An Accurate Packet Loss Emulation on a DPDK-based Network Emulator

Kanon Sasaki
Electrical Engineering and Electronics, Kogakuin University Graduate School, Tokyo, Japan
cm19019@ns.kogakuin.ac.jp

Takahiro Hirofuchi
National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan
t.hirofuchi@aist.go.jp

Saneyasu Yamaguchi
Department of Information and Communication Engineering, Kogakuin University, Tokyo, Japan
sane@cc.kogakuin.ac.jp

Ryousei Takano
National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan
takano-ryousei@aist.go.jp

ABSTRACT

A software-based network emulator is widely used for performance evaluation of communication software and protocols thanks to lower cost and higher extensibility compared with a hardware-based approach. However, NetEm, which is a popular software-based network emulator in Linux, suffers from inaccurate emulation capability on high speed networks. The DPDK-based network emulator (DEMU) is a promising tool to address this problem, but it does not support packet loss emulation. In this paper, we design and implement the Gilbert-Elliott packet loss model on DEMU. Through experiments of TCP performance on a 10 Gigabit network environment, we demonstrate that (1) the proposed method accurately controls the packet loss ratio and burstiness, and (2) the TCP offload engine mechanism can degrade packet loss accuracy. Consequently, the accuracy of DEMU is 305 times higher than that of NetEm for random packet loss emulation.

CCS CONCEPTS

• Network → Network performance evaluation → Network measurement

KEYWORDS

Network emulator, Packet loss, TCP/IP

ACM Reference format:

Kanon Sasaki, Takahiro Hirofuchi, Saneyasu Yamaguchi, and Ryousei Takano. 2019. An Accurate Packet Loss Emulation on a DPDK-based Network Emulator. In *Proceedings of the 15th Asian Internet Engineering Conference (AINTEC '19), August 7-9, 2019, Phuket, Thailand. ACM, New York, NY, USA, 8 pages.*

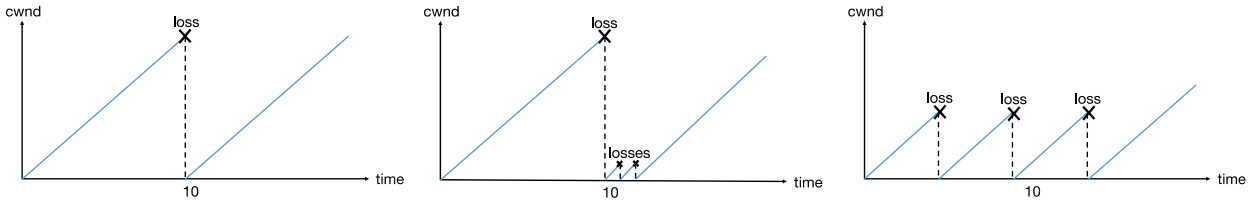
AINTEC '19, August 7–9, 2019, Phuket, Thailand

ACM ISBN 978-1-4503-6849-0/19/08 $15.00
https://doi.org/10.1145/3340422.3343635

1 Introduction

Packet loss is one of the major QoS parameters for network services such as 5th generation (5G) mobile networking. Therefore, many researchers have proposed packet loss models and methodologies for simulation and emulation studies [1]. For example, the 5G mobile network is a network with high throughput and relatively large number of packet loss caused by transmission errors in the air. On high-speed wide area networks, even a small amount of packet losses can have a huge impact on TCP performance [2]. TCP is the most widely used transport protocol on the Internet. Most services, such as Web browsing, email, and Social networking service (SNS) rely on this protocol. A network emulator is a useful tool for testing the behavior and performance of such applications under various network conditions such as a long-fat pipe, lossy network. It is sometimes costly and difficult to reproduce such behaviors in an emulated environment. Accuracy is critical in these tools to obtain reliable and reproducible results from experiments.

