Performance Evaluation of TCP Variants with Packet Reordering

Yutaka Fukuda
Information Science Center,
Kyushu Institute of Technology
1-1 Sensui-cho, Tobata,
Kitakyushu 804-8550, Japan
Email: fukuda@isc.kyutech.ac.jp

Daiki Nobayashi
Faculty of Engineering,
Kyushu Institute of Technology
1-1 Sensui-cho, Tobata-ku,
Kitakyushu 804-8550, Japan
Email: nova@ecs.kyutech.ac.jp

Takeshi Ikenaga
Faculty of Engineering,
Kyushu Institute of Technology
1-1 Sensui-cho, Tobata-ku,
Kitakyushu 804-8550, Japan
Email: ike@ecs.kyutech.ac.jp

***Abstract*—Packet reordering in a short fixed period is considered in this paper. We believe that data will be transmitted dynamically and in parallel in the near future, which will require more frequent periodic packet reordering. This in turn will lead to unnecessary retransmissions and throughput degradation to the TCP (Transmission Control Protocol). There has been much research to improve the TCP performance with packet reordering, but the considered reorder intervals have been based on measurements on the existing Internet, and the short, fixed reorder intervals caused by the flexible transmission schemes have not been studied sufficiently. Therefore, in this paper, we vary the fixed reorder interval from within the Round-Trip Time (RTT) to over the RTT, and evaluate the communication performance of TCP NewReno and Cubic. From the simulation results, we show that the performance of TCP Cubic is highly affected by the packet reordering.**

***Keywords*–TCP; Cubic; NewReno; packet reordering.**

I. Introduction

Packet reordering is out-of-order packet arrival at the receiver. Namely, the destination receives a packet in a different order from its sending one. Although there are several causes, one of the main reasons for packet reordering is that some packets take different paths because of route oscillations over the network layer. TCP (Transmission Control Protocol) addresses this issue by performing sequence control and sends back duplicate ACKs (Acknowledgments) to report the packet gap. However, the TCP sender assumes packet loss after receiving three duplicate ACKs from the receiver, and decreases the transmission rate, which causes a substantial throughput degradation. To avoid this performance deterioration, various methods have been proposed in previous studies [4]–[10].

In contrast, recent development trends in network technologies, in addition to the usual communication performance metrics, such as fault tolerance and delay, have included multipath routing for a variety factors such as power savings. Furthermore, research and development are also being conducted extensively on maintaining communication while simultaneously using different types of media on mobile devices, such as LTE and IEEE 802.11 wireless LAN. Networks in the future are therefore expected to move further toward dynamic multipath routing, which will generate more packet reordering along the path and change the reordering pattern. Especially, packet reordering may occur continuously at regular intervals because of, for example, simultaneous multipath use in order to achieve high-speed and efficient communication. However, to the best of the authors' knowledge, the impact of packet reordering occurring at short, regular intervals on the performance of TCP communication has not been previously studied.

Therefore, the objective of this paper is to study the impact of packet reordering occurring at regular intervals on the performance of TCP communication. Specifically, this paper examines the communication performance when frequent packet reordering occurs continuously within and over the Round-Trip Time (RTT) using NewReno and Cubic as TCP congestion control algorithms, and shows the requirement to adapt the packet reordering from simulation results.

The remainder of the paper is organized as follows. Section 2 discusses the related studies. Section 3 describes the simulation model and the packet reordering schemes used in this paper. Section 4 presents the communication performances of TCP NewReno and Cubic when packet reordering occurs, and Section 5 summarizes our conclusions.

II. Related Work

Many studies [1]–[3] show actual measurements of packet reordering occurrence on the Internet. When a packet arrives out of order, the TCP receiver sends out a duplicate ACK on the missing packet. If at least three duplicate ACKs arrive, then the TCP sender interprets them as packet loss, and retransmits the packet indicated by the duplicate ACK. Then, fast recovery is triggered with fast retransmit, and congestion window *cwnd* is set to half its value, which causes significant performance degradation. To date, there have been many studies [5]–[10] aimed at solving this problem.

