International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 4, Number 2, pages 291–306, July 2014

Improving RTT Fairness on CUBIC TCP

Tomoki Kozu, Yuria Akiyama, Saneyasu Yamaguchi
Kogakuin University,1-24-2
Nishi-Shinjuku, Shinjuku-Ku, Tokyo, Japan

Received: February 15, 2014
Revised: May 6, 2014
Accepted: June 3, 2014
Communicated by Yoshiaki Kakuda

Abstract

CUBIC TPC is a congestion control algorithm for TCP. It is the current default TCP algorithm in Linux. Because many Internet servers, such as web servers, are running on Linux operating system, keeping throughput obtained with this TCP enough is quite important. Then, many performance studies have been published. However, most of these studies have been based on network simulators. Thus, evaluations using an actual TCP implementation and actual network elements are important in addition to these existing studies. In this paper, we focus on RTT (round trip time) fairness on CUBIC TCP, which is performance fairness among CUBIC TCP connections with different network delay times. Firstly, we present RTT fairness evaluation using actual TCP implementations and actual network elements and show that the fairness is not enough. Secondly, we discuss the cause of the unfairness based on CUBIC TCP behaviors. Thirdly, we propose a method for improving RTT fairness of CUBIC TCP. Unlike an existing work, the proposed method is not based on heuristic optimization. Finally, we present evaluation results and demonstrate that the proposed method provides better fairness than original CUBIC TCP implementation.

Keywords: TCP, congestion control algorithm, CUBIC TCP, RTT fairness

1 Introduction

TCP (Transmission Control Protocol) is one of the core protocols of the Internet. Its implementations have congestion control algorithms and their behavior has strong impact on obtained throughput. Classical TCP implementations and some current implementations have TCP Reno [1]. It has been widely used, but it is pointed out that enough throughput cannot be obtained over current long fat networks with this algorithm [2]. For this issue, several new congestion control algorithms were proposed, such as TCP Vegas [3], BIC TCP [4], CUBIC TCP [5], and Compound TCP [6]. CUBIC TCP is considered one of the most suitable algorithms for current networks. Then, it is the default TCP algorithm of current Linux operating system.

On TCP congestion control algorithms, their performance has been rigorously discussed using network simulators, such as NS2 [7]. However, performance obtained with actual TCP implementations and actual network elements have not been studied enough. Especially, RTT fairness, which is performance fairness between connections with different RTTs, has not been discussed.

In this paper, we focus on RTT fairness in CUBIC TCP. We evaluate RTT fairness of CUBIC TCP connections and demonstrate that the fairness is not sufficient. Then, we discuss the cause of unfairness. After the discussion, we propose a method for improving RTT fairness. CUBIC TCP is one of the most important TCP congestion control algorithms because most Internet traffic is sent from servers, such as web server processes, to client PCs and most of these server processes are running on Linux operating system. Thus, we think improving RTT fairness of this TCP implementation contributes many users of the Internet. We evaluate "fairness" with Fairness Index, which will be described in section 3.1, and define our objective to improve this value.

The remainder of this paper is organized as follows. Section 2 gives related work. Section 3 provides evaluation results of RTT fairness on CUBIC TCP and demonstrates that the fairness is not enough. After the evaluation, the section presents discussion on cause of the unfairness and shows K of CUBIC TCP is the main cause. From these discussions, we propose a method for improving RTT fairness of CUBIC TCP and evaluate the proposed method in section 4. Section 5 presents discussion and Section 6 gives conclusion.

2 Related Work

2.1 TCP Congestion Control Algorithms

In order to avoid network congestion, TCP implementations manage congestion window size and control output speed. There are many TCP congestion control algorithms. These are classified into three general groups, loss-based methods, delay-based methods, and hybrid methods.

Loss-based methods manage congestion window size based on packet losses. In usual cases, congestion window size is increased every ACK packet receiving. When a packet loss is detected, congestion window size is decreased significantly. TCP Reno [1], BIC TCP [4], and CUBIC TCP [5] are loss-based methods.

Delay based methods manage congestion window size based on RTT. These methods decrease congestion windows size according to RTT increase, which implies increase of network routers' load and queue length. While loss-based methods decrease its congestion window size after congestion, delay-based methods decrease it before congestion. Thus, obtained throughput is expected to be stable. However, these methods have a performance issue. In case of sharing a network link with a loss-based method, performance obtained by a delay-based method is much less than that of a loss-based method, because a delay-based method decreases its congestion window size before congestion and a loss-based method does not decrease it until a packet loss. TCP Vegas [3] is a delay-based method.

Hybrid methods are methods adopting both loss-based policy and delay-based policy. Compound TCP [6] is a hybrid method.

2.2 CUBIC TCP

CUBIC TCP [5] is an algorithm based on BIC TCP [4], so it has high scalability similar to BIC TCP. Moreover, it has better TCP fairness and RTT fairness than BIC TCP. TCP fairness is performance fairness among TCP algorithms. RTT fairness is performance fairness among connections with different RTTs, as described above. CUBIC TCP uses the following cubic function, while BIC TCP uses binary search.

$$cwnd = C(t - K)^3 + W_{max} \tag{1}$$

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{2}$$

$cwnd$ is congestion window size, t is time from the last packet loss, W_{max} is congestion window size at the last packet loss, C and β are parameters to tune increasing speed in usual state and dropping ratio at packet losses, respectively. The larger C results in the faster increase. In most cases, C is 0.4 and β is 0.2. Because this function uses time from a packet loss and does not depend on ACK receiving, it is expected to provide good RTT fairness and avoid too aggressive increasing over a

short RTT network. As shown in formula (1), K is time to the inflection point and *cwnd* achieves W_{max} at time K. Therefore, we can understand K as time to recovery.

CUBIC TCP calculates congestion window size obtained by TCP Reno using the following function.

$$W_{max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{1}{RTT} \tag{3}$$

If its current congestion window size is less than that obtained by TCP Reno, then CUBIC TCP adopts this calculated size.

