Hindawi
Journal of Computer Networks and Communications
Volume 2020, Article ID 8465191, 8 pages
https://doi.org/10.1155/2020/8465191

Review Article

Packet Reordering Metrics to Enable Performance Comparison in IP-Networks

Pedro Rodrigues Torres-Jr and Eduardo Parente Ribeiro

Post-Graduate Program in Electrical Engineering, Federal University of Paraná, Centro Politécnico, Curitiba, PR 81531-890, Brazil

Correspondence should be addressed to Pedro Rodrigues Torres-Jr; pedro.torres@ufpr.br

Received 15 October 2019; Revised 15 March 2020; Accepted 4 May 2020; Published 30 May 2020

Academic Editor: Djamel F. H. Sadok

Despite the existence of several metrics to perform measurements on out-of-order packets, few works have used these metrics for comparative purposes. A potential reason for this is that the use of these simple singleton metrics makes it difficult to analyze all the effects of packet reordering. On the other hand, more complete metrics are represented in a vectorial manner, making comparative analysis challenging. In this paper, we present a scenario for testing and describe a methodology for conducting experiments to compare network paths with respect to unordered packets. The results of several simulations explore simple packet reordering metrics derived from vector metric that may allow future work to be benchmarked against. We demonstrated the behaviour of some TCP congestion control algorithms by adjusting different levels of reordering. We highlight good results with the *Entropy* reorder metric.

1. Introduction

A desirable network feature is that the packets arrive in the same order they were sent. Due to the ever-present parallelism in the network infrastructure, the reordering happens and affects the performance of the transport and application layer protocols. Several factors can contribute to that effect, among them: links load balancing, routing instability, parallel processing, and others [1]. In general, we can consider that the various redundancies applied to the network, either to increase availability or capacity, can be a point of packets overtaking, which can lead to out-of-order delivery.

RFC 4737 [2] defines a standard with several metrics for packet reordering. Some simple scalar metrics are as follows: Reordered Packet Ratio and Reordering-Free Runs. Other nonsingleton metrics are Reordering Extent, Reordering Late Time Offset, Reordering Byte Offset, Gaps between Multiple Reordering Discontinuities, and n-Reordering. However, the use of any of these metrics alone is insufficient to completely characterise all aspects of packet reorder in a path on the network. RFC 5236 [3] describes other improved vectorial metrics that allow capturing the effects of packet reordering. A detailed comparison of these metrics regarding advantages and disadvantages can be seen in [4].

A protocol that has been thoroughly tested to tolerate well the packets reordering in the network is the TCP. Although the research carried out with TCP is quite exuberant to evaluate its performance with each modification that is proposed, it is still difficult to find works that use a standard reordering metric that turns possible to make a comparative analysis.

In this work, we describe a methodology for measuring packet reordering. We present the results obtained through simulations performed in a virtual environment that uses the protocols and algorithms available in several systems. A well-known metric is used, the *Reorder Density*, and from it we derived simpler known metrics that turn possible the comparison with others works. We evaluated these metrics through more than 16000 simulations, with different probability distributions to generate the packet reordering and apply different TCP configurations.

The simple metrics *Mean Displacement* and *Entropy* [5] are used to analyze the results. These metrics can be easily used for comparing the protocol performance and also allow a simpler comparison of the quality of service provided in terms of packets reordering by the Internet Service Providers (ISP). In scenarios where a small difference in reordering may lead to a large difference in protocol performance, the *Entropy* metric showed to be more appropriate to be used.

1.1. Organisation of This Work. The purpose of this article is to present the main metrics of packet reordering and suggest methods for producingg and measuring out-of-order sequences that can be useful for performance comparison for the development of communication protocols and systems. We will present the methodology and results of many simulations with differentt types of reordering and their behaviour in relation to the TCP throughput configured with various congestion control algorithms.

The remainder of this article is organised as follows: In Section 2 we will review the metrics about packet reordering, as well as the works that study the effects of this reordering over TCP. In Section 3 we will present the methodology and the environment configuration that we used to carry out the simulations. Next, in Section 4, we will present the results. Finally, in Section 5 we will describe the conclusions and future work. Information about the source code used in this work is available in Section 6.

2. Review on Packets Reordering

In this session, we will present the main definitions of packets reordering and discuss the tolerance of TCP to unordered segments. Before we begin with the definitions that will be used in the remainder of this paper, we will show the problem of using a simple reordering metric, commonly used by network operators: percentage of out-of-order packets.