Proposed solutions for packet reordering are classified as follows: (1) dynamically control the number of duplicate

ACKs to enter fast recovery [4], [6], [7]; (2) detect the occurrence of packet reordering by the TCP timestamp option and restore the reduced *cwnd* and *ssthresh* to their original values [5]; and (3) detect packet loss not by duplicate ACKs but by using timers [8]. The advantages and disadvantages of these proposed solutions, as well as their evaluation through a simulation study, are provided in detail in Leung et al. [9]. In addition, Feng et al. [10] evaluated the performance of the proposed solutions for packet reordering in a high-speed communication environment.

To evaluate the performance of the proposed solutions, these previous studies use actual measurements on the Internet. In simulations, they vary the packet delay between relay routers to invoke packet reordering. However, we believe that data packets will be transmitted more dynamically and in parallel to achieve effective performance on the near-future Internet, which will cause more frequent periodic packet reordering. To the best of the authors' knowledge, TCP communication performance with frequent and continuous packet reordering within and over the RTT has not been studied sufficiently.

III. Simulation Model

In this study, we use the network simulator ns-3 [11] after adding a packet reordering function. The simulation topology for the simulation is given in Figure 1. The sending terminal S sends TCP segments with a size of 1,500 bytes to the receiving terminal D. It is assumed that each TCP flow is used for greedy file transfer. Assuming concurrent use of multiple paths, packet reordering was modeled to occur at the bottleneck link between routers R_1 and R_2 with a link bandwidth of 10 Mb/s and a delay of 5 ms. In contrast, the link bandwidth of the access links between each terminal and routers R_1 and R_2 is 100 Mb/s with a delay of 15 ms, resulting in an RTT of 70 ms between the terminals. The TCP congestion control algorithms used for the study are NewReno [12] [13] and the Linux-standard Cubic [14], [15]. Simulation time is 10.2 s, and TCP starts transmission at 0.2 s after starting the simulation.

In previous studies, the setting of the packet reordering interval is based on arrival distributions obtained by actual measurements. In contrast, in this paper, we assume that reordered packets arrive predetermined interval or more. Specifically, if the buffer size of R_1 is larger than the predetermined variable RI (Reorder Interval), then the head of line packet in R_1 is moved behind by the RI packet size. Note that other packets are not reordered until transmission of the moved packet is completed.

Reordering behavior in our study is illustrated in Figure 2. First, assume that there are 11 packets in R_1, packets 10 to 20, as shown in Figure 2(a) and RI = 4. Since the buffer size is 11, which is larger than RI, and head of line packet 10 has not been reordered before, packet reordering occurs and packet 10 is moved behind by RI = 4 packets as shown in Figure 2(b). Packets are thereafter sequentially transmitted until packet 10 as shown in Figure 2(c). After packet 10 is transmitted, six packets, packets 15 to 20, still remain in R_1. Since the packet reordering condition is satisfied, the head of line packet 15 is moved behind

TABLE I. SIMULATION CONDITIONS

Segment Size	1,500 bytes
RTT	70 ms
Reorder Interval	4–100 packets (Minimum 4.8–120 ms intervals)
TCP variants	NewReno, Cubic
Simulation time	10.2 s
Simulator	ns-3

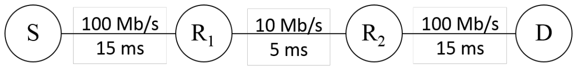

Figure 1. Simulation model.

by 4 packets as shown in Figure 2(d). Based on this scheme, packet reordering occurs and packets at every RI are moved backward if the buffer size of R_1 is larger than RI.

In the simulation, RI was set from 4 to 100 packets. Note that since the TCP segment size is 1,500 bytes and the link bandwidth between R_1 and R_2 is 10 Mb/s, the forwarding time of one packet is 1,500 bytes $\times$ 8 bits/10 Mb/s = 1.2 ms. Thus, if RI = 4, then packets are reordered at intervals of 1.2 ms $\times$ 4 = 4.8 ms. Moreover for RI = 4, since the RTT between terminals is 70 ms, reordering can occur $70/4.8 \simeq 14$ times at most within one RTT. The simulation conditions given above are shown in Table I.