For giving enough network bandwidth to a new joining flow, CUBIC TCP has Fast Convergence mechanism. If congestion window size at the last packet loss is less than that of the preceding packet loss, then W_{max} is determined by the following function. This mechanism decreases bandwidth of the existing flows in order to increase performance of a new joining flow [8].

$$W_{max} = cwnd\frac{2-\beta}{2} \tag{4}$$

2.3 Fairness on TCP

The following works are on RTT fairness of TCP. For achieving RTT fairness, Floyd *et al*. proposed Constant-Rate window increase algorithms [9] [10], with which each connection increases its window size by roughly $a * r^2$ packets each roundtrip time, for some fixed constant a, and for r the calculated average roundtrip time. Henderson *et al*. profoundly investigated RTT fairness and showed Constant-Rate policy [9] could improve fairness dramatically [11].

Marfia *et al*. suggested that TCP's RTT unfairness was caused by its ACK-based mechanism and proposed a new TCP algorithm named TCP-Libra [12]. It increases congestion window size of large RTT connections more quickly. Ogura *et al*. proposed a new TCP algorithm, which is named "HRF (Hybrid RTT Fair)-TCP" for improving RTT fairness [13]. This work also presented analytical models. They evaluated the proposed method with both a simulator and their implementation.

The works in [9], [10], and [11] were based on additive increase policies. Therefore, these works are useful for classical TCP algorithms, such as TCP Reno, but are not directly effective for modern fast TCP algorithms and modern operating systems. On the other hand, our work focuses on one of the most important modern fast TCP algorithms and discusses with a practical TCP implementation. In addition, these methods do not have a target situation and do not have mechanism to lead throughput close to fair one. On the contrary, our method has a mechanism with which performance of insufficient and exceeds throughput connections are increased and decreased, respectively. Performance comparison between Constant-Rate and our proposed method is presented in appendix A. The methods in work [12] and [13] are pioneer ones, but their purposes are quite different from that of our work. They discussed not for CUBIC TCP but for a novel TCP algorithm. TCP-Libra increases its congestion window size based on ACK receiving unlike CUBIC TCP. On the other hand, we focus on CUBIC TCP, which is currently widely used, and aim to improve it.

The following works are on TCP fairness. Mo *et al*. evaluated fairness between TCP Reno and TCP Vegas [3]. The work demonstrated that TCP Vegas reduced its performance while TCP Reno monotonously increased, and their fairness was poor. The evaluation was executed with network simulator (NS). Itsumi *et al*. showed that CUBIC TCP outperformed Compound TCP when they shared network links [14]. In addition, they proposed a method to improve fairness by using RED (Random Early Detection) [15], and demonstrated that the proposed method worked effectively. The evaluation was executed with simulation. In [16], TCP fairness among modern fast TCPs were evaluated with actual TCP implementations and several solutions for this issue were proposed. These works are for TCP fairness, thus they are not directly useful for improving RTT fairness.

In work [17], a method for improving RTT fairness of CUBIC TCP was proposed. The proposed method was evaluated with an actual TCP implementation and actual network elements. However, the proposed method was based on a tuning function. Thus, achieved fairness depended on heuristic optimization. On the contrary, the method proposed in this paper can improve RTT fairness of CUBIC TCP without parameters tuning. The comparison between this existing method and the proposed method in this paper will be described in appendix B.

3 RTT Fairness of CUBIC TCP

3.1 RTT Fairness Evaluation

In this subsection, we give RTT fairness evaluation between CUBIC TCP connections. We constructed the experimental network shown in Figure 1 and measured CUBIC TCP performance.

The executed experiments are as follows. We established two CUBIC TCP connections. One was a connection between PC1 and PC3. The other was a connection between PC2 and PC3. These connections were established by netperf [18], which was a network throughput benchmark software. Packets were sent from PC1 and PC2 to PC3. The network delay times of these connections were set independently. Both of the connections shared network elements between the emulator and PC3. The network emulator was a network bridge which was able to emulate network delay. This emulator was constructed by FreeBSD Dummynet [19]. All elements in this experimental network supported 1Gbps Ethernet. Specifications of the computers in this network are shown in Table 1. C and β of CUBIC TCP were set 0.4 and 0.2, respectively. These are the standard settings [5] and the default settings of the current Linux operating system. Advertise windows size was set 16MB.

The experimental results are shown in Figure 2 to Figure 4. In Figure 2, the network delay time of one connection was always 2ms. The delay time of the other connection ranged from 2ms to 128ms. The horizontal axes of these graphs represent RTTs of the two connections. The vertical axis in the left graph shows the obtained throughput and the vertical axis of the right graph depicts Jain's Fairness Index [20]. For n flows, with flow i receiving a fraction b_i on a giving link, the fairness of the allocation is defined as:

$$Fairness\ Index = \frac{\left(\sum_{i=1}^{n} b_i\right)^2}{n \times \sum_{i=1}^{n} (b_i^2)} \tag{5}$$

It ranges continuously in value from $1/n$ to 1, with 1 corresponding to equal allocation for all connection [11]. The left graph indicates that the throughput obtained by the connection with the smaller RTT is higher than that with larger RTT. These graphs indicate also that performance difference grows and Fairness Index declines as RTT difference grows. In the case of RTT 2ms and RTT 128ms, one connection obtained more than twice performance. From these, we can conclude that RTT fairness of CUBIC TCP is not enough.

Figure 3 and Figure 4 show congestion window size of the experiment with RTT 2ms and RTT 128ms, respectively. The average congestion windows size of connections with RTT 2ms and RTT 128ms are 0.82MB and 2.83MB, respectively. Because obtained throughput is less than $WindowSize/RTT$ [21], connections with RTT 2ms and RTT 128ms should increase congestion window size up to 100 KB and 6.4MB at least for getting 400 Mbps throughput, respectively. A detailed explanation for these values is shown in appendix C. These figures indicate that the connection with RTT 2ms grew its congestion window size enough. On the other hand, the congestion window size of the connection with RTT 128ms was not always enough. This is the most important cause of the unfairness and insufficient performance of the connection with larger RTT.

Next, we focus on K of CUBIC TCP, which is time of congestion window recovery. Transitions of K are also shown in Figure 3 and Figure 4. In most time, K of RTT 128ms connection is larger than that of RTT 2ms connection. This is the main reason of deficient congestion window size of the connection with RTT 128ms. The reason why larger K prevents congestion window size from growing enough will be described in the next subsection.

