Packet Reordering in High-Speed Networks and Its Impact on High-Speed TCP Variants

Jie Feng, Zhipeng Ouyang, Lisong Xu, Byrav Ramamurthy
Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, NE 68588-0115
Email :{jfeng,zouyang,xu,byrav}@cse.unl.edu

***Abstract*— Several recent Internet measurement studies show that the higher the packet sending rate, the higher the packet reordering probability. This implies that recently proposed high-speed TCP variants are more likely to experience packet reordering than regular TCP in high-speed networks, since they are designed to achieve much higher throughput than regular TCP in these networks. In this paper, we study the characteristics of packet reordering in high-speed networks, and its impact on high-speed TCP variants. In addition, we evaluate the effectiveness of the existing reordering-tolerant TCP enhancements. Our simulation results demonstrate that high-speed TCP variants perform poorly in the presence of packet reordering, and existing reordering-tolerant algorithms can significantly improve the performance of high-speed TCP variants.**

I. Introduction

In the past few years a number of high-speed TCP variants, such as HSTCP [8], STCP [13], BIC [22], and FAST [12], have been proposed to address the under-utilization problem of TCP in high-speed and long-distance networks. These high-speed TCP variants modify the congestion avoidance algorithms [2] of TCP to be more aggressive in high-speed networks, but they still use the same fast retransmit and fast recovery algorithms [2] as TCP, which enables TCP to detect and recover from packet loss earlier than the timeout period. However, it is well known [5], [18], [23] that the fast retransmit and fast recovery algorithms of TCP may misinterpret packet reordering as packet loss, and therefore TCP performs poorly in networks with severe packet reordering. Consequently, high-speed TCP variants using the same fast retransmit and fast recovery algorithms may not achieve the expected high throughput when packet reordering occurs.

Internet measurement results [7], [10] show that a small percentage of TCP traffic experiences packet reordering, and only very few reordered packets actually trigger TCP fast retransmit and fast recovery. This explains why TCP can achieve a satisfactory performance in most cases. However, several recent studies [9], [3], [20] demonstrate a strong correlation between inter-packet spacing and packet reordering. Basically, smaller inter-packet spacing may increase the probability of packet reordering. This result implies that for the same packet size, *the higher the packet sending rate, the greater the packet reordering probability*. For example, a recent DARPA SuperNet experiment [9] shows that a flow with a 1500-byte packet size experiences significantly more reordered packets when its sending rate is higher than 600Mbps. Since high-speed TCP variants achieve much higher throughput than regular TCP in high-speed networks, it is *highly likely* that they will experience more packet reordering than a regular TCP in these networks.

While there is some debate [4] about whether packet reordering is a pathological behavior of the Internet, and whether this issue should be addressed by designing a reordering-free network or by designing a reordering-tolerant TCP, this paper focuses on characterizing packet reordering phenomenon in high-speed networks, and its impact on recently proposed high-speed TCP variants. In addition we evaluate the effectiveness of the existing reordering-tolerant TCP enhancements.

The rest of this paper is organized as follows: Section II briefly reviews the cause of packet reordering and its negative impact on TCP. Section III describes a new packet-reordering generator used in our simulation. Section IV presents the characteristics of packet reordering in high-speed networks. Section V discusses the simulation results of high-speed TCP variants in networks with packet reordering. Section VI discusses the effectiveness of the existing reordering-tolerant TCP enhancement. Finally, section VII provides a conclusion.

II. Background of Packet Reordering

Packet reordering [15], [26] is a phenomenon in which packets with higher sequence numbers are received earlier than those with smaller sequence numbers. For example, a TCP sender sequentially sends four packets: P_1, P_2, P_3 and P_4. They may arrive at the TCP receiver in the following order: P_1, P_3, P_2 and P_4, where P_2 is reordered.

Packet reordering can be caused by networks due to the following two major reasons [4], [19], [23]. *First*, due to local parallelism within a packet router, which is a promising approach to build a high-speed and inexpensive packet router. *Second*, due to load balancing among multiple links. Multiple links with slightly different link delays may introduce significant packet reordering.