IV. Simulation Results and Discussion

The simulation results are presented in this section. We set the buffer size of relay router R_1 sufficiently in order not to lose packets, and the impact of packet reordering alone on the performance of TCP communication is studied. Next, we show how packet reordering impacts communication performance both when RI is within and over RTT.

A. Fundamental Characteristics

As a preliminary step, throughput was measured when packet reordering does not occur under the same simulation conditions. A throughput of 9.54455 Mb/s was confirmed for both Cubic and NewReno. Figure 3 shows the normalized throughput (= throughput with packet reordering / throughput without packet reordering) when RI is varied from 4 to 100. Figure 4 shows the number of fast recovery events.

From Figure 3, normalized throughput of both NewReno and Cubic are not strictly increasing, since the packet reordering pattern in each RI is different. However, the occurrence of packet reordering caused throughput to decrease by a maximum of approximately 50 % for NewReno and a maximum of approximately 65 % for Cubic. The throughput performance of Cubic is lower than that of NewReno except for RI = 30. In addition, Figure 4 also shows that packet reordering causes fast recovery for RI below 100. Moreover, a shorter RI corresponds to a larger number of fast recovery events. Note that for RI = 100, Figure 4 shows that there is no fast recovery, and since packet reordering does not occur at R_1, the normalized

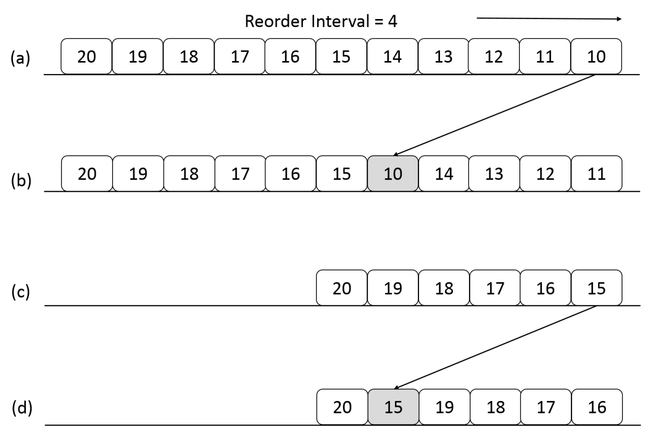
Reorder Interval = 4
(a) 20 19 18 17 16 15 14 13 12 11 10
(b) 20 19 18 17 16 15 10 14 13 12 11
(c) 20 19 18 17 16 15
(d) 20 15 19 18 17 16

Figure 2. Packet reordering.

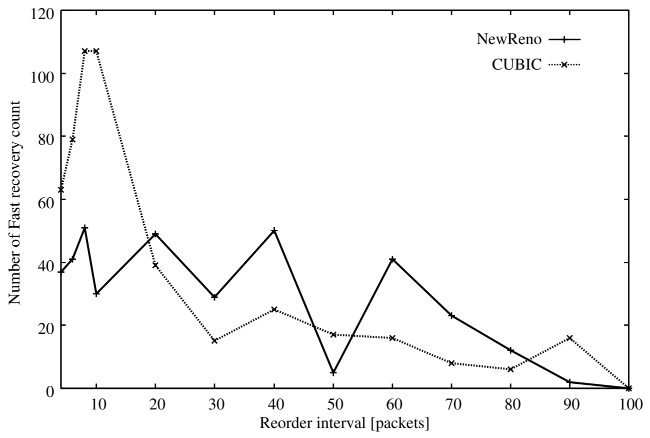

Figure 4. Fast recovery count.