3.2 Cause of unfairness

In this subsection, we explain why different Ks result in unfair performance.

In case of a usual network, which is constructed of routers using drop tail, all connections encounter congestion and a packet loss at the same time. This is called "global synchronization".

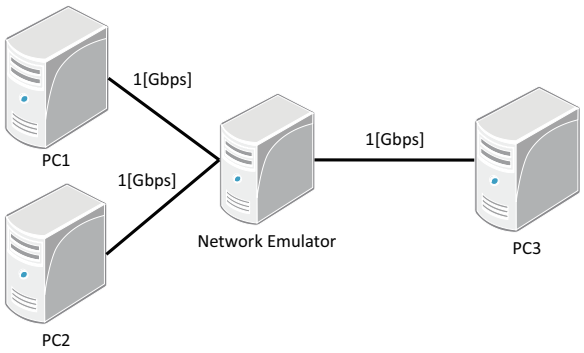

Figure 1: Experimental Network

Table 1: Computer specifications

CPU	Intel Celron CPU G530, 2.40GHz
Memory	2 [GB]
OS	(PC1, PC2) Linux 2.6.32.27 (PC3) Linux 2.6.35.6 (Network Emulator) FreeBSD 8.2
NIC	Intel PRO/1000 GT Desktop Adapter

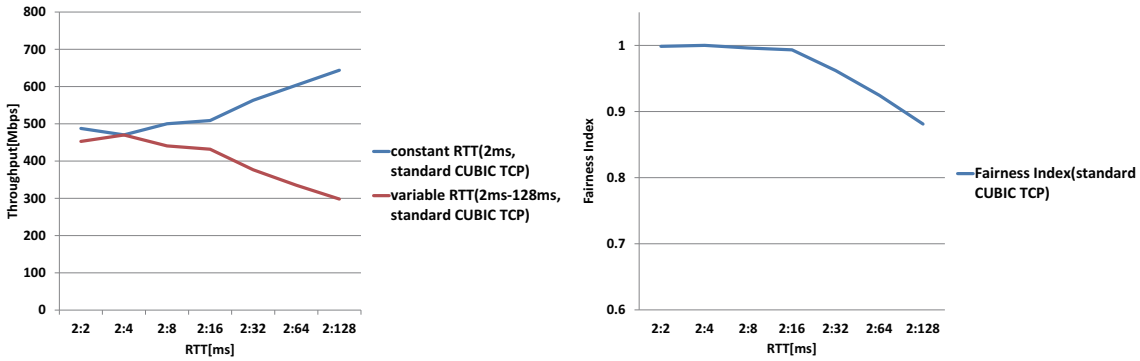

Figure 2: Throughput and fairness (standard CUBIC TCP)

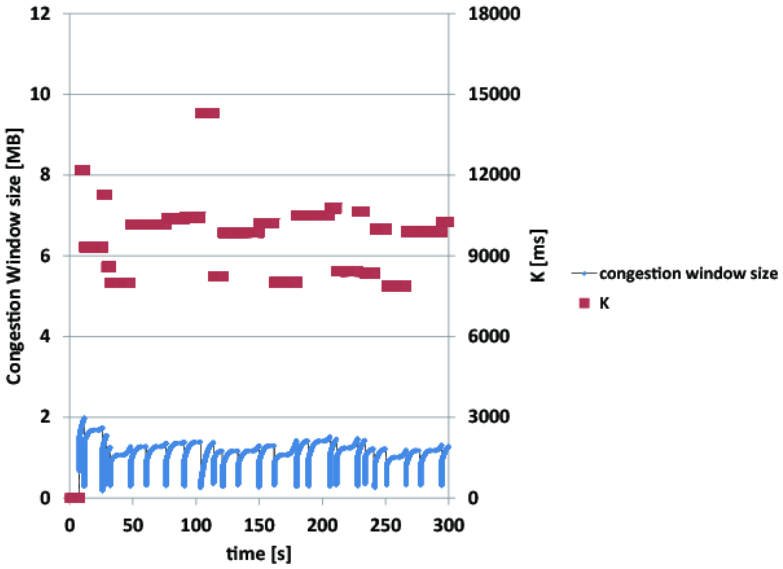

Figure 3: Transition of *cwnd* and K (RTT=2ms)

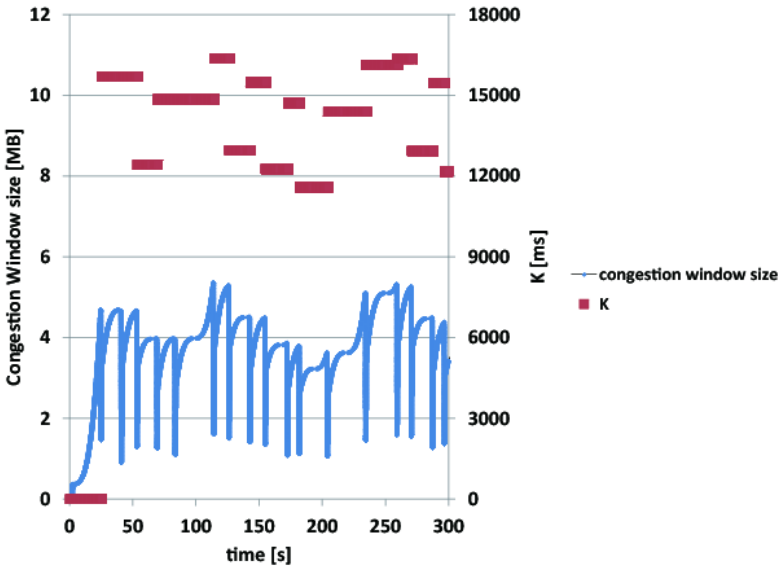

Figure 4: Transition of *cwnd* and K (RTT=128ms)