Consider two out-of-order packet sequences (1, 2, 4, 5, 3, 6) and (1, 2, 5, 4, 3, 6). We can interpret the sequences in distinct ways. (1) In both sequences we can say that the packets 3, 4, and 5 are out-of-order (50%). (2) In the first sequence only packet 3 is out-of-order (16.6%). (3) In the second sequence packets 3 and 4 are out-of-order (33, 3%). Even considering only delayed packets as out-of-order, that metric does not capture all the characteristics of the disorder.

A reordering metric should be simple and still capture the amount and the magnitude of the reordering. At the same time, it should be orthogonal to other metrics such as packet loss and duplicate packets. It should be useful for applications, for example, to determine the buffer size, as well as having other features including being robust with respect to network events and being easy to compute [6].

A reordering can be represented for a sequence of packets (1, 2, ..., N) transmitted in a network, assigning an index in order of arrival at the recipient, disregarding lost and duplicate packets. If the index given to a packet m is $(m + d_m)$, with $d_m \neq 0$, it can be said that a reordering event occurred in the network. For this we will use the notation $r(m, d_m)$. If no reordering occurs on the network, the sent stream will have the same index received on the recipient, and therefore $dm = 0$. A packet will be said late if $d_m > 0$ and early if $d_m < 0$. Thus, the reordering of a sequence of packets is represented by the union of reordering events of R, represented by $R = \cup\{r(m, d_m) \mid d_m \neq 0\}$ [6].

Reorder Density (*RD*) is defined as the discrete density of the frequency of packets with respect to their displacement, i.e., the lateness and earliness from the original position. Let $S[k]$ denote the set of reordering events in R with displacement equal to k, i.e., $S[k] = \{r(m, d_m) \mid d_m = k\}$. Let $|S[k]|$ be the cardinality of the set $S[k]$. Thus, $RD[k]$ is defined as $|S[k]|$ normalised with respect to the total number of received packets N, that is, not considering lost or duplicates packets.

$$RD[k] = \frac{|S[k]|}{N}. \tag{1}$$

Equation (1) defines *RD* for $k f = 0$. *RD* uses a threshold D_t, in which a packet arriving early or late, beyond to that value, is considered as lost; therefore $-D_t \leq k \leq +D_t$.

$$RD[0] = 1 - \frac{|S[k]|}{N}. \tag{2}$$

Equation (2) define $RD[0]$, that represents the packets in which the index is the same as the sent sequence number and, therefore, have not been reordered. Some features derived from the $RD[i] \geq 0$, $\forall i$ are as follows:

$$\sum_i RD[i] = 1, \tag{3}$$

$$\sum_i i \times RD[i] = 0. \tag{4}$$

Table 1 shows the example of an eight-packet sequence that was received out-of-order, the index assigned, and the displacement. In this sequence no packet was considered lost or duplicated. The value of the set R is as follows:

$$R = (2, 2), (3, 2), (4, 2), (5, 2), (6, -4), (7, -4). \tag{5}$$

Table 2 shows *RD* for the prior example. It can be seen that two packets arrived in four positions early, two packets arrived in right order, and four packets arrived in two positions late.

In addition to the *RD*, other metrics have already been defined. The *Reorder Buffer-occupancy Density* (RBD) is also a vector metric of a buffer that could allow the recovery of out-of-order packets [7]. The authors of [4] show the advantage of using RD over other metrics, in particular, *Reordering Extent* and *n-Reordering*, detailed in RFC 4737. These metrics are classified as lateness-based metrics; i.e., a packet is not considered reordered unless it is late. Thus, in this work, we will use RD to derive the simpler metrics that allow a comparative analysis for performance evaluation.

2.1. Simple Reordering Metrics. The use of *RD* is described in RFC 5236 [3]. Although the document shows the advantages of using this metric, it does not define a simpler metric, analogous to the percentage. In the work of [5] several scalar

Table 1: Example of out-of-order for eight-packet sequence.

Description	$p1$	$p2$	$p3$	$p4$	$p5$	$p6$	$p7$	$p8$
Arrived sequence	1	6	7	2	3	4	5	8
Received index	1	2	3	4	5	6	7	8
Displacement	0	−4	−4	2	2	2	2	0

Table 2: *Reorder Density (RD)* for the eight-packet sequence.

Description	d1	d2	d3
Displacement	−4	0	2
Packets	2	2	4
Ratio	0.25	0.25	0.50

metrics are derived from *RD*. Among these metrics are *Mean Displacement* and *Entropy*.