TCP [5] attempts to distinguish packet reordering from packet loss by using the number of dupACKs (duplicate acknowledgements). An ACK is generated by a TCP receiver to inform the TCP sender the next sequence number that it is expecting to receive. Let us consider the above example: four packets arrive at a TCP receiver in the following order: P_1, P_3, P_2 and P_4. When P_1 arrives, the receiver sends an ACK to ask the next packet which is P_2, but then P_3 arrives, which indicates that P_2 is missing. Therefore, the receiver sends an ACK for P_2 again (This ACK is a dupACK). In this example, the reordered packet P_2 causes only one dupACK.

RFC 2581 [2] suggests that a TCP sender should consider three or more dupACKs as an indication that a packet has been lost, based on the assumption that a reordered packet can trigger only one or two dupACKs. However, if a reordered packet causes three or more dupACKs, TCP misinterprets it as a lost packet. Consequently, TCP unnecessarily calls fast retransmit (referred to as *false fast retransmit* [23]) to retransmit the packet that seems to be lost, and unnecessarily calls fast recovery to reduce the TCP sending rate. Therefore, TCP performs poorly in networks with severe packet reordering.

III. Packet Reordering Models

In this section, we present a new packet-reordering model, which can be used to generate relatively realistic packet reordering patterns, and more importantly, with which we can study the impact of a specific property of packet reordering on TCP performance.

A. Limitations of Current Packet-Reordering Models

Several methods have been proposed to simulate packet reordering in previous studies. The models can be classified into two categories and both categories have their own limitations. One class of models reorders only one packet each time by swapping two packets in a router queue [5], or by using hiccup module [17]. However in practice, a block of packets instead of one packet may be reordered at the same time. The other class of models generates multiple reordered packets each time by extending the NS-2 error model to delay a configurable percentage of packets [23], or by changing the link delay periodically [14]. Although these models can generate more realistic reordered traffic, it is hard to isolate the impact of a specific property of packet reordering on TCP performance from that of other properties. IETF is currently developing metrics [16], [11] to capture the occurrences and characteristics of packet reordering in the Internet. However, these metrics are more appropriate for describing packet reordering experienced by a flow instead of generating packet reordering.

B. Proposed Packet-Reordering Models

Considering the limitations of current packet-reordering models, we propose a new packet-reordering generator that is implemented by extending the error model in NS-2 [1]. The proposed packet-reordering generator models packet reordering phenomenon with three parameters, and it enables us to study the impact of each of the three parameters on the performance of high-speed TCP variants.

Our packet-reordering generator can be described by the following three parameters:

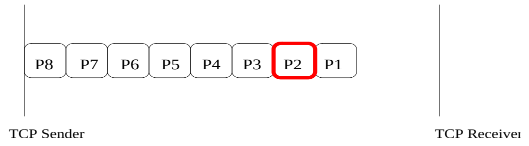

Fig. 1. Original packet sequence without reordering

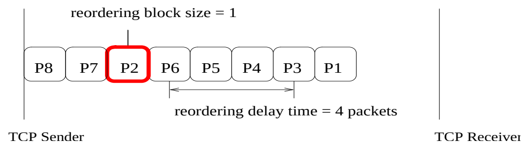

Fig. 2. New packet sequence after packet 2 is reordered with reordering block size = 1 and reordering delay time = 4 packets

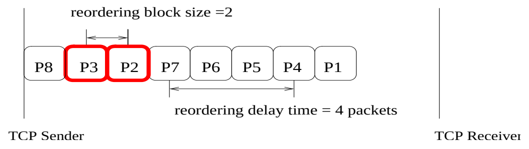

Fig. 3. New packet sequence after packet 2 and 3 are reordered with reordering block size = 2 and reordering delay time = 4 packets

- *Reordering interval*: the time interval between two consecutive packet-reordering events.
- *Reordering delay time*: the time interval from the first reordered packet in a reordering event to the earliest forwarded packet with a higher sequence number.
- *Reordering block size*: the number of packets being reordered as an entity.