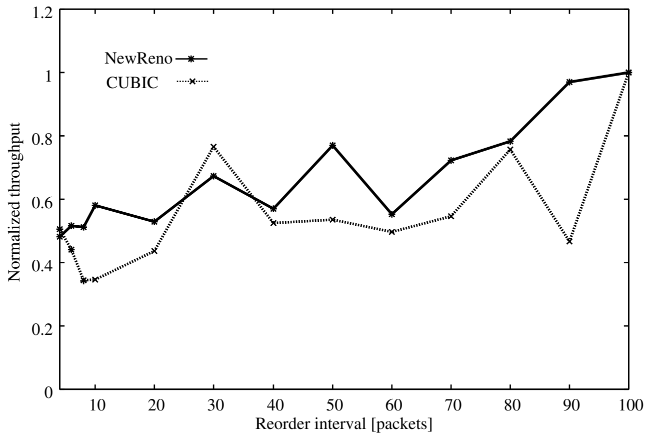

Figure 3. Normalized throughput [Mb/s].

throughputs of both TCP variants are equal to 1. Furthermore, a comparison of Figures 3 and 4 reveals that, for Cubic, throughput performance deteriorates regardless of the number of fast recovery events. We can therefore consider that both frequent occurrence of packet reordering (small RI) and occurrence of packet reordering with a large *cwnd* (large RI) affect throughput performance for Cubic. Thus, in the next section, we examine the behavior of NewReno and Cubic for different RIs.

B. For RI shorter than RTT

We first consider the case that RI is 8 (1.2 ms × 8 = 9.6 ms), where the interval for packet reordering is shorter than RTT. The *cwnd* of NewReno and Cubic are shown in Figures 5 and 6, respectively. *cwnd* increases even after fast recovery due to packet reordering in TCP NewReno, whereas *cwnd* fluctuates at a low range in TCP Cubic, as illustrated by Figures 5 and 6. To show their behaviors in more detail, the time range between 2 and 2.3 s is shown in Figures 7 and 8. Figure 7 shows that, after fast recovery is invoked because of packet reordering, NewReno sets *cwnd* to half its value before fast recovery and continues communication in congestion avoidance mode. In contrast, Cubic repeatedly decreases *cwnd* to its initial value of 2 after fast recovery, as demonstrated in Figure 8.

To understand Cubic's behavior, *cwnd* from 0 to 2 s is considered in Figure 9. As shown in the figure, in Cubic, the size of *cwnd* after fast recovery is gradually reduced until it falls to the minimum value of 2. This is because Cubic updates *cwnd* and *ssthresh* according to the following equation [15].

$$cwnd = ssthresh = \max\left(\frac{cwnd \times \beta}{BICTCP\_BETA\_SCALE}, 2\right) \quad (1)$$

We use $\beta = 819$ and $BICTCP\_BETA\_SCALE = 1024$ for the simulation, thus multiplicative decrease factor is 0.8. Based on (1), *cwnd* with successive fast recovery gradually becomes smaller and eventually converges to the minimum value of 2, which significantly reduces throughput. These results show that, for frequent packet reordering, the throughput performance of TCP Cubic is highly affected by the packet reordering because Cubic updates *cwnd* based on (1).

C. For RI longer than RTT

In this section, we consider the case that RI is 80 (1.2 ms × 80 = 96 ms), where the interval for packet reordering is longer than RTT. The *cwnd* for each congestion control algorithm is shown in Figures 10 and 11. Figure 10 shows that NewReno can send packets in congestion avoidance mode even after the occurrence of packet reordering. In contrast, as shown in Figure 11, increase of *cwnd* in Cubic occurs intermittently in the range between 2 and 8 s. In Cubic, *cwnd* can be increased after receiving the number of ACKs given by *cnt*. The count variable *cnt* is calculated according to the following equation [15].

$$\begin{aligned} &if \quad (cwnd < W(t+RTT)) \\ &\quad cnt = \tfrac{cwnd}{W(t+RTT)-cwnd} && (2) \\ &else \\ &\quad cnt = 100 \times cwnd && (3) \end{aligned}$$

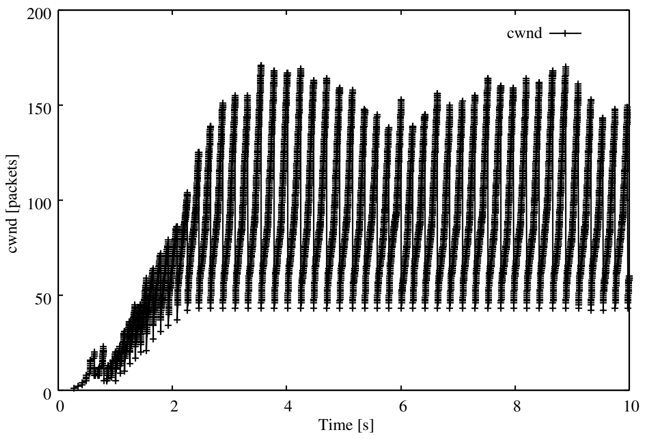

Figure 5. TCP NewReno (RI = 8).