Since packets may arrive early or late depending on the reorder event that occurs on the network, the displacement may be either positive or negative. When computing the Mean Displacement of these packets, the result will be zero, as seen in equation (4). Therefore, one way to calculate the *Mean Displacement* (*MD*) is by considering the magnitude of the displacement, according to the following equation:

$$MD = \sum_{i=-D_t}^{i=+D_t} |i| \times RD[i]. \quad (6)$$

where D_t is the threshold used to restrict the calculation to a finite number of packets. So, packets arriving too late or too early are considered lost.

Another simple metric that can be used to capture packet scattering is *Entropy* (E_r), which is defined as the negative of the logarithm of the probability density function [8]. Since *RD* is a discrete probability distribution, this can be calculated as follows:

$$E_r = -1 \times \sum_{i=-D_t}^{i=+D_t} RD[i] \times \ln RD[i]. \quad (7)$$

For *RD* from Table 2 the *Mean Displacement* (*MD*) is equal to 2 and the Entropy (E_r) is approximately 1.04.

2.2. TCP Tolerance to Reordering. The *Transmission Control Protocol* (TCP) is a reliable, connection-oriented, stream-oriented transport layer protocol that has become widely used on the Internet because of these characteristics. In TCP, flow control and acknowledgment (ACK) are indexed to a byte value, rather than a packet or message number. The TCP minimum transfer unit is a segment or packet of data.

When the destination receives a TCP segment, it sends an acknowledgment (ACK) with the next byte of data expected to receive. In addition to ensuring the delivery of all segments, this packets exchange allows calculating round-trip time (RTT). Note that the RTT encompasses not only the round-trip time but the times spent in queues and processing.

TCP flow control is used to prevent data from being sent at a higher receiver reception capacity. For this, an *advertised window* field is used in the TCP header to indicate the amount of data that can be sent without acknowledgment. TCP also needs to deal with network congestion and adapt the sending rate to the network's circumstances. Several congestion controls have been implemented in TCP [9–13]. These algorithms deal with controlling a variable in the sender known as the *congestion window* (*cwnd*). The maximum number of segments that can be sent before receiving an ACK is the minimum between the *advertised window* and *cwnd*.

The control of the variable *cwnd* is divided into two stages. The first stage precedes the fact that a packet has been lost or has arrived out-of-order and the algorithm's *slow start* and *congestion avoidance* are used. In the second stage, after the detection of a lost or out-of-order packet, the *fast retransmit* and *fast recovery* algorithms take effect [14].

The effects of packet loss or reordering imply the TCP control algorithms behaviour. The loss of a packet is a sign that some kind of congestion has occurred in the network. On the other hand, reordering can occur in such a way that it will be confused with a loss and will cause a decrease in data transfer throughput. Any packet that arrives out-of-order will cause a duplicate ACK to be sent, which is the same behaviour if that segment had been lost. The *fast retransmit* algorithm uses a default number to detect a packet loss, typically configured as three duplicates.

ACKs on most operating systems resend the segment immediately. The degradation of the TCP throughput is impacted if more packets are lost and reordering occurs in the same transmission window [15].

The cumulative TCP acknowledgment prevents the fast retransmit algorithm from knowing more than one loss simultaneously, and this will cause the value of the variable *cwnd* to decrease, which will result in a decrease in transmission rate [16]. For this, several papers have been proposed to include selective acknowledging (SACK) [17,18] and duplicate acknowledging (D-SACK) [19]. SACK allows to simultaneously inform which segments have been received. D-SACK defines a method in which the recipient informs the sender about segments that have received duplicates, thus allowing the sender to infer that reordering has occurred.

Several papers have studied the side effect of the arrival of out-of-order packets on the network. However, in the simulation environment of these works, the reordering measurement unit is not standardised and is also difficult to reproduce and compare. In [15,20] the measurement is performed considering the percentage of the packets out-of-order. In [21], a comparison of several TCP algorithms that are tolerant to reordering is made. In the simulation performed, the packet reordering rate is measured using their own metric, called *path delay factor*.

Although, by the characteristics of the Internet, the packets can arrive out-of-order, any parallelism in the network must be tested to avoid this reordering [16]. The question whether or not we should be concerned with reordering packets should be considered at all levels, from the development and testing of new protocols to the approval of the links being activates.

3. Reordering Measurement Methodology

In this section, we will describe the methodology used to measure packet reordering in the network and how we use this measurement in several TCP throughput tests configured with different characteristics.

Figure 1 represents the topology created in *Mininet* emulator [22] to perform the simulations. Two hosts, A and B, were configured as origin and destination traffic, respectively. The central host S1 has been manipulated to limit the bandwidth between A and B and cause the packets to be unordered. The settings performed on S1 were only to simulate different network operating conditions. The measurement methodology does not depend on these settings.