A hardware-based emulator is normally equipped with custom-made ASICs or FPGAs, which enable highly-accurate emulation of a wide range of network links including 10 Gigabit Ethernet (10GbE) and faster interconnect technologies. For example, GtrcNET-10p3 (a successor of GNET-1 [3]) supports three 10GbE ports and controls the transmission timing of each packet very accurately (i.e., 25.6 nanosecond precision). The major drawbacks are the cost of the product and the flexibility. Since most hardware-based emulators are available as a proprietary commercial product, users cannot flexibly customize the features of these emulators.
Software-based emulators, including NetEm [4], dummynet [5], NIST Net [6] and WANem [7], are widely used by researchers and engineers, which are implemented as a feature of the network stack of operating systems. It is possible to use commodity server computers for emulation and set up an emulation environment quickly without extra hardware cost. Conventional software-based tools have problems on the throughput of packet forwarding and the accuracy of emulation. DEMU [8] addresses this problem by



(a) Behavior of congestion window (cwnd) in a loss-based congestion control algorithm

(b) Cwnd under burst packet losses

(c) Cwnd under intermittent packet losses

Figure 1: Congestion window in a loss-based congestion control algorithm with burst and random packet loss cases

using a high-performance user-land network framework, i.e., DPDK, and it can emulate latencies on the order of 10 microseconds for short-packet traffic at the 10GbE line rate with high accuracy. It only supported network latency emulation.

We have implemented a packet loss emulation based on Gilbert-Elliott model in DEMU, and we have demonstrated its effectiveness through several experiments on a 10 GbE environment.

Our contributions are summarized as follows: (1) we design and implement an accurate and flexible packet loss emulator in the Linux operating system, (2) we demonstrate an empirical method to measure the impact of burst packet losses on TCP performance by using this network emulator, (3) the performance of a loss-based TCP congestion control algorithm increases in proportion to the burstiness of packet losses even if the packet loss probability is the same, and (4) TCP offload engine can degrade the accuracy of packet loss emulation.

The remainder of the paper is organized as follows. Section 2 describes related work. In Section 3, we explain the design and implementation of a packet loss emulation in DEMU. Section 4 shows performance evaluations on a 10 GbE environment. Finally, Section 5 concludes the paper and briefly mentions future work.

2 Related Work

2.1 Software-based Network Emulator

A network emulator enables us to evaluate the real applications with real-world network conditions in a laboratory. It emulates various network conditions including packet losses, reordering, duplication, corruption, transmission delay, and bandwidth limitation. Several software-based network emulators have been proposed [4][5][6][7][8]. NetEm is implemented in the Linux kernel and R. Lübke et al. [9] has reported that it achieves higher accuracy compared with other existing software-based emulators.

This paper focuses on packet loss emulation. The existing empirical studies assume that packets are randomly dropped with a uniform distribution [10]. Although random loss can provide a worst-case scenario, this assumption is not realistic because burst packet loss is reported to be more characteristic of network performance [11].

Aketa et al. [8] reported that the accurate network emulation with existing network stack becomes difficult on high-speed networks, e.g., 10GbE, 40GbE, 100GbE. This is because a conventional network protocol stack is the bottleneck in performance. Instead of using generic packet processing in the Linux kernel, DEMU implements a specialized packet processing using Data Plane Development Kit (DPDK) [12] to achieve highly accurate network emulation, however it only emulates network latency. We have implemented a packet loss emulation according to the Gilbert-Elliott model in DEMU.

XNetEm [13] addresses the same problem by using eXpress Data Path (XDP) instead of DPDK. XDP provides a high-performance, programmable data path while still leveraging the Linux kernel. A network emulation function is implemented as a kernel plugin that is compiled into eBPF code. The eBPF code is evaluated when packets are received. XNetEm provides packet loss, corruption and marking at high data rates. However, it has several limitations for implementing various network emulation functionalities due to the lack of the programmability of XDP and eBPF. For example, XNetEm cannot easily implement delay, reordering, and bandwidth emulation, because there is no way to handle a timer event.

In addition, recent Ethernet controllers introduce hardware-based TCP offload engine mechanisms such as TCP Segmentation Offload (TSO) and Large Receive Offload (LRO) to reduce CPU load for per-packet processing by aggregating multiple incoming/outgoing packets into a large packet. The Linux kernel also implements similar techniques in the network protocol stack as Generic Segmentation Offload (GSO) and Generic Receive Offload (GRO). They are enabled by default in recent major Linux distributions, but these features have a harmful side effect for packet capturing as mentioned in [14]. There is no existing work to investigate such a side effect on network emulation.