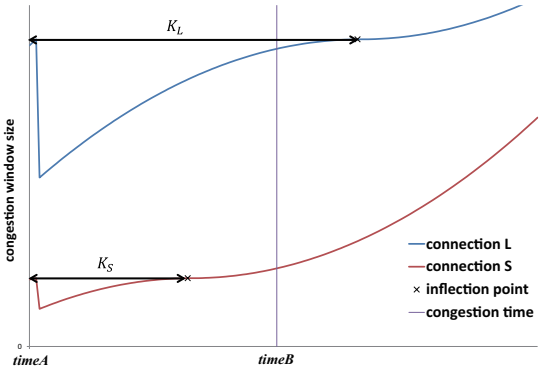

Figure 5: CUBIC TCP behavior

Figure 5 illustrates transitions of congestion window size of two connections with different Ks. We call the connections with larger and smaller K "connection L" and "connection S", respectively. K_L and K_S in the figure are "K of connection L" and "K of connection S", respectively. Congestion and a packet loss occurred at $timeA$. Each connection sets its K according to its W_{max}, as described in section 2.2. Therefore, connection L and connection S set larger and smaller K, respectively.

The next congestion occurred at $timeB$. It was the time when sum of the both congestion window sizes reached the sum at $timeA$. $TimeB$ was between K_L and K_S. As a result, connection L reached the next congestion before $timeA+K_L$ and its congestion window size at $timeB$ is smaller than W_{max} of L at $timeA$. In other words, congestion window size of connection L was decreased. On the contrary, connection S encountered the next congestion after $timeA + K_S$. This indicates that congestion window size of connection S was increased. From the both connections' behavior, we can say that congestion window size of a larger RTT connection will be decreased as congestion avoidance phases will be repeated. That is to say, a connection with larger RTT requires a larger congestion window size but the connection cannot keep larger congestion window size because larger congestion window size results in larger K and decrease of congestion window size.

3.3 Effect of K

In order to discuss effect of K on performance, we measured performance of a connection with large RTT and small K. The experimental results are shown in Figure 6. In the experiment, two connections were established. One was connection with standard CUBIC TCP and its RTT was always set 2ms. It is written as "constant RTT (2ms, standard CUBIC TCP)" in the figure. The other was connection with the modified CUBIC TCP. Its K was always set 1000ms and its RTT ranged from 2ms to 128ms. The constant K (1000ms) was much smaller than that of standard CUBIC TCP. It is written as "variable RTT (2ms-128ms, K=1000)". The figure indicates that the connection with large RTT and small K significantly outperformed that with small RTT. From the figure, we can say that adjusting K has large impact on CUBIC TCP performance when connections share a network.

3.4 RTT Fairness of CUBIC TCP with constant K

In this section, we have shown that RTT fairness of CUBIC TCP has not been sufficient and the main reason of unfairness has been K of CUBIC TCP. In this subsection, we discuss RTT fairness with the same Ks (K=5000ms).

Setting K constant can be considered to be the most straight-forward and simplest method for avoiding the unfairness caused by difference of K. We measured performances obtained by CUBIC TCP with constant K. The experimental results are shown in Figure 7. These graphs show that

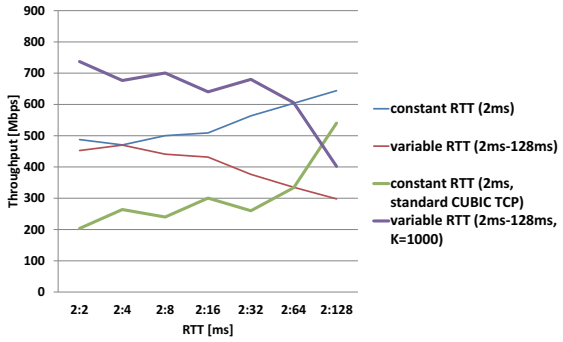

Figure 6: Throughput (large RTT and small K)

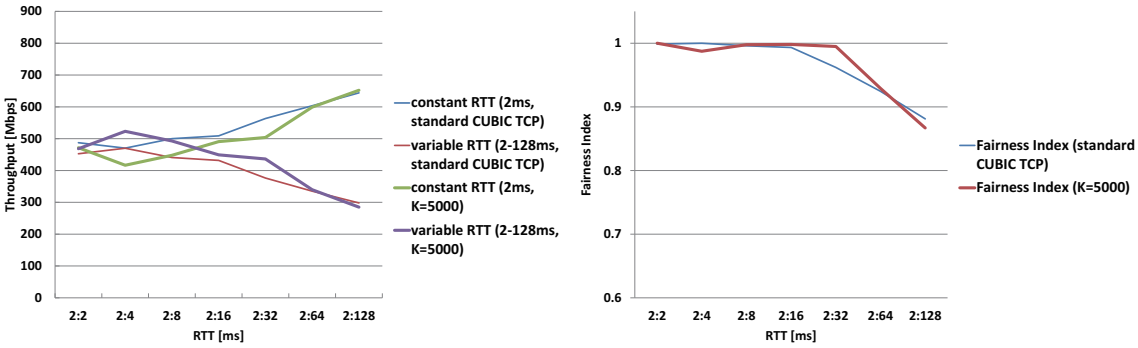

Figure 7: Throughput and fairness (K=5000ms)

setting K constant improves RTT fairness of CUBIC TCP. However, their performances have some difference and RTT fairness with constant K is not sufficient.

4 Proposal

4.1 Adjusting K

In this subsection, we propose a method for improving RTT fairness between CUBIC TCP connections. As described in section 3.2, large K of large RTT connections is the main reason of unfairness. For solving this issue, we propose to adjust K according to RTT, with which K is adjusted with the following formula.