Since all hosts in the Mininet are virtualized Linux systems, the links characteristics have been configured through *Network Emulation* (netem) [23], which provides the functionality for protocol testing, emulating network characteristics such as packet delay, loss, duplication, and reordering. The bandwidth between hosts is controlled by the use of the *Hierarchical Token Bucket* (HTB), which limits the output traffic based on the *Token Bucket Filter* algorithm, which does not depend on the physical link characteristics [24].

In our environment, we are interested only in the characteristics of packet reordering, so bandwidth and delay were kept constant in traffic forwarding direction, from A to B, with 100 Mbps and 100 ms, respectively.

To generate the packets reordering a random delay is applied to each packet. That jitter was configured in S1, in the forwarding traffic from A to B, through the use of *netem*. The queue inside *netem* that keeps packets in order by time to send was adjusted to accommodate up to 5000 packets. The packet delay used follows a probability distribution, thus causing its earliness or lateness. Three probability functions were tested: uniform distribution, normal distribution, and exponential distribution. We used 62 jitter values between 0 and 10 *ms* to test both lower and higher reordering.

To perform the measurements, an F packet stream must be generated with a sequential identification $F = (1, 2, \ldots, N)$. The packet type should be chosen in such a way that it does not overload the NIC drivers and CPU. The amount of packets N in the stream should be large enough to carry out a test of sufficient duration and to be able to occupy the buffers of the systems. Each packet in the stream is required to be M bytes and be forwarded at a rate of R in order to avoid excessive interruptions and to ensure that the packets are not dispersed with each other. Table 3 summarises these parameters and the values used in the experiments.

The link measurements of *RD*, *MD,* and E_r were performed on B by observing the arrival order of a sequence of 65535 packets sent from A. That traffic consisted of ICMP packets type 8 (*echo request*). Each packet had MTU interface size of 1518 bytes and forwarded at the configured bandwidth of 100 Mbps. The maximum packets size was chosen to not overload the hosts with excessive interruptions and the forwarding rate was chosen to allow the minimum interval between the packets, making it easy to change the order by the network. The threshold D_t, the maximum displacement for *RD* calculation *RD* used was 200. We use this value to capture large displacements. Note that in our topology, we have an environment free of packet loss or with some known anomalous behaviour that could duplicate packets.

Figure 1: Topology used in packet reordering tests: A is the traffic source, S1 is an intermediate node that limits bandwidth, delay, and generates packet reordering, and B is the traffic destination.

Table 3: Parameters to be configured to perform the reorder tests.

Parameter	Description	Selected value
F	Ordered packet stream	ICMP echo request. The 16-bit sequential number header field was used to set the values. The ICMP echo reply was disabled in the receiver.
N	Number of packets	65535 ICMP packets for $a \approx 8\,s$ test with the selected M and R.
M	Size of each packet	1518B for the complete packet. This value is the path MTU.
R	Packets-per-second rate	Since we are simulating 100 Mbps links, the sending rate was 8235 packets-per-second with 1518 bytes each one. This rate is well sustained on both sending and receiving hosts.

For TCP measurements, the *iperf* tool was used. Each test consisted of a data transfer of approximately 120 MB, which, in the best case, would spend more than 10*s* to complete the transfer. For each link configuration, 30 TCP tests were performed using three congestion control algorithms: *bic*, *high speed,* and *Westwood*. SACK and D-SACK were kept on, and the other parameters, including buffers size, were not changed. In total, 16, 740 simulations were generated. The results are presented in the following section.

4. Results

In this section we will show the simulations results performed in the *Mininet* environment as detailed in Section 3. Firstly, we will show the data for validation of the proposed methodology. The graph of Figure 2 shows the probability density function (PDF) with the three distributions used to cause the reordering of the packets in node S1: uniform distribution, normal distribution, and exponential distribution. The theoretical curves were plotted considering the delay of 100 *ms* as the mean and standard deviation of 1.2475.

After playing the ICMP traffic by sending the 65535 sequential packets from host A and captured in host B, as mentioned previously, the *RD* for the three different probability distributions was calculated. The graph of Figure 3 shows the *RD* histogram for the same delay and jitter of

