Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)

Cubic with Faster Convergence: An Improved Cubic Fast Convergence Mechanism

Ning Cao
Dept. of Computer Science & Technology
East China Normal University
Shanghai, China
caoning1985@hotmail.com

Wei Zhang
Dept. of Computer Science & Technology
East China Normal University
Shanghai, China
wzhang@cs.ecnu.edu.cn

***Abstract*—Cubic TCP [1] is a high-speed TCP variant protocol which has been the default TCP implementation in Linux since kernel-2.6.18. In high bandwidth-delay product environment, it can achieve a full utilization of the available bandwidth. However, Cubic TCP has a shortcoming in terms of convergence speed [4]. In this paper, we verify the slow convergence problem of Cubic TCP on testbed. After that, we propose an improved fast convergence mechanism which is called Cubic with Faster Convergence. The feature of this new mechanism is that we use a new bandwidth releasing method instead of simply cut down ssthresh directly. Testbed experiments show that this new mechanism can significantly reduce the convergence time, at the same time, do not affect other performance of Cubic TCP.**

Keywords-Congestion Control; Cubic TCP; Fast Convergence

I. INTRODUCTION

TCP (Transmission Control Protocol) provides a reliable data transmission mechanism. Most of the applications use TCP to transmit data over the Internet. However, it has been reported that standard TCP substantially underutilizes network bandwidth over high-speed and long distance networks [2]. To solve this problem, researchers have been proposing a variety of TCP variant protocols. As the default TCP implementation of Linux since kernel-2.6.18, Cubic TCP is the most widely used TCP variant. It replaces the linear increase function of standard TCP with a cubic increase function, and the increasing of *cwnd* is RTT independent. Because of these characteristics, the growth of *cwnd* in Cubic is more aggressive, so it can gain more bandwidth in high bandwidth-delay product environment. But these characteristics also cause the problem of slow convergence i.e. when a new Cubic flow shares the same bottleneck link with an existing Cubic flow, it takes a long time to achieve fair bandwidth allocation. The author of paper [3] verified this shortcoming of Cubic on Dummynet testbed.

Convergence time is an important metric of TCP: with shorter convergence time, new users can get fair network resource faster; also, when a new flow joins into the network, the local stability of this network system is perturbed from its stable state [4], the less convergence time TCP needs means the network system can return toward the locally stable state quicker. So, it not only can improve the fairness of the allocation of network resource, but also can improve the stability of whole network system.

In this paper, an improved Cubic fast convergence mechanism is proposed, which is called Cubic-FC (Cubic with Faster Convergence). This new mechanism does not change the increasing function of Cubic but divides the congestion avoidance into two phases: 1) competition detection phase; and 2) bandwidth releasing phase which uses a new bandwidth releasing method.

The rest of paper is organized as follows: Section II briefly describes Cubic TCP and its fast convergence mechanism. Section III elaborates our new fast convergence mechanism. Section IV gives the detail of the setup of our testbed and experiments result. Finally, conclusions are provided in Section V.

II. CUBIC TCP

The most important feature of Cubic TCP is that it uses a cubic window increasing function of the elapsed time from the last windows reduction:

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

where C is a Cubic parameter, t is the elapsed time from the last loss event, and K is the time period that the above function takes to increase $W_{(t)}$ to W_{max} when there is no further loss event. K is calculated by using the following equation:

$$K = \sqrt[3]{\frac{W_{max} * \beta}{C}}$$

where β is the window decrease constant.

From the above cubic function, we can learn: when current *cwnd* is less than W_{max}, the increasing of *cwnd* is a convex curve; when current *cwnd* is more than W_{max}, the increasing of *cwnd* is a concave curve. This divides the growth process of *cwnd* into "Steady State Behavior" and "Max Probing". The curve of *cwnd* is shown in Fig. 1.

