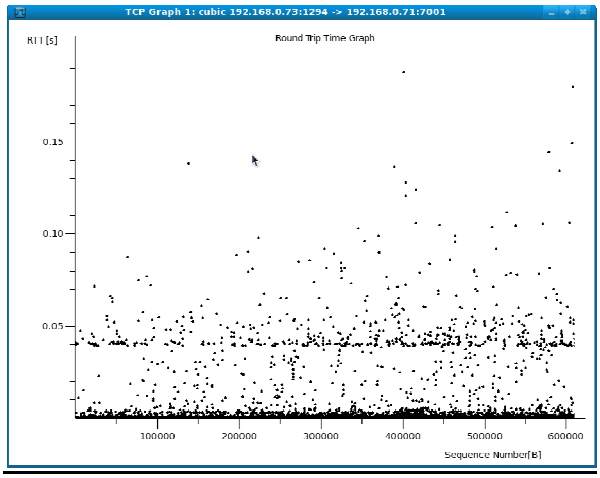

Figure 3: RTT Graph of TCP- CUBIC

As per the graph shown ,the minimum RTT was around 0.001 sec and maximum RTT was around 0.18.

The congestion window of CUBIC is determined by

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

Where,

C=Scaling Factor

t=elapsed time from the last window reduction.

W_{max} =window size=β/C

K=3

β= Constatnt Multiplication window decrease factor.

t=0.18

C=0.4 and β=0.8[10]

K=3√ 65535*08/04=50.7965

Wcubic=0.4(0.18-50.7965)+65535

=13662.601 or 13663 approx.

In this Graph we observe that, CUBIC starts probing for bandwidth in which the window grows slowly initially, accelerating its growth as it moves away from Wmax. This slows growth W_{max} enhances the stability of the protocol, and increases the utilization of the network while the fast growth away from W_{max} ensures the scalability of the protocol.

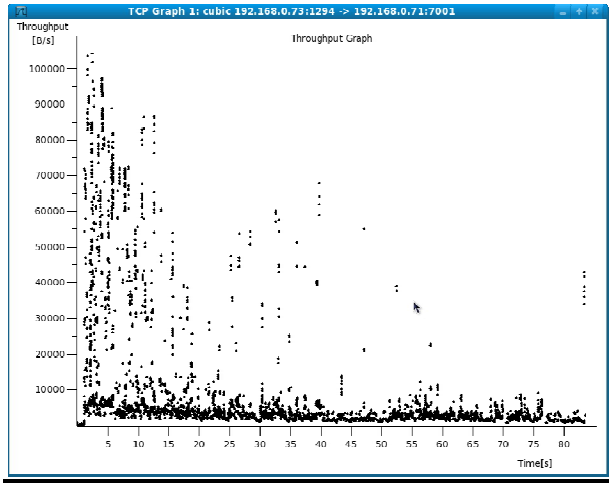

Figure 4

CUBIC TCP achieves greater utilization then standard duTCP . The Graph shows that there is almost steady state due to CUBIC. The highest peak of throughput goes to 105000B/s and with an immediate corrective congestion window size afterwards. Earlier there was some sharp.

5.Conclusion

Our motivation was to conduct an overall brief evaluation of CUBIC TCP which is a current default TCP implementation in Linux. We proposed a new TCP variant, called CUBIC, for fast and long distance networks. CUBIC is an enhanced version of BIC-TCP. CUBIC helps in simplifying the BIC-TCP window control and improves its friendliness with TCP and RTT-fairness. CUBIC uses a cubic increase function in terms of the elapsed

time since the last loss event. In order to provide fairness to Standard TCP, CUBIC also behaves like Standard TCP when the cubic window growth function is slower than Standard TCP. Also, the real-time nature of the protocol keeps the window growth rate independent of RTT,which keeps the protocol TCP friendly under both short and long RTT paths. We have shown the details of Linux CUBIC algorithm and implementation. Through extensive testing, we confirm that CUBIC tackles the shortcomings of BIC-TCP and achieves fairly good Intra-protocol fairness, RTT-fairness and TCP-friendliness.

6.Reference

1. S. Floyd, T. Henderson, and A. Gurtov, “The NewReno modification to TCP’s fast recovery algorithm,” IETF RFC 3782, Apr. 2004.
2. D. X. Wei, C. Jin, S. H. Low, and S. Hegde, “FAST TCP:Motivation, architecture,algorithms, performance,” IEEE/ACM Trans. on Networking, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.
3. L. Xu, K. Harfoush, and I. Rhee, “Binary increase congestion control (BIC) for fast long-distance networks,” in Proc. of IEEE Infocom, Hong Kong, China, Mar. 2004.
4. I. Rhee and L. Xu, “A new TCP-friendly high-speed TCP variant,” in Proc. PFLDNet’05, Lyon, France, Feb. 2005.
5. D. Leith, R.N. Shorten, and G. McCullagh, “Experimental evaluation of Cubic-TCP,” International Workshop on Protocols for Fast Long Distance Networks, 2007.
6. R. Shorten, and D. Leith, "H-TCP: TCP for High-Speed and Long-Distance Networks,” Second International Workshop on Protocols for Fast Long-Distance Networks, February 16-17, 2004, Argonne, Illinois USA.
7. J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: A simple model and its empirical validation,” in Proc. Of ACM SIGCOMM, Vancouver, Canada, Sept. 1998.
8. S. Hassayoun, P. Maille, and D. Ros, “On the impact of random losses on TCP performance in coded wireless mesh networks,” in Proc. Of IEEE Infocom, San Diego, CA, Mar. 2010.
9. Git logs for CUBIC updates. http://git.kernel.org/?p=linux/kernel/git/davem/net-2.6.git;a=history;f=net/ipv4/tcpcubic.c; h=eb5b9854c8c7330791ada69b8c9e8695f7a73f3d;hb=HEAD.
10. Ha, S. Cubic v2.0-pre patch. http://netsrv.csc.ncsu.edu/twiki/pub/Main/BIC/cubic-kernel-2.6.13.patch.
11.]http://ilab.cs.byu.edu/wiki/CUBIC:_A_New_TCP-Friendly_High-Speed_TCP_Variant (google images)