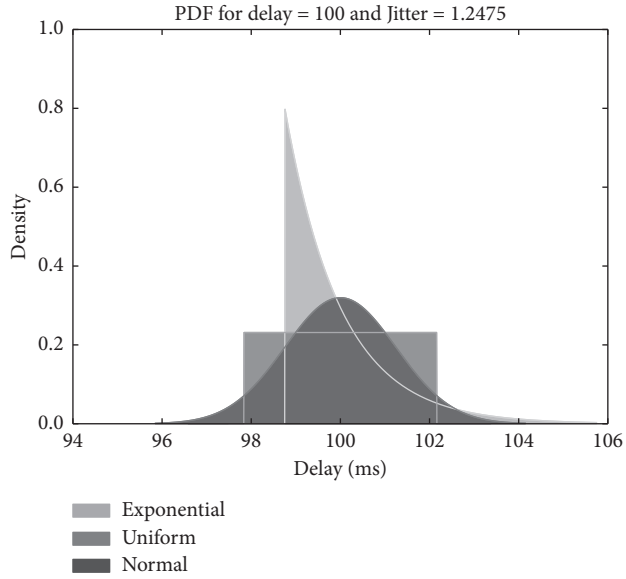

Figure 2: PDF for three different distributions.

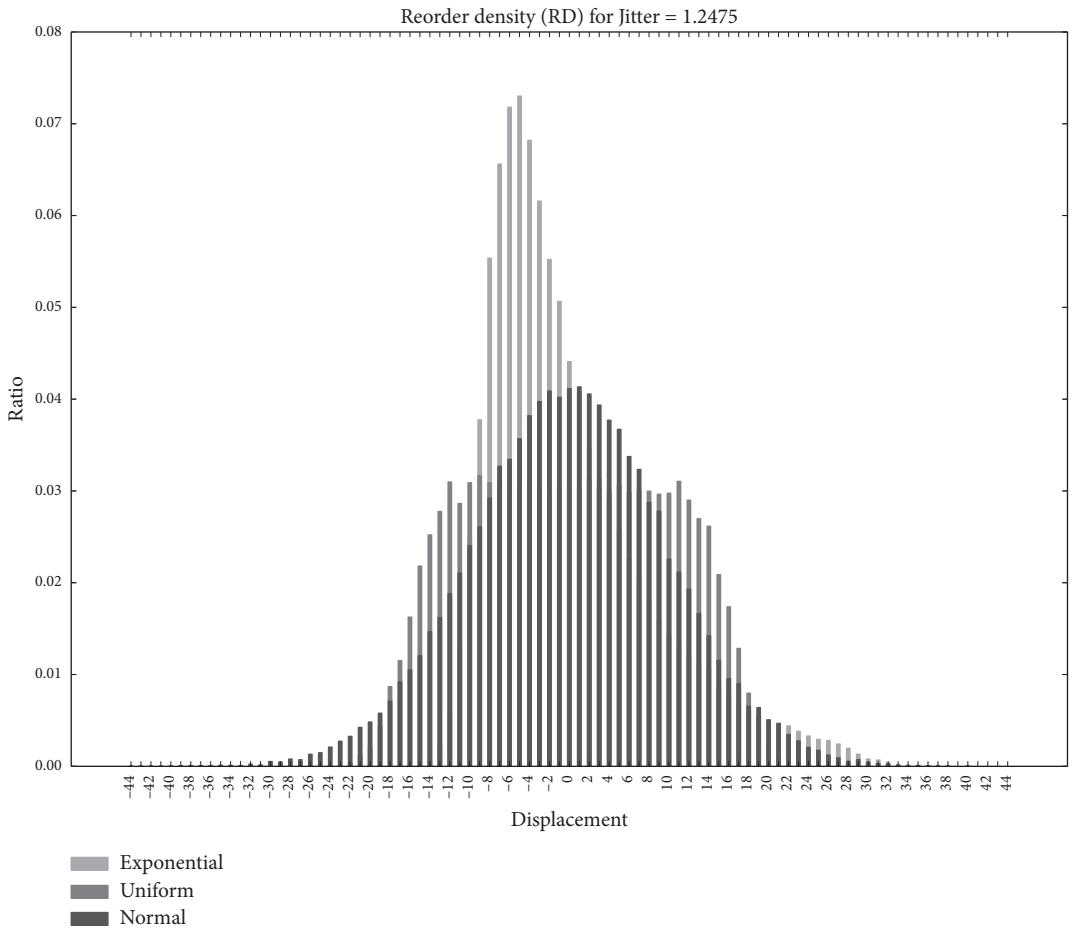

Figure 3: *RD* measured for each probability distribution.

the theoretical curves. The abscissa axis contains the packet displacement, which can be either negative for packets that have arrived earlier or positive for packets that have been late. It is possible to note that the shape of the curves is similar to the probability distribution function used. Since *RD* is a vector metric, the behaviour of each simulation is