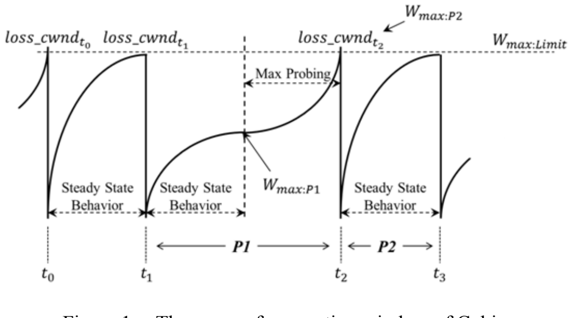

Figure 1. The curve of congestion window of Cubic

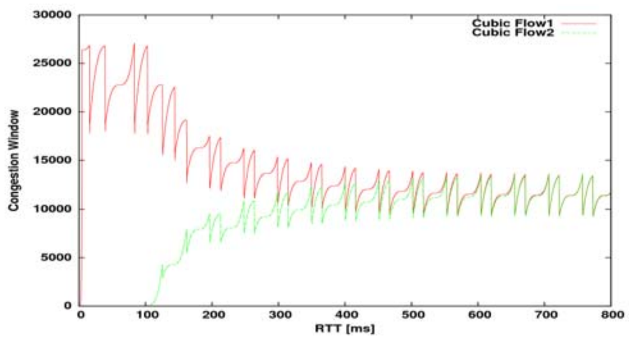

Figure 2. Congestion window size versus time for the original Cubic. Bandwidth of bottleneck link is 800Mbps, RTT is 200ms, and queue size is 100% BDP, no web traffic. Flow 1 starts at 0(s) and ends at 800(s), flow 2 start at 100+x(s) (0<x<1) and ends at 800(s).

Cubic executes the fast convergence algorithm when congestion is detected. Firstly, it determines whether the *cwnd* of current loss event (*loss_cwnd*) is less than the last W_{max} : if it is, it updates W_{max} to $loss\_cwnd * \frac{2-\beta}{2}$; otherwise, W_{max} is set to *loss_cwnd*. After that, Cubic sets ssthresh to $loss\_cwnd * (1-\beta)$, as shown in Fig. 1.

In a stable network environment, we assume that the maximum *cwnd* of a TCP flow is $W_{max:Limit}$ We can learn from the windows growth function of Cubic that when getting close to $W_{max:Limit}$, *cwnd*'s increasing rate in the "Max Probing" status is greater than it in the "Steady State Behavior" status. In Fig. 1, t_0, t_1, t_2, t_3 are the moments when congestion happens. Before time t_1, Cubic is in the "Steady State Behavior" status and it is in the "Max Probing" status before time t_2, the congestion window at time t_1 is smaller than the congestion window at time t_2. Also, we can get that $cwnd_{t_0} > cwnd_{t_1}$ and $cwnd_{t_2} > cwnd_{t_3}$, which leads that $W_{max}$of phase P1: $W_{max:P1} = cwnd_{t_1} * \frac{2-\beta}{2}$, and Wmax of phase P2: $W_{max:P2} = cwnd_2$. This kind of effect is mutual which leads to the periodical execution of these two different kinds of growth.

We took an experiment on our testbed to test the convergence time of Cubic. Fig. 2 shows the curves of *cwnd* of two Cubic flows. As we can see, Cubic needs about 400s to get to the convergence status.

Because flows with larger *cwnd*'s are more aggressive than flows with smaller *cwnd*'s, they can gain more bandwidth than flows with smaller *cwnd*'s within the same time. New flows are thus at a disadvantage and sustained unfairness can occur. "Slow convergence" is considered a general problem in other high-speed TCP variant protocols [5].

III. CUBIC-FC: CUBIC WITH FASTER CONVERGENCE

As the description in section II, W_{max} of phase P2 is greater than phase P1, it leads to the fact that the duration of P2 is much less than P1, so *cwnd* in phase P2 can arrive to the window limitation faster. Such growth pattern is suitable for maintaining the bandwidth already obtained. As the duration of P1 is much longer, *cwnd* needs more time to get to the window limitation, which makes phase P1 suitable for probing competitive flows and releasing bandwidth. We take advantage of this feature of Cubic and divide the congestion avoidance of Cubic into two phases: competition detecting and bandwidth releasing. The *cwnd* curve of our mechanism is shown in Fig. 3.

A. Competition detection

In this phase, Cubic-FC detects whether there is (are) competition flow(s) on the bottleneck link, and calculate the decrement of *cwnd* (Δ) at the end of this phase, as shown in Fig. 3. If Δ is bigger than $loss\_cwnd * \sigma$, where σ is small parameter (0<σ<0.1), it means that competition flows may exist on the bottleneck link. In the next bandwidth releasing phase, Cubic will release a certain amount of bandwidth that calculated by this Δ.