A packet-reordering generator works as a special router with one input and one output port, and it forwards all incoming packets to the outgoing port. To introduce packet reordering, a *reordering block size* number of packets are delayed for a *reordering delay time* every *reordering interval*, which is called a *reordering event*. All other packets are forwarded immediately without any delay. All three parameters could be random variables following certain distributions.

In this paper, we measure a reordering delay time in two different units. Let $t(P_i)$ denote the time that packet P_i departs from a generator. Suppose that P_x is the first reordered packet in a reordering event, and P_y is the earliest packet departing from the generator among all packets with a higher sequence number (i.e., $t(P_y) < t(P_x)$ and y > x). *First*, the reordering delay time is measured as time interval $t(P_y)-t(P_x)$. *Second*, it is measured as the number of packets forwarded between $t(P_y)$ and $t(P_x)$. Note that we can determine one of these two values from another one, if the packet sending rate is known and assuming the same inter-arrival time for all packets. Both units (i.e., seconds and packets) are used in this paper based on which one is more convenient.

Figures 1 to 3 illustrate how our packet-reordering generator works. Consider that eight packets are sent from a TCP sender to a TCP receiver in the order shown in Figure 1, and they arrive at a reordering generator back-to-back in that order.

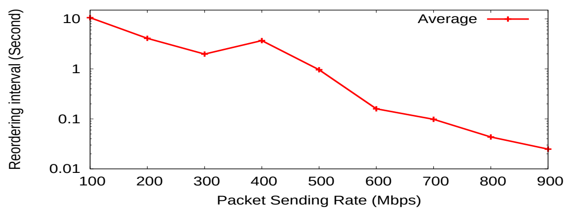

Fig. 4. The higher the packet sending rate, the shorter the average reordering interval; that is, the more the reordering events

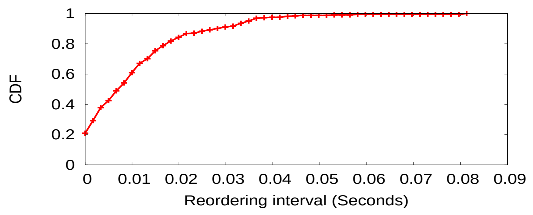

Fig. 5. The cumulative distribution function (CDF) of the reordering intervals in an experiment with a 900 Mbps rate

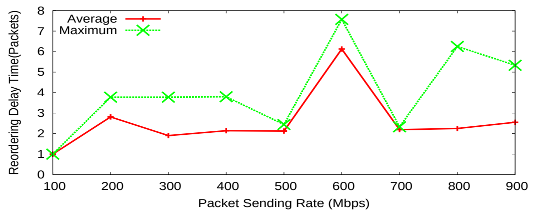

Fig. 6. Both average and maximum reordering delay times slightly increase as the packet sending rate increases.

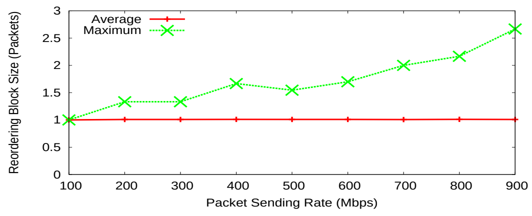

Fig. 7. Even though the average reordering block size remains almost same for all sending rates, the maximum reordering block size increases as the rate increases.

Assume that packet P_2 is the first one to be reordered at the beginning of a reordering interval. If the reordering block size is one packet, and the reordering delay time is four packets, then packet P_2 is delayed until packets P_3 to P_6 depart from the generator. In this case, eight packets arrive at the TCP receiver in the order shown in Figure 2. Now, we consider a case where the reordering block size is two packets. In this case, both packets P_2 and P_3 are delayed, and eight packets arrive at the TCP receiver in the order shown in Figure 3.

IV. Characteristics of Packet-Reordering in High-speed Networks

Now, we study the characteristics of packet-reordering in high-speed networks by using the new model. The reasonable values for three parameters are found by analyzing the experimental results conducted by Gharai *et al.* [9] in 2004.