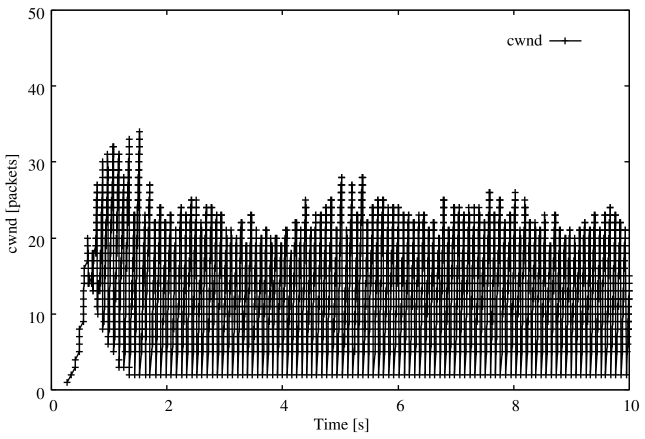

Figure 6. TCP Cubic (RI = 8).

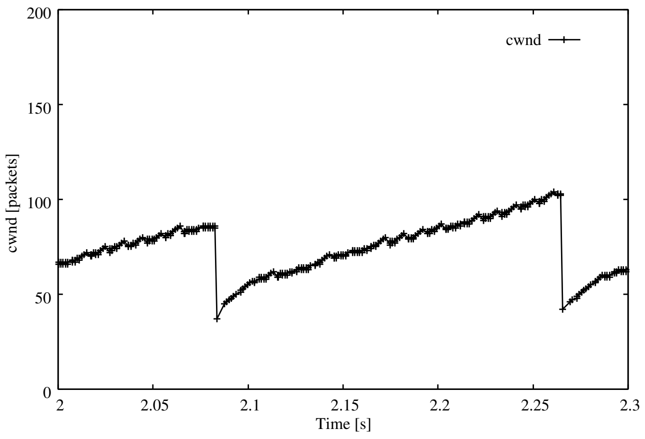

Figure 7. TCP NewReno (RI = 8; 2.0–2.3 s).

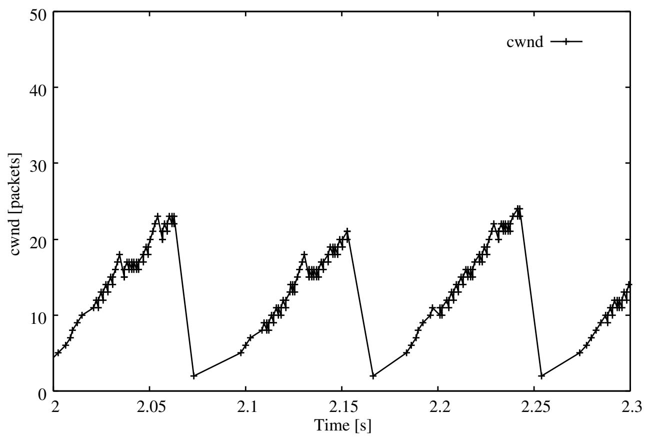

Figure 8. TCP Cubic (RI = 8; 2.0–2.3 s).