As we described before, it takes more time in CP1 phase, flow in this phase would more likely be influenced by short-term burst, which makes *cwnd* at t_1 may not accurately reflect the competition on the bottleneck link. To minimize the possibility of this situation, we use CP2 phase to confirm the decrement of *cwnd*, that is Δ=min$\{cwnd_{t_0} - cwnd_{t_2}, \Delta_{max}\}$, where Δ_{max} is the maximum of the decrement of *cwnd*, and $\Delta_{max} = loss\_cwnd * \beta$

B. Bandwidth releasing

The convergence speed of TCP depends on two factors [3]: a) the rate at which individual flows release bandwidth when informed of congestion; and b) the rate at which individual flows acquire available bandwidth. The increasing function of Cubic is aggressive enough so that it is not necessary to adopt a more aggressive method. To reduce the convergence time, the only way is to select an appropriate method to release bandwidth. There are two ways to release bandwidth: a) using a bigger decrease parameter when informed of congestion; and b) slowing down the increasing in congestion avoidance. We take advantage of both of these two methods in our mechanism, when detected competition flows:

- Set ssthresh to $loss\_cwnd * (1-\beta) - f(\Delta) * \Delta_{max}$ instead of $loss\_cwnd * (1-\beta)$ in original Cubic;
- Set W_{max} to $loss\_cwnd * \frac{2-\beta}{2} - f(\Delta) * \frac{\Delta_{max}}{2}$ instead

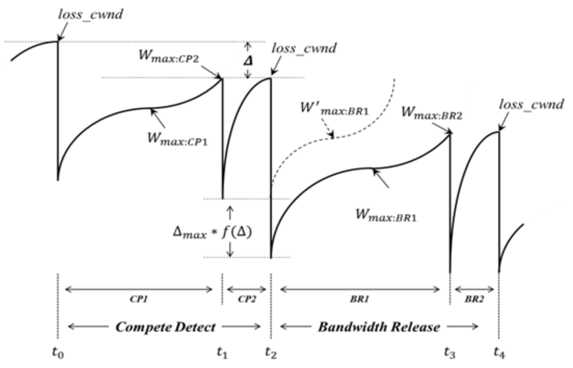

Figure 3. The curve of congestion windows of Cubic-FC

of $loss\_cwnd * \frac{2-\beta}{2}$ in original Cubic;

- Prolong the duration of two congestion event by decrease t to $t * (1 - f(\Delta))$, where t is the elapsed time from the last loss event and calculated after receive an ACK.

loss_cwnd is the *cwnd* of congestion event at the end of last phase, instead of *cwnd* of last congestion event in original Cubic, as shown in Fig. 3. β is the decrease parameter of Cubic as in the original Cubic.

$f(\Delta)$ is a convex function which the range of values is from 0 to f_{max}, where f_{max} is the maximum of $f(\Delta)$ and f_{max}<1. Also $f(\Delta)$ should meet the following requirements:

- If $\Delta = 0$ or $\Delta = \Delta_{max}$, then $f(\Delta) = 0$, and when Δ gets closer to 0 or Δ_{max}, $f(\Delta)$ is closer to 0;
- If $\Delta = \Delta_i$ (Δ_i is a certain value between 0 and Δ_{max}), $f(\Delta) = f_{max}$, and Δ gets closer to Δ_i, $f(\Delta)$ is closer to f_{max}.

The reason $f(\Delta)$ needed these restricts is that:

1) If Δ is closer to 0, it means that the reduction of *cwnd* is more due to normal jitter in the network, we don't need to release bandwidth redundantly;
2) If Δ is closer to Δ_{max}, it implies that Cubic has already released an amount of bandwidth of Δ; if $\Delta = \Delta_{max}$, it may be a multiple packet losses event, and Cubic has already released an amount of bandwidth of Δmax, it's not necessary to release bandwidth repeatedly.

As shown in Fig. 3, in phase BR1 of bandwidth releasing phase, the dotted line is the window increasing curve of original Cubic, and $W'_{max:BR1}$ is the W_{max} of original Cubic in this phase.

After releasing bandwidth in BR1 of bandwidth releasing phase, Cubic needs to maintain the bandwidth and restrain the growth of the competition flows. Otherwise, as the window growth of Cubic is quite aggressive, the new Cubic flow may in turn exceed the old flow, leads to a new unfairness. To achieve this, we simply set W_{max} of BR2 phase to the *cwnd* of congestion event at the end of BR1 phase and set $f(\Delta)$ to 0 that makes Cubic increasing *cwnd* as aggressive as original algorithm.

IV. Experiment Result

In this section, we present the detail of our testbed and the experiment. For each scenario, we took at least 10 times and calculated average value.