Gharai *et al.* [9] measured the occurrence of packet reordering by transmitting UDP flows with various packet sizes for one minute at a rate ranging from 1Mbps to 900Mbps among Washington DC, Pittsburgh, and Los Angeles over DARPA SuperNet. We analyze all their experiments, and observed similar results for experiments with different packet sizes. Limited by space, below we discuss and show our findings only for experiments with a packet size of 1500 bytes.

Figure 4 shows the reordering interval that is averaged over all reordering intervals in all experiments with a 1500-byte packet size. It clearly shows that the higher the packet sending rate, the shorter the reordering interval; that is the more the reordering events. For example, the average reordering intervals at 100Mbps is about 10 seconds, whereas that at 900Mbps is only about 0.02 seconds. This is consistent with the observations in [9]. We also observe that even in the same experiment, the reordering interval is not a constant. For example, the cumulative distribution function (CDF) of the reordering intervals in an experiment with a 900Mbps rate shown in Figure 5 indicates that the maximum and minimum reordering intervals experienced by the same flow may differ by more than 100 times. This suggests that previous simulation results obtained with a fixed reordering interval may not reflect the actual performance of TCP in the presence of packet reordering.

Figure 6 shows the average and maximum reordering delay times measured by first calculating the average and maximum reordering delay times of each experiment, and then calculating the average among all experiments with the same sending rate. We note that even through the results are obtained by analyzing all experiment results conducted in [9], the plots are uneven due to the limited number of experiments. However, we can still see the pattern that both average and maximum reordering delay times slightly increase as the packet sending rate increases. For example, the average reordering delay time at 100Mbps is only 1 packet, whereas that at 900Mbps is more than 2 packets. In addition, in most cases, the average reordering delay time is less than 3 packets. That is, a significant number of reordering events lead to less than three dupACKs, and then do not trigger false fast retransmit. However, a high-speed flow is more likely to experience reordering events with a reordering delay time long enough to trigger false fast retransmit.

Figure 7 shows the average and maximum reordering block sizes. We note that the average reordering block size is relatively insensitive to the packet sending rate, and it is always very close to 1. That is, in most reordering events, only one packet is reordered, even for a flow with a high sending rate. We also clearly see that the maximum reordering block size increases as the rate increases. This implies that it is more likely for a larger number of packets to be reordered as an entity for a flow with a higher sending rate.

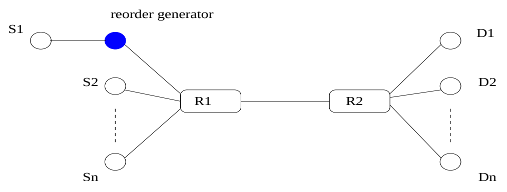

Fig. 8. Simulation Network topology

V. Impact of Packet Reordering on the Performance of High-Speed TCP Variants

The impact of each of the three reordering parameters on the performance of high-speed TCP variants is studied via simulation. We only show the simulation results for BIC [22] and HSTCP [8] in this section. Since all high-speed TCP variants use the same TCP fast retransmit and fast recovery algorithms, other high-speed TCP variants should achieve the similar performance degradation trends as BIC and HSTCP.

A. Simulation Setup

Figure 8 shows the NS-2 simulation setup. The reordering generator is introduced between high-speed TCP sender S_1 and router R_1, so that only the TCP connection between S_1 and D_1 experiences packet reordering. The bandwidth and one-way delay of the bottleneck link are set to 1000 Mbps and 50 ms, respectively. Each source and sink are connected to the bottleneck link through different access links with delays randomly varied from 0.1 ms to 0.9 ms to mitigate the phase effect. To increase traffic dynamics and further reduce phase effect, various kinds of background traffic are simulated in both directions.

The three parameters of the reordering generator are set according to our analysis in section IV. Both the reordering interval and reordering delay time are random variables with a distribution approximately following their measured CDFs respectively. Unless otherwise noted, the averages of the reordering interval and reordering delay time are set to 0.02 seconds and 20 μs, respectively, which correspond to their measured average values at 900Mbps. The reordering block size is set to a fixed value of 1 packet.