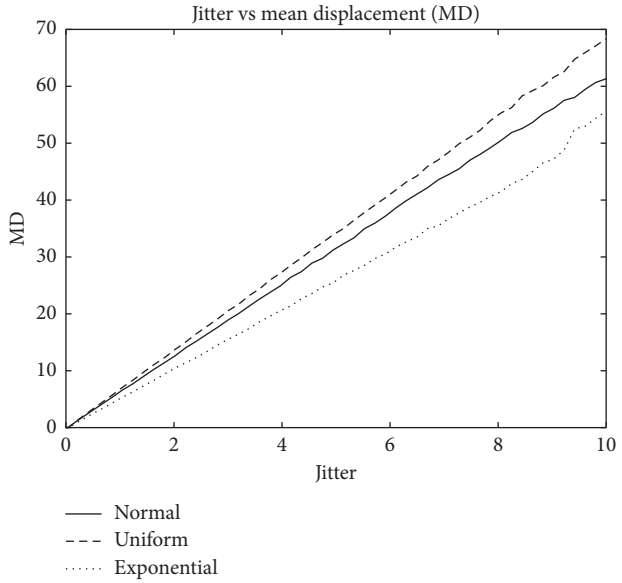

Figure 4: *MD* for diverse *jitters*.

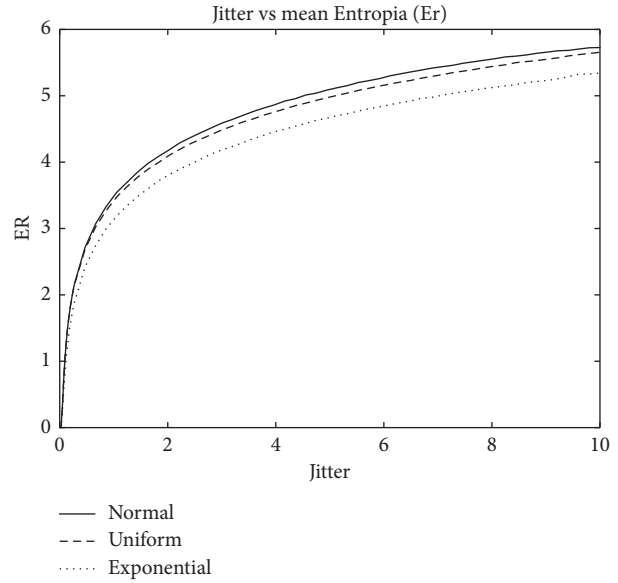

Figure 5: E_r for diverse *jitters*.

easily shown in a histogram-type graph, but it is difficult to use for comparison with other variables.

To evaluate the reordering caused by these different distributions we will check the corresponding *Mean Displacement* (*MD*) and *Entropy* (E_r) metrics. The graph of Figure 4 shows the *MD* curve calculated for each *RD* corresponding to the 62 jitters chosen between 0 and 10 *ms* to be tested. It can be noted that the *MD* for the uniform distribution is systematically higher than the *MD* of the normal distribution, which is higher than the *MD* of the exponential distribution. For a 1 *ms* jitter, the *Mean Displacement* already reaches approximate 7.1, 6.6 and 5.3, for the uniform, normal, and exponential distributions, respectively. The *MD* growth is linear with the jitter, which can be a disadvantage in cases where there is a great degradation of the quality of the network for a small variation in the packets delay.

The *Entropy* for the corresponding *RD* has also been calculated. The graph of Figure 5 shows E_r for each distribution tested. Different from *MD*, we can observe that E_r for the normal distribution is systematically higher than the uniform distribution, which in turn is higher than the exponential distribution. This difference occurs because the *Entropy* calculation is influenced positively by greater spread distributions, that is, with larger packet displacements. It is possible to observe that for small delay variation, the Entropy is already significant and this value, other than *MD*, does not grow linearly. Instead, growth follows a logarithmic curve, making Entropy suitable for cases where a small variation in the packet send order can significantly affect the network quality.

In order to show the utilisation of *MD* and E_r metrics for the comparison of the performance of the upper layer protocols, 30 TCP flow tests were performed for each of the 62 jitters values. In this scenario, we use the default system configurations with SACK and D-SACK enabled, together with *bic* control congestion algorithm. The graph of Figure 6 presents the results of the average TCP throughput for