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \times \frac{1}{\sqrt[3]{RTT}} \tag{6}$$

$\frac{1}{\sqrt[3]{RTT}}$ in formula (6) is determined based on the following discussion. An obtained throughput is limited to $WindowSize/RTT$ by window size (see also appendix C for a detailed explanation). Thus, fairness is achieved when the all connections have the same $WindowSize/RTT$. In this case, window size and W_{max} are proportional to RTT. So, we have made this situation the target of our proposed method, and we have concluded that our method should moves congestion window size and W_{max} close to this situation when these are not. If W_{max} is exactly proportional to RTT, performance fairness is achieved and K is the same for all connections with the proposed method. In theory, the next congestion will occur after K and the all connections encounter packet losses at their flexion point, then all connections keep their W_{max} with the same K. However, this balance

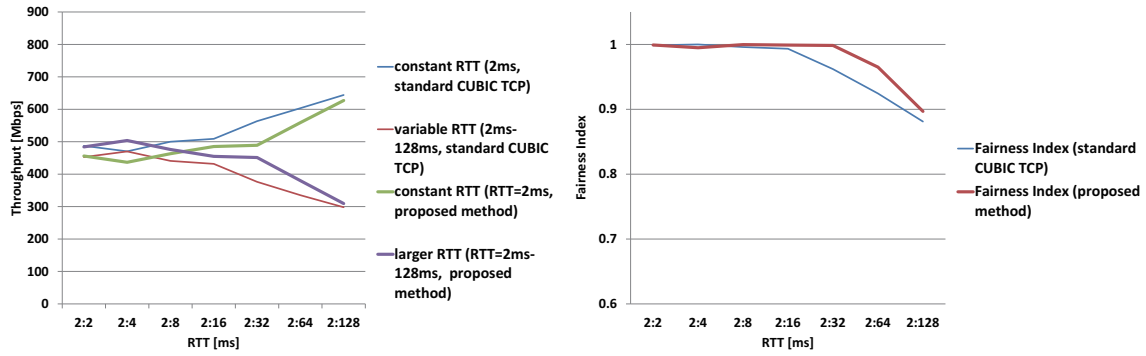

Figure 8: Throughput and fairness (proposed method)

is not stable. As described in section 3.4, RTT fairness is not achieved with constant K. Thus, we can say that adjusting window size and W_{max} close to the target situation is required. With the proposed method using formula (6), a larger RTT connection with insufficient W_{max} gains smaller K than other connections. Therefore, this connection can recover its congestion window faster. As described in section 3.2, faster recovery results in increase of congestion window size. Similarly, a connection with excess window size gains larger K then its window size is decreased. That is to say, a connection with smaller W_{max}, K, and performance than the target situation increases their values and performance, and a connection with larger values decreases their values and performance. Then, this mechanism leads a situation into a fair one.

4.2 Evaluation

We evaluated our proposed method with the network in Figure 1. The experimental setup is the same as that in section 3.1. CUBIC TCP implementations were modified according to the proposed method. Then, Ks in the implementations were determined with the formula (6).

The obtained performances are shown in Figure 8. This figure indicates that RTT fairness is significantly improved by the proposed method. The congestion window size and K of the experiment with RTT 2ms and 128ms are shown in Figure 9 and Figure 10, respectively. From these figures, we can see that K of the larger RTT connection is smaller than that of the smaller RTT connection. Then, the congestion window size of the larger RTT connection kept larger than that of the smaller RTT connection. These figures demonstrate that the proposed method can adjust K suitably and improves RTT fairness.

5 Discussion

5.1 Exponent of RTT in K

In the proposed method, we have used third root of RTT. As mentioned in section 4.1, third root of RTT enables controlled throughput in theory and our evaluation have demonstrated that the proposed method have improved fairness. However, the complete fairness has not obtained with practical systems. Further fairness can be achieved by tuning exponent of RTT in K. If you got deficient performance with larger RTT connection, you should use $K = \sqrt[3]{\frac{W_{max}\beta}{C}} \times \frac{1}{\sqrt[3-\alpha]{RTT}}$, $(\alpha > 0)$ instead of $K = \sqrt[3]{\frac{W_{max}\beta}{C}} \times \frac{1}{\sqrt[3]{RTT}}$. As you increase α, performance of a connection with larger RTT increases.

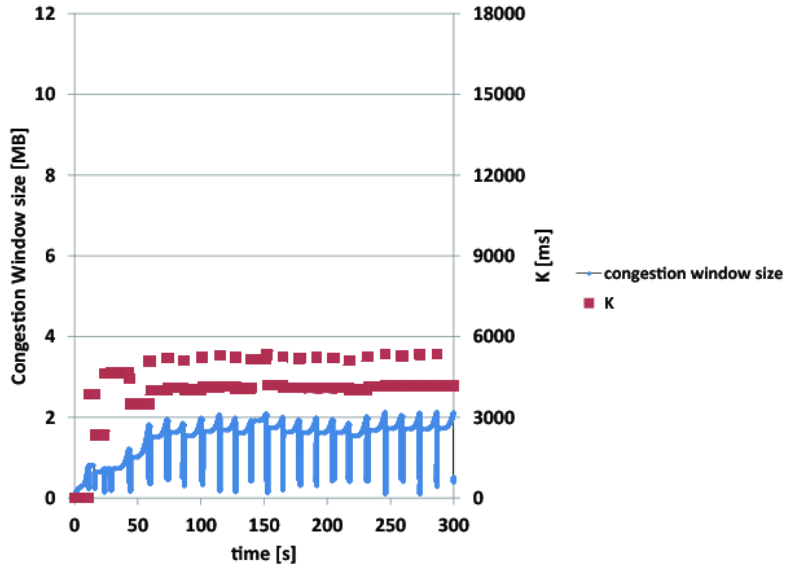

Figure 9: Transition of *cwnd* and K (RTT=2ms, proposed method)

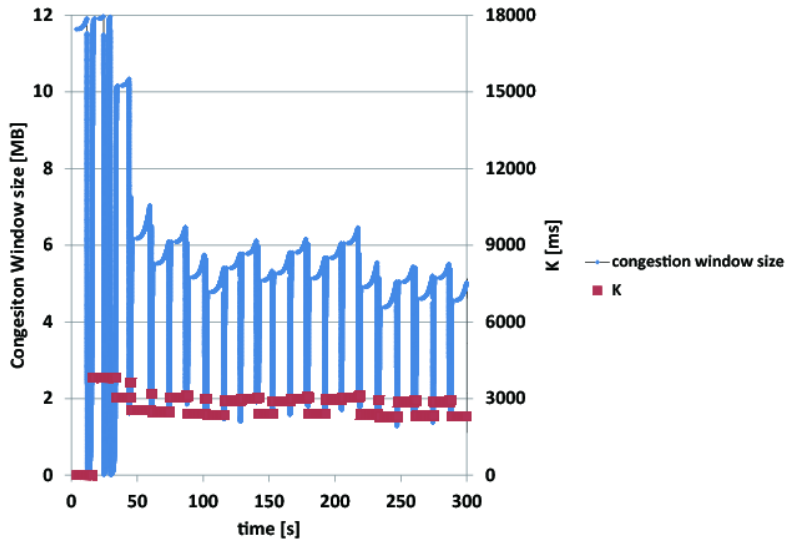

Figure 10: Transition of *cwnd* and K (RTT=128ms, proposed method)