We use TCP/SACK as the agent for TCP connections (except for reordering tolerant algorithms that need special TCP agents). TCP/SACK is chosen for its higher performance compared with TCP/Reno and TCP/NewReno, and the fact that most of the reordering tolerant algorithms are based on TCP/SACK.

B. Simulation Results

Figure 9 shows the impact of reordering intervals on the performance of BIC and HSTCP. We vary the average reordering interval from 0.01 seconds to 0.5 seconds, and set the other two parameters to the values given in Section V-A. We make the following three observations. First, as we expected, a smaller reordering interval does lead to lower throughput. Second, the throughput of BIC drops sharply when the reordering interval is less than 0.02 seconds. This is because when the reordering interval is small enough so that a flow may experience more than one reordering event within one RTT, it cannot efficiently recover from false fast retransmit and fast recovery. Third, HSTCP suffers more than BIC from packet reordering. For example, with a 0.5-second reordering interval, BIC can achieve as high as 400Mbps throughput, however, HSTCP can achieve only 10Mbps throughput. Intuitively, this is because BIC is more aggressive than HSTCP in high-speed networks.

Figure 10 evaluates the impact of reordering delay times. We vary the average reordering delay time from 0 to 100 μs, and set the other two parameters to the values given in Section V-A. *First*, we note that both BIC and HSTCP achieve very high throughputs, when there is no packet reordering. However, even a very small reordering delay time (such as 20 μs) can significantly degrade their throughput. This is because the average reordering interval is set to 0.02 seconds (that is the average reordering interval of all experiments with a 900Mbps rate conducted in [9]). As discussed in the last paragraph, this interval is so small that a flow may experience more than one reordering events within one RTT. With a small reordering delay time, even though one reordering event alone may not lead to false TCP fast retransmit, two or more intermixed reordering events are more likely trigger false TCP fast retransmit and fast recovery. *Second*, we see that further increasing the reordering delay time only slightly reduces the throughput. Intuitively, once false TCP fast retransmit and fast recovery are triggered, a longer reordering delay time only makes the flow stay in fast recovery slightly longer, which does not change the throughput too much.

Figure 11 plots the throughputs of BIC and HSTCP for different reordering block sizes. As we can see from the figure, the impact of the reordering block size is very similar to that of the reordering delay time. Even a small reordering block size (such as one packet) can significantly degrade the throughput of BIC and HSTCP. However, further increasing the reordering block size only slightly reduces the throughput. The reason is that the TCP/SACK mechanism can recover quickly from multiple lost packets or reordered packets in one congestion window unless the succeeding partial ACKs are triggered.

The simulation results shown in this section demonstrate that all three reordering parameters negatively affect the performance of high-speed TCP variants. When the reordering interval is very small as measured in DARPA SuperNet [9], high-speed TCP variants suffer significantly from packet reordering even with very small reordering delay times and block sizes.

VI. Effect of Existing Reordering-Tolerant Enhancements of TCP

A. Evaluation of Existing Reordering-Tolerant TCP Enhancements

In this section, we evaluate the effectiveness of current reordering-tolerant algorithms in high-speed networks. We simulate TCP-NCR [24], AVG-DEV [14], and TCP-RR [23], which are three recently proposed algorithms to make TCP

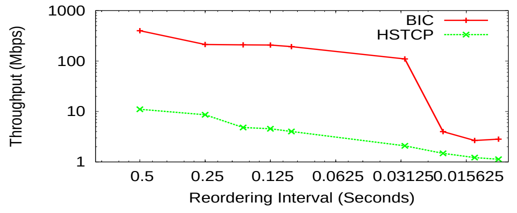

Fig. 9. The shorter the reordering interval, the lower the throughput of high-speed TCP variants. BIC achieves better performance than HSTCP, but still suffers from very small reordering intervals.