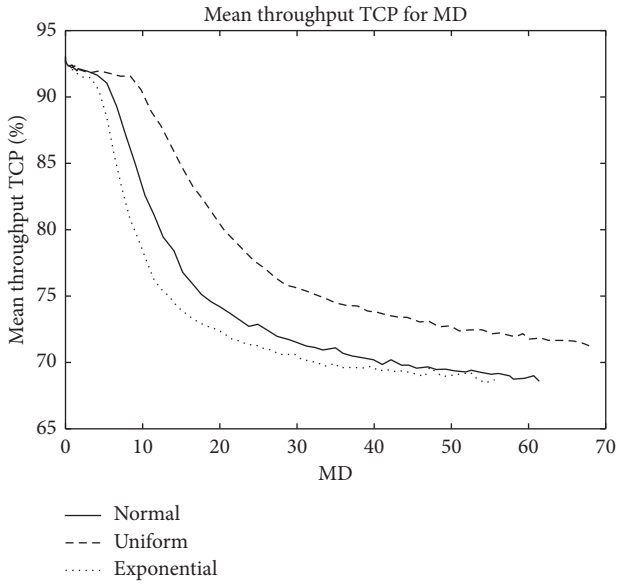

Figure 6: Throughput TCP for the values of *MD*.

different values of *MD*. Note that the *Mean Displacement* of the packets, at the moment of the throughput collapse, is quite different for the three distributions; in addition, much of the degradation occurs in a small range of delay variance, making it difficult to compare.

Still using the same configuration described above, the analysis was performed against E_r. The graph of Figure 7 shows the results of the various TCP flow tests. Unlike the *MD*, the moment of the throughput collapse occurs for intermediate *Entropy*, allowing for observing more easily the behaviour for small variations in the order of the packets. Nevertheless, it is possible to notice that the distribution of probability used considerably compromises the comparison. While for the uniform distribution, an *Entropy* of 3.8, the

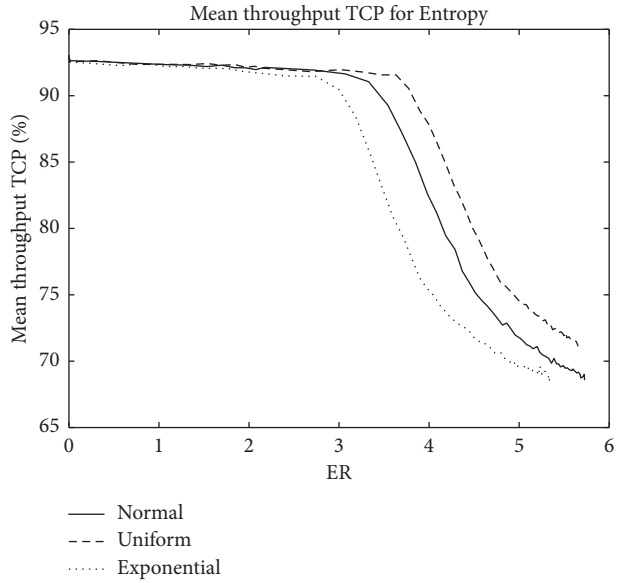

Figure 7: Throughput TCP for the values of E_r.

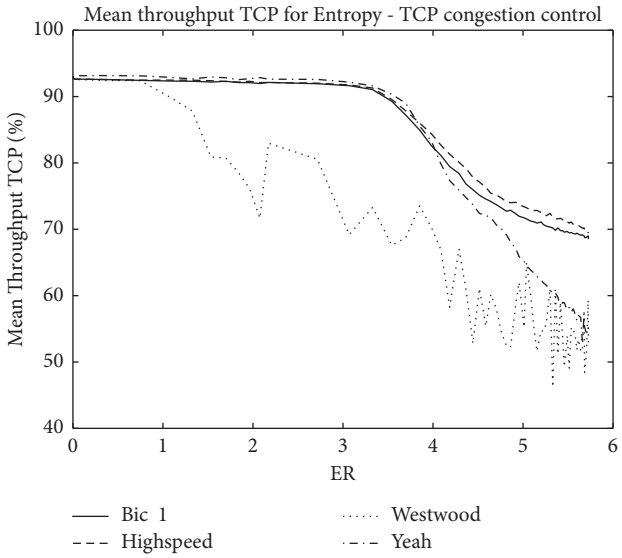

Figure 8: TCP throughput using different congestion controls for E_r.

throughput is above 90%, for the exponential distribution, the TCP throughput falls below 80%.

When we have a known probability distribution and use *Entropy* to compare the protocols, this allows evaluating the robustness of these protocols to a certain packet reordering. The graph of Figure 8 shows the comparison of four TCP congestion control algorithms: *bic*, *highspeed*, *Westwood,* and *yeah*. The tests were performed using jitters between 0 and 10 *ms*, but with the same probability distribution: normal. We see that the *bic*, *highspeed,* and *yeah* algorithms perform well for *Entropy* of approximately 3.3. The *Westwood* algorithm collapsed with a much lower value, approximately 0.75. The Entropy values for the normal distribution were generated with a jitter of 0.8585 *ms* and 0.075 *ms*, respectively. In spite of the small time difference of the *jitter* to the collapse point among the algorithms, the scale of the *Entropy* allowed differentiating them clearly.