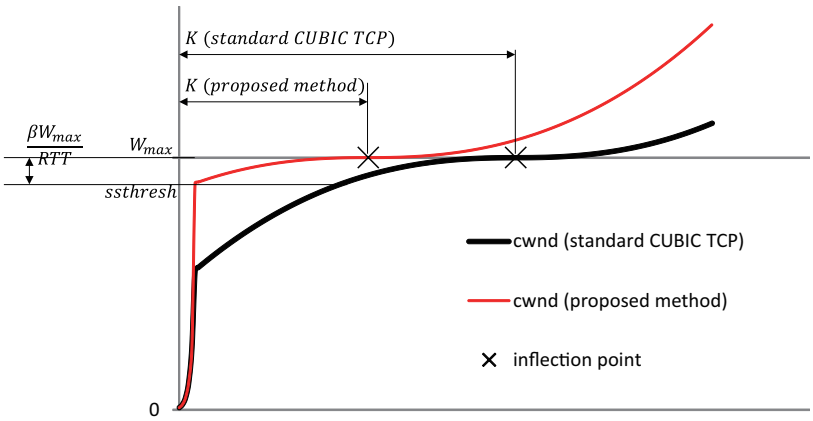

Figure 11: *ssthresh* (proposed method)

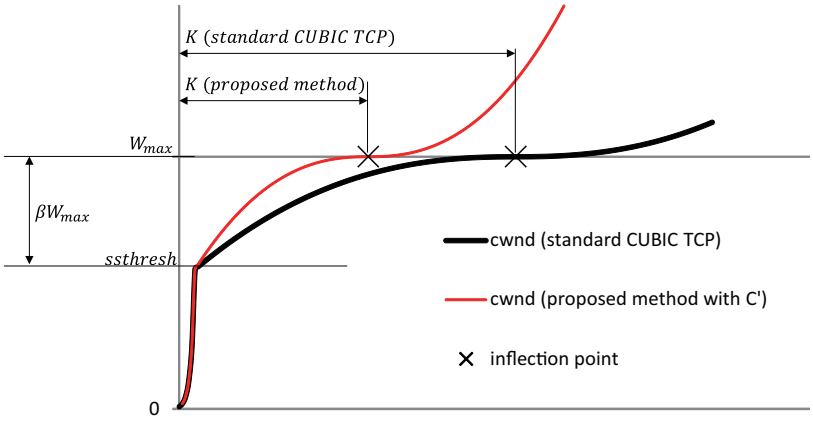

Figure 12: *ssthresh* (proposed method with C')

5.2 C in the proposed method

In the original CUBIC TCP, K is determined using C and β as shown in section 2.2. *ssthresh* (slow start threshold), which is a border of slow start phase and congestion avoidance phase, is set $W_{max}(1-\beta)$. In many cases, β is set 0.2 and *ssthresh* is $0.8 \times W_{max}$. With the proposed method, K is modified as shown in section 4.1. This modification changes *ssthresh* like Figure 11. This change can be avoided by using C' instead of C like Figure 12.

$$cwnd = C' \times (t-K)^3 \quad (7)$$

$$C' = C \times RTT \quad (8)$$

Because throughput is effected by congestion window size, performance using the formula (6) and the formula (7) are different. However, we focus on controlling performance in this work. As mentioned above, performance can be controlled by adjusting K, i.e. giving smaller and larger K result in increase and decrease of throughput, respectively. So, we have discussed on adjusting K with standard setup, with which C is 0.4 and β is 0.2. The evaluation results using formula (7) and (8) are shown in appendix D.

6 Conclusion

In this paper, we presented RTT fairness evaluation of CUBIC TCP and demonstrated its insufficient fairness. After the evaluation, we have proposed a method for improving RTT fairness of CUBIC TCP by adjusting K according to RTT. Our evaluation has demonstrated that our proposed method has been able to improve RTT fairness of CUBIC TCP.

We plan to evaluate fairness between the proposed method and other TCP congestion control algorithms.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 24300034, 25280022, 26730040.

References

[1] W. Stevens, "TCP Congestion Control," IETF RFC 2581, 1999.

[2] Dina Katabi, Mark Handley, and Charlie Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," SIGCOMM '02. Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 89-102, 2002.

[3] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand, "Analysis and Comparison of TCP Reno and Vegas," INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, March 1999.

[4] L. Xu, K. Harfoush and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," Proc. INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, March 2004.

[5] Injong Rhee, and Lisong Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," Proc. Workshop on Protocols for Fast Long Distance Networks, 2005.

[6] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks," Proceedings 25th Conference on Computer Communications (InfoCom 2006), April 2006.

[7] The Network Simulator-ns-2, `http://www.isi.edu/nsnam/ns/`

[8] SUZUKAWA Ryuji, and ADACHI Naotoshi, "Enhancing Fairness of Bandwidth Sharing for CUBIC-TCP," IEICE Technical Report NS2008-108, 2008 (in Japanese).

[9] Sally Floyd, "Connections with multiple congested gateways in packet-switched networks part 1: one-way traffic," SIGCOMM Comput. Commun. Rev., Vol. 21, No. 5, pp. 30-47, Oct. 1991.

[10] S. Floyd and V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways," Computer Communication Review (ACM), Vol. 21, No. 2, 1991.

[11] Thomas R. Henderson, Emile Sahouria, Steven Mccanne, and Y H. Katz, "On Improving the Fairness of TCP Congestion Avoidance," Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE, vol.1, pp. 539-544, 1998.