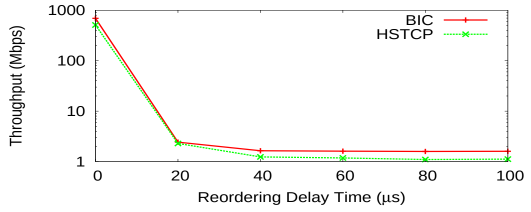

Fig. 10. Even a small reordering delay time (such as 20 μs) can significantly degrade the throughput of BIC and HSTCP. However, further increasing the reordering delay time only slightly reduces the throughput.

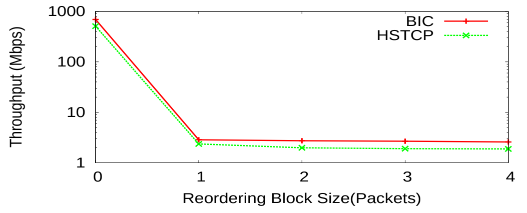

Fig. 11. Even a small reordering block size (such as 1 packet) can significantly degrade the throughput of BIC and HSTCP. However, further increasing the reordering block size only slightly reduces the throughput.

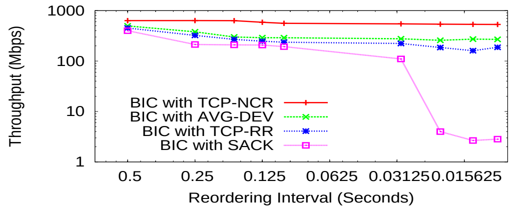

Fig. 12. Existing reordering-tolerant algorithms can significantly improve the throughput of BIC. However, BIC still suffers from packet reordering with small reordering intervals

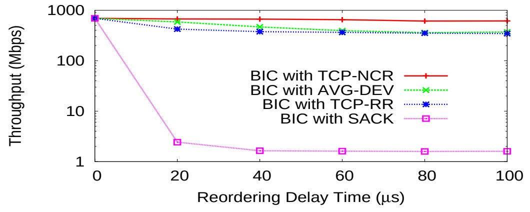

Fig. 13. Existing reordering-tolerant algorithms can significantly improve the throughput of BIC. BIC can achieve very high throughput even with long reordering delay times.

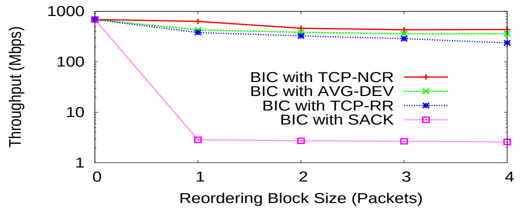

Fig. 14. Existing reordering-tolerant algorithms can significantly improve the throughput of BIC. However, BIC still suffers from packet reordering with large reordering block sizes

more robust to packet reordering. They use slightly different methods to prevent false TCP fast retransmit and fast recovery, and their simulation results show that they work better than other similar reordering-tolerant algorithms in their studied simulation environments.

We have obtained similar simulation results for both BIC and HSTCP, and below we show the results for BIC only. We simulate BIC with each of these reordering-tolerant algorithms, and evaluate their effectiveness as we vary each of the three reordering parameters of our reordering generator. We use the same sets of simulation parameters as in Figures 9, 10 and 11.

Figures 12, 13 and 14 show the effectiveness of these three reordering-tolerant algorithms as we vary the reordering interval, reordering delay time, and reordering block size. We also show the throughput of BIC without any reordering-tolerant algorithm (referred to as BIC-SACK in figures) as a reference case. We make the following three observations. *First*, all reordering-tolerant algorithms can significantly improve the throughput of BIC in all simulation scenarios. For example, BIC-SACK achieves less than 10Mbps throughput in most cases, whereas BIC with any of TCP-NCR, AVG-DEV, and TCP-RR achieves a rate greater than 100Mbps in all cases. *Second*, TCP-NCR consistently performs better than the other two algorithms: AVG-DEV and TCP-RR. This is expected, since TCP-NCR has the largest *dupthresh* (i.e., the number of dupACKs to trigger TCP fast retransmit and recovery) and the virtual layer. *Third*, even with TCP-NCR, BIC still suffers from packet reordering, especially with small reordering intervals and large reordering block sizes. For example, Figure 14 shows that BIC with TCP-NCR achieves only 463Mbps throughput with a reordering block size of 2 packets, whereas it achieves 632Mbps throughput with a reordering block size of 1 packet. Intuitively, even though TCP-NCR can effectively prevent most false TCP fast retransmit and recovery due to packet reordering only, it cannot recover quickly from TCP fast retransmit and recovery caused by packet loss and packet reordering.