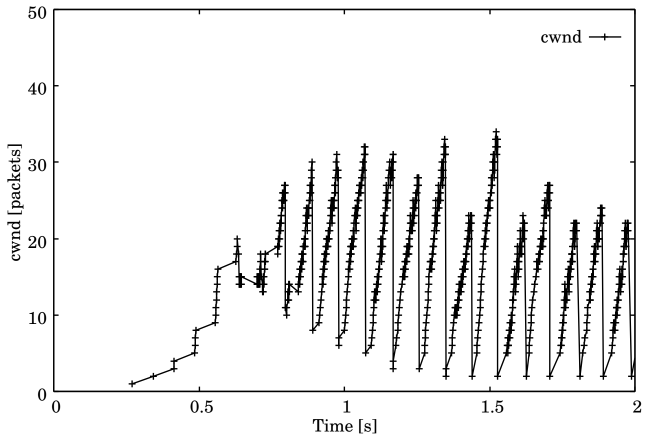

Figure 9. TCP Cubic (RI = 8; 0–2.0 s).

Based on (2) and (3), *cnt* becomes large if *cwnd* becomes somewhat large and the difference between $W(t+\text{RTT})$ and *cwnd* is small. In other words, a large number of received ACKs is required to increase *cwnd*. For the case of RI = 80 considered in this section, the value of *cnt* is relatively large because packet reordering occurs when *cwnd* has increased to some extent. Thus, the *cwnd* increase is intermittent for a long period of time, whereas NewReno can increase *cwnd* with normal congestion avoidance mode. In order to improve the performance of TCP Cubic when packet reordering occurs with large *cwnd*, different packet loss detection aproach is required.

From these simulation results, we have shown that the throughput performance of Cubic may deteriorate considerably when packet reordering occurs at intervals longer than RTT and *cwnd* is large, even if the number of packet reordering events per unit time is small. Furthermore, results of Sections 4.2 and 4.3 show that there is a need for packet loss detection that does not rely on duplicate ACKs and a transmission method with higher tolerance to packet reordering, whereas the cause for the lower throughput in Cubic varies depending on how packet reordering occurs.

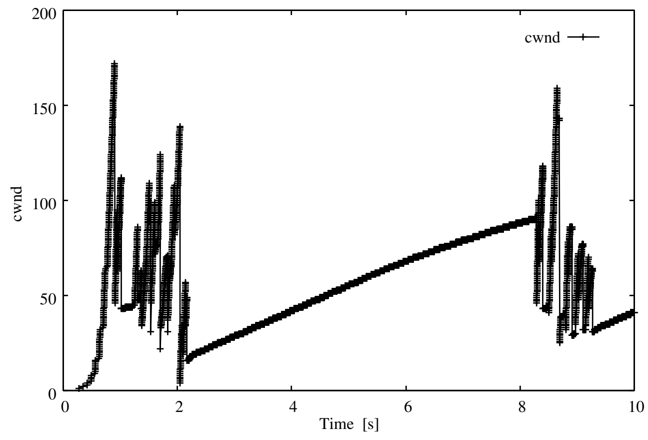

Figure 10. TCP NewReno (RI = 80).

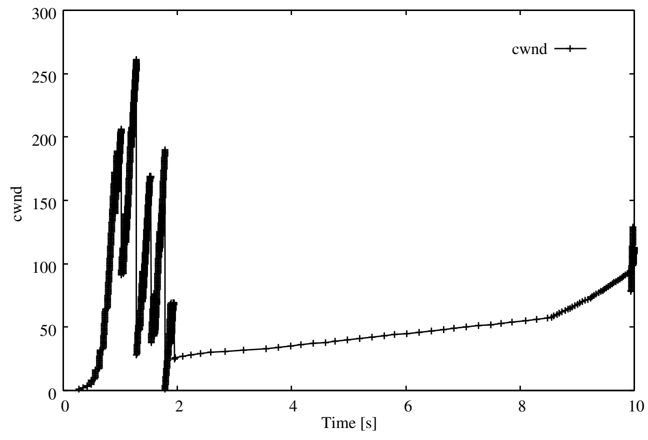

Figure 11. TCP Cubic (RI = 80).