[12] G. Marfia, C. E. Palazzi, G. Pau, M. Gerla M. Y. Sanadidi, and M. Roccetti, "Balancing Video on Demand Flows over Links with Heterogeneous Delays," Proceedings of the 3rd international conference on Mobile multimedia communications, pp. 1-6, 2007.

[13] Kazumine OGURA, Yohei NEMOTO, Zhou SU, and Jiro KATTO, "A New TCP Congestion Control Supporting RTT-Fairness," IEICE TRANSACTIONS on Information and Systems Vol.E95-D No.2 pp.523-531, 2012.

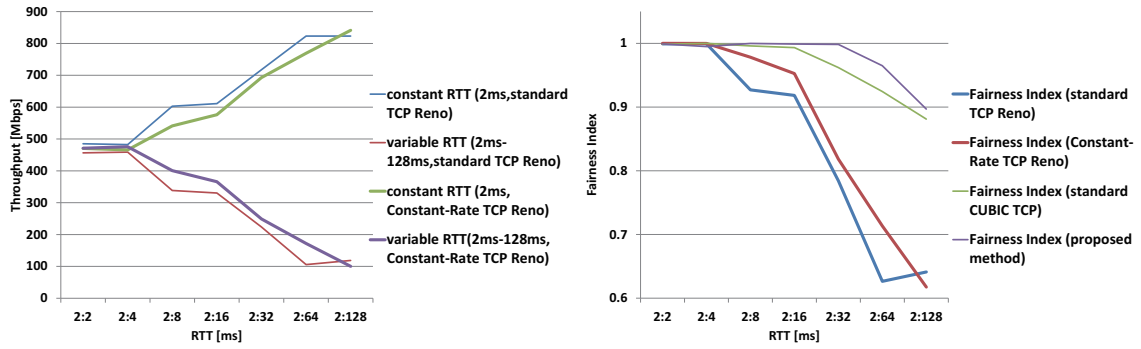

Figure 13: Throughput and fairness (Constant-Rate)

[14] Hayato Itsumi, and Miki Yamamoto, "Improving Fairness between CUBIC and Compound TCP," IEICE Technical Report NS2010-160, pp. 103-108, 2010 (in Japanese).

[15] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, V.1 N.4, pp. 397-413, August 1993.

[16] Ryo Oura and Saneyasu Yamaguchi, "Fairness Comparisons among Modern TCP Implementations," Proceedings of the 26th International Conference on Advanced Information Networking and Applications Workshops, pp. 909-914, 2012.

[17] Tomoki Kozu, Yuria Akiyama and Saneyasu Yamaguchi, "Improving RTT Fairness on CUBIC TCP," The First International Symposium on Computing and Networking, 2013.

[18] netperf homepage, `http://www.netperf.org/netperf/`

[19] Luigi Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," ACM SIGCOMM Computer Communication Review, Volume 27 Issue 1, pp. 31-41, Jan. 1997.

[20] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," Computer Networks and ISDN Systems, Volume 17, Issue 1, pp. 1-14, 1989.

[21] Kevin R. Fall, W. Richard Stevens, "TCP/IP Illustrated, Volume 1 Second Edition," Addison-Wesley, pp. 729-730, 2012.

A Comparing with Constant-Rate

In [9], [10], and [11], it is suggested that Constant-Rate window increase improves RTT fairness. This method is not for modern fast TCP, such as CUBIC TCP, but for classical TCP, such as TCP Reno. Thus, exact comparison cannot be conducted but we introduce evaluation results as a guide. We have modified TCP Reno implementation according to Constant-Rate [9]. It has a parameter a as described in section 2.3. Then, we used $a = 4$, which was introduced in the original work [9]. Figure 13 shows performance and fairness of standard TCP Reno, Constant-Rate TCP Reno, and our proposed method. Comparing the figure with Figure 2 and Figure 8, we can see that RTT fairness of TCP Reno is much worse than that of CUBIC TCP. Figure 13 indicates that Constant-Rate method can improve RTT fairness of TCP Reno. However, fairness obtained by Constant-Rate with default setting is not comparable with those of CUBIC TCP and the proposed method.

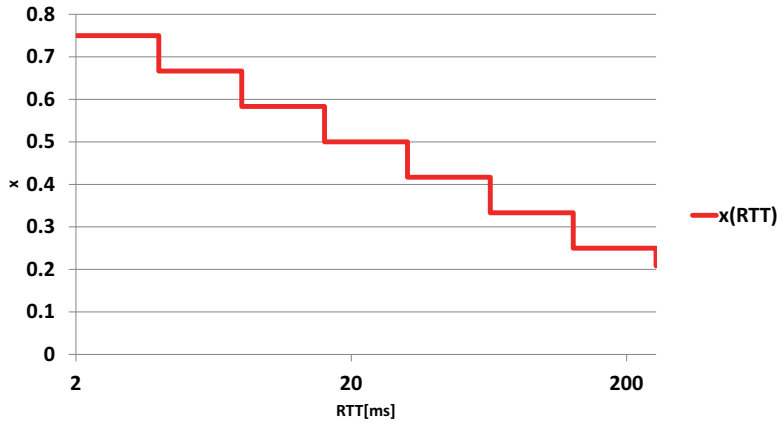

Figure 14: x(RTT)

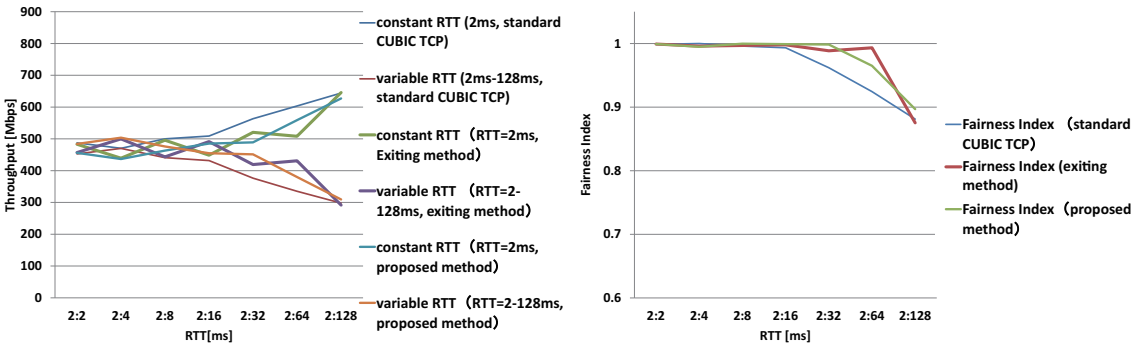

Figure 15: Throughput and fairness (existing method)