5. Conclusion

In this work we describe the methodology for generating out-of-order packet sequences and the measurement of *Reorder Density* in a virtual environment, which can be modified for the validation of links and protocols in a real environment. From the vectorial metric *RD* we use two scalar metrics, *Mean Displacement* and *Entropy*, to compare the performance of several TCP protocol configurations.

The results showed that both *MD* and E_r, derived from *RD*, can be used to compare the performance of protocols sensitive to reordering. For cases where few packets reorder can cause a high performance degradation, the *Entropy* was more significant in the measurement due to its logarithmic scale. However, we realise that these two simple metrics are sensitive to the shape of the probability distribution that produces the disorder, and thus, the same *Mean Displacement* or *Entropy* may reflect on a different performance to a protocol being evaluated.

In the emulations where the reordering was caused using the same probability distribution, it was possible to clearly detect the maximum *Entropy* until the performance collapses. As future work, we propose an investigation in a scalar metric that allows easy comparison of the *RD* histograms, considering the data density shape.

Data Availability

The software components used in this work are open-source and available for download in https://bitbucket.org/torresweb/reorder/

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. Przybylski, B. Belter, and A. Binczewski, "Shall we worry about packet reordering?" *Computational Methods in Science and Technology*, vol. 11, no. 2, pp. 141–146, 2005.

[2] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser, *Packet reordering metrics*, RFC 4737, RFC Editor, 2006.

[3] A. Jayasumana, N. Piratla, T. Banka, A. Bare, and R. Whitner, *Improved packet reordering metrics*, RFC 5236, RFC Editor, 2008.

[4] M. Nischal, P. Anura, and P. Jayasumana, "Metrics for packet reordering—a comparative analysis," *Inter-National Journal of Communication Systems*, vol. 21, no. 1, pp. 99–113, 2008.

[5] B. Ye, A. P. Jayasumana, and M. Nischal, "On monitoring of end-to-end packet reordering over the internet," in *Proceedings of the International conference on Networking and Services (ICNS'06)*, p. 3, IEEE, Silicon Valley, CL, USA, July 2006.

[6] M. Nischal, A. P. Jayasumana, and A. A. Bare, "Reorder density (RD): a formal, comprehen- sive metric for packet reordering," *Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, vol. 3462, pp. 78–89, 2005.

[7] M. Nischal, A. P. Jayasumana, A. A. Bare, and T. Banka, "Reorder buffer-occupancy density and its application for measurement and evaluation of packet reordering," *Computer Communications*, vol. 30, no. 9, pp. 1980–1993, 2007.

[8] C. E. Shannon and W. Weaver, "The Mathematical Theory of Communication," The University of Illinois Press, Champaign, IL, USA, 1971.

[9] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proceedings of the INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4, pp. 2514–2524, IEEE, Hong Kong, China, March 2004.

[10] S. Ha, I. Rhee, and L. Xu, "Cubic," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[11] S. Floyd, "Highspeed TCP for large congestion windows," RFC 3649, RFC Editor, 2003.

[12] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP westwood: end-to-end congestion control for wired/ wireless networks," *Wireless Networks*, vol. 8, no. 5, pp. 467–479, 2002.

[13] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–12, IEEE, Barcelona, Spain, April 2006.

[14] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control. RFC 5681," RFC Editor, 2009.

[15] M. Laor and L. Gendel, "The effect of packet reordering in a backbone link on application throughput," *IEEE Network*, vol. 16, no. 5, pp. 28–36, 2002.

[16] J. C. R. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 789–798, 1999.

[17] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," *RFC 2018*, RFC Editor, 1996.

[18] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno and SACK TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[19] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," *RFC 2883*, RFC Editor, 2000.

[20] J. Bellardo and S. Savage, "Measuring packet reordering," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, pp. 97–105, ACM, Marseille, France, November 2002.

[21] 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, 2007.

[22] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings Of the 9th ACM SIGCOMM Workshop On Hot Topics In Networks, Hotnets- IX*, pp. 19:1–19:6, ACM, New York, NY, USA, October 2010.

[23] S. Hemminger, "Network emulation with NetEm," in *Proceedings Of the Linux Conference Australia*, pp. 19–23, Canberra, Australia, April 2005.

[24] Hierachical token bucket theory, 2002, http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm.

[25] Reorder scritps, 2019, http://www.bitbucket.org/torresweb/reorder/.