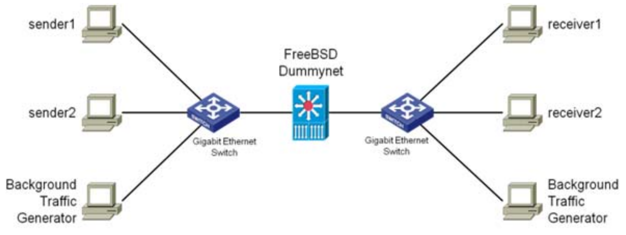

Figure 4. Topology of dummynet testbed

TABLE I. System configuration of our testbed

CUP	Intel Core i5-2310 2.9GHz
Memory	4Gbyte
NIC	Intel Pro 1000PT DUAL PORT SERVER ADAPTER
NIC Driver	e1000 7.1.7
Txqueuelen	1000
Max_backlog	2500
TX & RX Descriptors	4096

A. Testbed setup

The experiment testbed we are using is a network with a dumbbell topology, see Fig. 4. There are three machines running Linux 2.6.32 with identical hardware and software configurations at each edge of the network. *Iperf* [6] is used to generate TCP traffic in the senders and receivers. The rest two machines are used to generate background traffic by *Surge* [7] and *Iperf*. The amount of background traffic is about 10% of the bandwidth bottleneck link in both forward and backward directions. The router is running the FreeBSD Dummynet software. In our experiments, the bandwidth of the bottleneck router is set to 800Mbps, and RTT is varied from 50ms to 300ms. The bottleneck buffer size is set to 100% BDP. We use the drop-tail router at the bottleneck.

The detail of system configuration is shown in Table I.

The function $f(\Delta)$ is an important factor for Cubic-FC. In our experiments, we simply chose a linear function as follow:

$$f(\Delta) = \begin{cases} f_{max} * \dfrac{\Delta}{\Delta_i}, if\ \Delta \le \Delta_i \\ f_{max} * \dfrac{\Delta_{max} - \Delta}{\Delta_i}, if\ \Delta > \Delta_i \end{cases}$$

where $\Delta_i = \frac{\Delta_{max}}{2}$ and $f_{max} = 0.6$. And also, in competition detection phase, the small parameter σ which is used to determine whether there are competition flows is set to 0.05.

B. Convergence Time

First of all, we evaluated the convergence time of Cubic-FC in the same scene as the experiment in section II. Fig. 5 shows the congestion window size for two Cubic-FC flows.

Compare to original Cubic which needs about 400s to

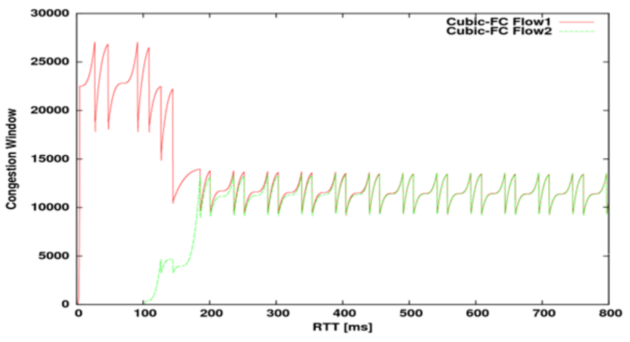

Figure 5. Congestion window size versus time for the Cubic-FC. Bandwidth of bottleneck link is 800Mbps, RTT is 200ms, and queue size is 100% BDP, no web traffic.

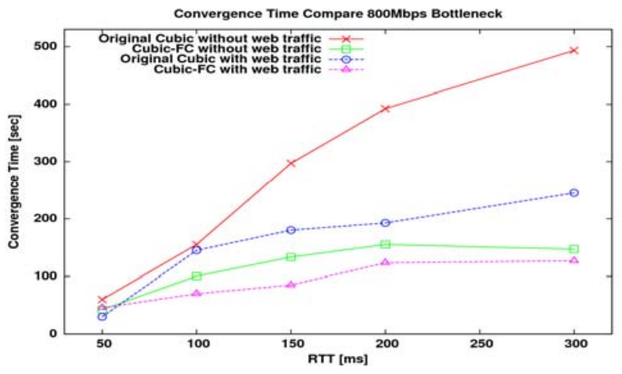

Figure 6. Convergence time compare between original Cubic and Cubic-FC. Bandwidth is 800Mbps, the running time is 600s, and RTT is varied from 50ms to 300ms. 100%BDP

achieve to the convergence state as shown in Fig. 2, Cubic-FC flows just need less than 200s.

We also took a series of experiments to confirm the improvement of our mechanism. In each experiment, flow 1 starts at 0(s) and ends at 600(s), flow 2 start at 100+x(s) (0<x<1) and ends at 600(s). Fig. 6 shows the convergence times with and without background traffic. We can see Cubic-FC needs shorter time to achieve convergence than original Cubic, and with the RTT gets bigger, Cubic-FC can achieve more improvement. In Fig. 6 we can also see that even background traffic breaks the global synchronization, Cubic-FC can converge faster than original Cubic.