B. *Related Reordering-Tolerant TCP Enhancements*

Most reordering-tolerant algorithms increase the *dupthresh* from the default value of 3 to a larger number to prevent false TCP fast retransmit and recovery. Blanton and Allman [5] adjust *dupthresh* using reordering history based on D-SACK information [27]. TCP-RR [23] proposed by Zhang *et al*. uses D-SACK and a cost function of timeout and false fast retransmission to adjust *dupthresh*. AVG-DEV [14] developed by Ma and Leung adjusts *dupthresh* by considering both the average and the deviation of reordering history. TCP-NCR [24] designed by Bhandarkar *et al*. sets *dupthresh* to approximately the size of a congestion window. TCP-Aix [25] recently proposed by Ekstr$\ddot{o}$m et al. adjusts *dupthresh*, and it separates loss recovery from congestion control.

Some algorithms, such as TCP-PR [6] proposed by Blhacek *et al*., use a time threshold based on RTT estimation to trigger TCP fast retransmit and recovery, instead of the number of dupACKs. Some algorithms, such as RN-TCP [21] developed by Sathiaseelan and Radzik, require additional information from routers to distinguish packet reordering from packet loss. A more comprehensive survey of reordering-tolerant algorithms can be found in [26].

VII. Conclusion

In this paper, we presented a new packet-reordering generator which can be described by three parameters: reordering interval, reordering delay time, and reordering block size. The new reordering generator enables us to study the impact of each of these three parameters on the performance of high-speed TCP variants. We also analyzed the experiment results measured in DARPA SuperNet [9]. Our analysis result is consistent with the findings in [9] that the average reordering interval decreases as the packet sending rate increases. We also find that the reordering delay time and reordering block size slightly increases as the packet sending rate increases. All of these analysis results imply that a TCP flow with a high sending rate is more likely to experience more false TCP fast retransmit and recovery due to packet reordering.

We also performed NS-2 simulation to study the performance of high-speed TCP variants in the presence of packet reordering. Our simulation results demonstrate that all three reordering parameters negatively affect the performance of high-speed TCP variants. When the reordering interval is very small as measured in DARPA SuperNet, high-speed TCP variants suffer significantly from packet reordering even with very small reordering delay times and block sizes.

Finally, we evaluated the effectiveness of the existing reordering-tolerant TCP enhancements. Our simulation results show that current reordering-tolerant algorithms can significantly improve the performance of high-speed TCP variants.

Acknowledgments

We would like to thank Dr. Ladan Gharai, Dr. Colin Perkins and Dr. Tom Lehman for sharing their experimental results with us. The work reported in this paper is supported in part by a Grant-in-Aid from UNL Research Council and Nebraska EPSCoR First Award.