V. Conclusion

In this paper, we examined the impact of packet reordering occurring at very short regular intervals on the communication performances of TCP NewReno and Cubic. Implemented on ns-3, packet reordering was designed to occur at each predetermined RI interval and head of line packet move to RI packets. RI was then varied so that it was within or over RTT to study the throughput and number of fast recovery events due to packet reordering. For an RI smaller than RTT, the throughput for Cubic decreases considerably since Cubic continually reduces *cwnd* and *ssthresh* when duplicate ACKs are detected due to frequent packet reordering. For an RI larger than RTT, the throughput performance of Cubic also deteriorates substantially when packet reordering occurs with a large *cwnd* even if the number of packet reordering events per unit time is small. Specifically, in congestion avoidance mode, NewReno can increase the *cnwd* by 1 segment for each RTT, whereas Cubic cannot increase *cwnd* until the predetermined number of ACKs are received. These simulation results indicate that both a packet loss detection scheme which does not rely on duplicate ACKs and a method to maintain packet transmission in spite of packet reordering are vital. Future work includes evaluations of other TCP variants, such as Compound TCP and TCP-PR [8], which detects packet loss using timers rather than duplicate ACKs, in the same packet reordering environment as in this paper. In addition, congestion control algorithms with high tolerance to packet reordering should be considered.

Acknowledgments

This work was supported in part by a JSPS KAKENHI Grant-in-Aid for Scientific Research (C) (Research Project No. 16K00129).

References

[1] X. Zhou and P. V. Mieghem, "Reordering of IP Packets in Internet," Passive and Active Network Measurement: 5th International Workshop, PAM 2004, pp. 237-246, Antibes Juan-les-Pins, France, Apr., 2004.

[2] L. Gharai, C. Perkins, and T. Lehman, "Packet reordering, high speed networks and transport protocol performance," In Proceedings of the 13th International Conference on Computer Communications and Networks (IEEE Cat. No. 04EX969), pp. 73-78, Chicago, IL, USA, Oct. 11-14, 2004.

[3] N. M. Piratla and A. P. Jayasumana, "Metrics for packet reordering: a comparative analysis," International Journal of Communication Systems, Vol. 21, No. 1, pp. 99-113, 2008.

[4] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," ACM SIGCOMM Computer Communication Review, Vol. 32, No. 1, pp. 20-30, 2002.

[5] R. Ludwig and R. H. Katz, "The Eifel algorithm: making TCP robust against spurious retransmissions," ACM SIGCOMM Computer Communication Review, Vol. 30, No. 1, pp. 30-36, Jan. 2000.

[6] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: a reordering-robust TCP with DSACK," In Proceedings of the 11th IEEE International Conference on Network Protocols, pp. 95-106, Nov., 4-7, 2003.

[7] K. C. Leung and C. Ma, "Enhancing TCP performance to persistent packet reordering," Journal of Communications and Networks, Vol. 7, No. 3, pp. 385-393, Sept. 2005.

[8] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim, and K. Obraczka, "A new TCP for persistent packet reordering," IEEE/ACM Transaction on Networking, Vol. 14, No. 2, pp. 369-382, Apr. 2006.

[9] K. C. Leung, V. O. K. Li, and D. Yang, "An overview of packet reordering in Transmission Control Protocol (TCP): problems, solutions, and challenges," IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 4, pp. 522-535, Apr. 2007.

[10] J. Feng, Z. Ouyang, L. Xu, and B. Ramamurthy, "Packet reordering in high-speed networks and its impact on high-speed TCP variant," Computer Communications, Vol. 32, No. 1, pp. 62-68, Jan. 2009.

[11] "Network Simulator ns-3," http://www.nsnam.org/ retrieved: Mar. 2018.

[12] S. Floyd and T. Henderson,"RFC2582: the NewReno modification to TCP's fast recovery algorithm," RFC, 1999.

[13] S. Floyd, T. Henderson, and A. Gurtov, "RFC3782: the NewReno modification to TCP's fast recovery algorithm," RFC, 2004

[14] I. Rhee and L. Xu, "CUBIC: a new TCP friendly high-speed TCP variant," SIGOPS Operating System Review, Vol. 42, No. 5, pp. 64-74, Jul. 2008.

[15] B. Levasseur, M. Claypool, and R. Kinicki, "A TCP CUBIC implementation in ns-3," In Proceedings of the 2014 Workshop on ns-3 (WNS3'14), Atlanta, Georgia, USA, May 7, 2014.