C. *Throughput*

Convergence time can affect the throughput of TCP flows, as the shorter convergence time they need, two TCP flows can achieve fairer throughput.

Fig. 7 shows us the difference of the average throughput of two flows, i.e. *average throughput of Flow 1 - average throughput of Flow 2*. We can see that, whether with or without background traffic, the two flows of Cubic-FC always can achieve fairer throughput than original Cubic.

To test the impact to the total throughput in our fast convergence mechanism, we measured the total throughput of original Cubic and Cubic-FC in different scenes.

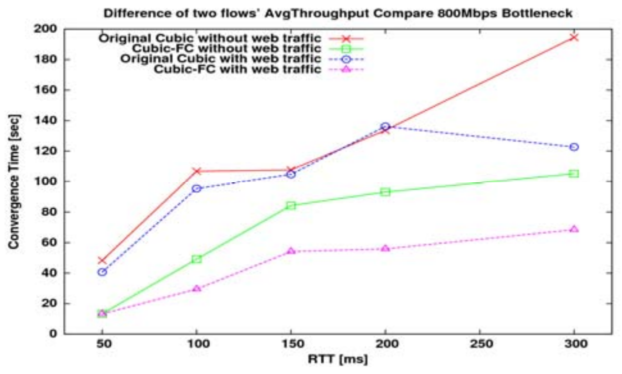

Figure 7. Average throughput of two flows compare between original Cubic and Cubic-FC. Bandwidth is 800Mbps, the running time is 600s, and RTT is varied from 50ms to 300ms. 100%BDP

TABLE II. Total Throughput Compare

RTT (ms)	Total Throughput (Mbps)			
	Without background traffic		With background traffic	
	Cubic	Cubic-FC	Cubic	Cubic-FC
50	772.173	772.058	685.203	680.896
100	772.259	772.062	685.68	681.452
150	772.590	772.181	686.053	683.176
200	772.334	772.135	684.754	681.869
300	772.814	772.057	683.844	682.086

Table II shows the compare of total throughput of two mechanisms. We can see our mechanism does not make any serious impact of the total throughput: in the environment without background traffic, the maximum decrement of total throughput is only 0.098%, and the average value is only 0.014%; in the environment with background traffic, these two values are 0.63% and 0.46%, respectively. As the duration of two congestion event is longer and congestion window grows slower in bandwidth release phase of Cubic-FC, the total throughput of Cubic-FC is little smaller than original Cubic, but it is worthy that the slight decline of total throughput in exchange the upgrade of convergence speed.

V. CONCLUSIONS

This paper has proposed a new convergence mechanism of Cubic TCP, named Cubic with Faster Convergence (Cubic-FC for short). We divided the congestion avoidance into two phases: competition detection phase and bandwidth releasing phase. In competition detection phase, Cubic-FC detects the competition flows, and in bandwidth releasing phase, we use a new bandwidth releasing method to release a certain amount of bandwidth according to the decrement of *cwnd* in competition detection phase.

We used testbed rather than simulation, which is closer to realistic network environment. Testbed experiments show that Cubic-FC can converge faster than original Cubic in the environment with and without background traffic, at the same time, it do not seriously impact to the other performance of Cubic.

REFERENCES

[1] S. Ha, I. Rhee, and L. Xu, “CUBIC: A new TCP-friendly high-speed TCP variant,” ACM SIGOPS Operating Systems Review, vol. 42, no. 5, pp. 64–74, 2008.

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

[3] D. Leith, R. Shorten, and G. McCullagh. Experimental evaluation of cubic-TCP. In Proc. Protocols for Fast Long Distance Networks 2007, 2007.

[4] D. Papadimitriou, M. Welzl, M. Scharf and B. Briscoe, “RFC 6077: Open Research Issues in Internet Congestion Control”, RFC 6077, INFORMATIONAL, February 2011.

[5] Yee-Ting Li , Douglas Leith and Robert N. Shorten, “Experimental evaluation of TCP protocols for high-speed networks”, IEEE/ACM Transactions on Networking (TON), v.15 n.5, p.1109-1122, October 2007.

[6] Iperf. http://sourceforge.net/projects/iperf .

[7] Barford, P., and Crovella, M. Generating representative web workloads for network and server performance evaluation. In Measurement and Modeling of Computer Systems (1998), pp. 151–160.