References

[1] Network simulator 2, *http://www.isi.edu/nsnam/ns/*.
[2] G. M. Allman, V. Paxson and W. Stevens, ”TCP Congestion Control”, *RFC 2581*, April 1999.
[3] J. Bellardo and S. Savage, ”Measuring packet reordering” *ACM SIGCOMM Internet Measurement Workshop*, Marselle, France, November 2002, pp. 97 - 105.
[4] J. Bennett and C. Partridge and N. Shectman, ”Packet reordering is not pathological network behavior” *IEEE/ACM Transaction on Networking*, 7(6):789–798, December 1999.
[5] E. Blanton and M. Allman, ”Using TCP DSACKs and SCTP Duplicate TSNs to Detect Spurious Retransmissions”, *RFC 3708*, February 2004.
[6] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka, ”TCP-PR: TCP for Persistent Packet Reordering” *23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03)*, May 2003, pp. 222 - 231.
[7] S. Dharmapurikar and V. Paxson, ”Robust TCP stream reassembly in the presence of adversaries”, *14th USENIX Security Symposium*, Baltimore, MD, August 2005, pp. 222 - 231.
[8] S. Floyd, ”HighSpeed TCP for Large Congestion Windows”, *RFC 3649*, December 2003.
[9] L. Gharai, C. Perkins, and T. Lehman, ”Packet reordering, high speed networks and transport protocol performance”, *In Proceeding of the 13th ICCCN*, Chicago, IL, October 2004, pp. 73- 78.
[10] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, ”Measurement and Classification of Out-of-Sequence Packets in a Tier-1 Backbone”, *IEEE Infocom 2003*, San Francisco, CA, March 2003, pp. 1199- 1209.
[11] A. Jayasumana, N. Piratla, A. Bare, T. Banka, R. Whitner, and J. McCollom, ”Reorder Density and Reorder Buffer-occupancy Density Metrics for Packet Reordering Measurements”, *Internet Draft*, 2006.
[12] C. Jin and D. Wei and S. Low, ”FAST TCP: Motivation, Architecture, Algorithms, Performance”, *In Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.
[13] T. Kelly, ”Scalable TCP: Improving Performance in HighSpeed Wide Area Networks”, *ACM SIGCOMM Computer Communication Review*, 33(2):83 - 91, April 2003.
[14] C. Ma and K. Leung, ”Improving TCP robustness under reordering network environment”, *In Proceedings of Global Telecommunications Conference*, November 2004.
[15] J. Mogul, ”Observing TCP dynamics in real networks”. *In Proceedings of ACM SIGCOMM*, Baltimore, MD, August 1992.
[16] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov and J. Perser, ”Packet Reordering Metric for IPPM ”. *Internet Draft*, 2006.
[17] G. Neglia, V. Falletta and G. Bianchi, ”Is TCP packet reordering always harmful?”, *In Proceeding of MASCOTS*, October 2004.
[18] V. Paxson, ”End-to-end internet packet dynamics”, *IEEE/ACM Transactions on Networkings*, 7(3): 277 -292, June 1999.
[19] N. Piratla and A. Jayasumana, ”Reordering of packets due to multipath forwarding - an analysis”, *In Proceedings of IEEE ICC*, Istanbul, Turkey, June 2006, pp. 28 - 36.
[20] M. Przybylski, B. Belter, and A. Binczewski, ”Shall we worry about packet reordering?”, *In Proceedings of TERENA Networking Conference*, Poznan, Poland, June 2005, pp. 28 - 36.
[21] A. Sathiaseelan and T. Radzik, ”Reorder Notifying TCP (RN-TCP) with explicit packet drop notification”, *International Journal of Communication Systems*, 19(6): 659 - 678, August 2006.
[22] L. Xu, K. Harfoush, and I. Rhee, ”Binary increase congestion control for fast long-distance networks”, *In Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.
[23] M. Zhang, B. Karp, S. Floyd, and L. Peterson, ”RR-TCP: A reordering robust TCP with DSACK”. *In Proceedings of IEEE ICNP*, Atlanta, Georgia, November 2003.
[24] S. Bhandarkar, A. L. N. Reddy, M. Allman, E. Blanton, ”Improving the Robustness of TCP to Non-Congestion Events”, *RFC 4653*, August 2006.
[25] S. Landstr$\ddot{o}$m, H. Ekstr$\ddot{o}$m, L. Larzon and R. Ludwig, ”TCP-Aix: making TCP robust to reordering and delay variations”, *Research Report in Luleá University of Technology*, 2006.
[26] 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 (IEEE TPDS)*, 2006.
[27] S. Floyd, J. Mahdavi, M. Mathis and M. Podolsky, ”An Extension to the Selective Acknowledgement (SACK) Option for TCP”, *RFC 2883*, July 2000.