B Comparison with the existing method

In [17], a method for improving RTT fairness based on heuristic optimization was proposed. In this section, we present comparison between this existing method and the proposed method in this paper. In the existing method, K is adjusted using the following formula

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \times x(RTT) \tag{9}$$

$x(RTT)$ is a function of RTT. $x(RTT)$ monotonically decreases as RTT increases. $x(RTT)$ is shown in Figure 14. Comparison of the existing method and the proposed method are shown in Figure 15. These graphs indicate that the proposed method can provide fairness comparable to the existing method without introducing an additional control parameter. In addition, we can see that fairness of the proposed method is stable.

C Bandwidth-Delay Product

In this section, we explain the reason why obtained throughput is less than $WindowSize/RTT$. If window size is not enough, the transmission is executed like Figure 16. A sender can send window size of data at largest without receiving an ACK packet. Thus, sending is suspended until receiving the

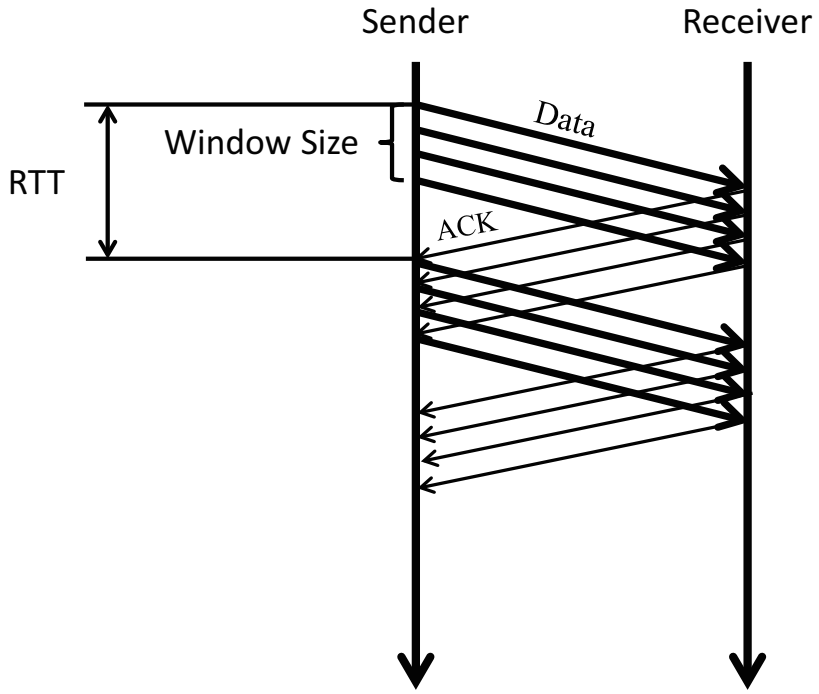

Figure 16: Bandwidth-Delay Product

next ACK. After receiving ACK, the sender can send window size of data and sending is suspended until receiving ACK. The next ACK will be received after RTT. In this case, window size of data can be sent every RTT. Hence, the obtained throughput is $WindowSize/RTT$.

In the case of RTT 2ms, 100 KB window size is required for getting 400 Mbps because of the following formula.

$$100 \times 10^3[bytes] \times 8[bits/byte]/(2 \times 10^{-3}[sec]) = 400 \times 10^6[bps] = 400[Mbps] \quad (10)$$

Similarly, 6.4MB of window size is required for getting 400 Mbps with RTT 128ms because of the following formula.

$$6.4 \times 10^6[bytes] \times 8[bits/byte]/(128 \times 10^{-3}[sec]) = 400 \times 10^6[bps] = 400[Mbps] \quad (11)$$

D Performance Evaluation using C'

In this section, we introduce performance evaluation of the proposed method using C', formula (7) and formula (8). The performance is shown in Figure 17. The figure has demonstrated that using C' provides better fairness in some cases but the fairness with C' is not stable. As shown in Figure 12, using C' increases congestion window size more aggressively with large RTT. Thus, we can expect that this method may afford more network bandwidth for connections with large RTT but it may also cause increase of congestions without tuning it. In this work, we applied standard C setting, which is C=0.4 [5]. With our approach, tuning C is not required.

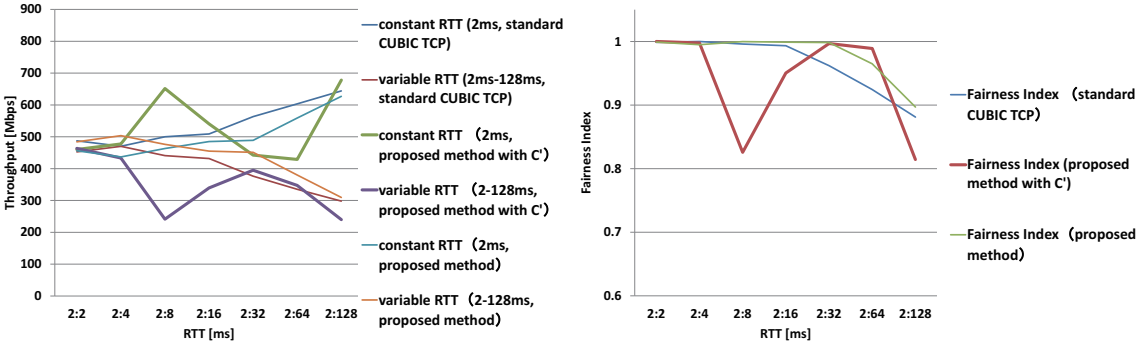

Figure 17: Throughput and fairness (proposed method using C')