2.2 TCP Performance under Burst Loss Condition

There is a relationship among the TCP congestion control algorithm, burstiness of packet losses, and performance. The congestion control algorithms are divided into two types: loss-based one and delay-based one. A loss-based congestion control algorithm detects network congestion by packet losses and shrinks the congestion window (cwnd) to regulate the transmission rate. NewReno and CUBIC TCP [15] are popular loss-based congestion control algorithm. Although new delay-based and hybrid congestion control algorithms like Compound TCP and TCP BBR [16] were proposed recently, loss-based congestion control algorithms are still dominant in major device platforms. Recently, CUBIC TCP is used as the default congestion control algorithm in Linux, Android, Windows 10, macOS, and iOS. The performance of a loss-based congestion control algorithm largely depends on the burstiness of packet loss. Therefore, accurate and flexible packet

loss emulation is useful for the evaluation of TCP performance in various network conditions. This paper empirically demonstrates this behavior in a network emulation environment.

A loss-based congestion control algorithm linearly increases cwnd and exponentially decrease cwnd when congestion is detected, regardless of the number of lost packets, which is called Additive increase/multiplicative decrease (AIMD). Figure 1 illustrates this behavior. Note that we ignore Fast Recovery and Slow Start mechanisms to simplify the explanation. If one packet is lost at 10, the cwnd decreases significantly once as shown in the part of (a). If three packets are lost at 10, the cwnd decreases significantly once as shown in (b), similar to (a). If a packet is lost three times at 5, 10, and 15, the size decreases significantly three times as shown in (c). Comparing the cases of three packet losses of (b) and (c), we can see that the cwnd with burst packet losses is much higher than the case with many isolated packet losses.

3 Packet Loss Emulation in DEMU

3.1 Packet Loss Model

The Gilbert-Elliot model [17][18] is a well-known and general packet loss model, and it is implemented in several network emulators such as NetEm. This model has two states: G (Good) and B (Bad), and four independent parameters: h, k, p, r, as illustrated in Figure 2. The Good state transitions to the Bad state with a probability p; the Bad state transitions to the Good state with a probability r. $1-k$ and $1-h$ are loss probabilities in the Good and Bad state, respectively. An isolated packet loss can occur in the Good state, and a burst packet loss can occur in the Bad state. For example, if $k = 1$ packets are never lost in the Good state; if $h = 0$ packets are always lost in the Bad state.

The packet loss probability P_{loss} and the mean burst size $E(B)$ are given by the following equations [19]. Here we define *a burst loss size* as the number of continuous packet losses.

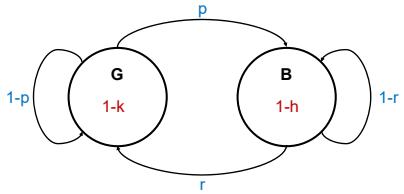


Figure 2: Gilbert-Elliott model

$$P_{loss} = \frac{r}{r+p}(1-k) + \frac{p}{r+p}(1-h)$$

$$E(B) = \frac{(1-h+r^2h)}{r(1-h+rh)}$$

The parameter p is given from the first equation.

$$p = \frac{r(1-k-P_{loss})}{P_{loss}+h-1}$$

Here we set $k = 1$ and $h = 0$ to make it easy to understand.

$$p = \frac{rP_{loss}}{1-P_{loss}}$$

$$E(B) = \frac{1}{r}$$

Therefore, the burst loss size becomes larger as r becomes smaller. For example, if $P_{loss} = 0.01$ and $r = 0.2$ then $p = 0.002$. The mean burst size is 5 packets in this case. Furthermore, if we set $r = 1-p$, then this configuration is equivalent to the random loss model.

3.2 DEMU: DPDK-based network EMUlator

DEMU is a software-based network emulator implemented as a DPDK application. DPDK is the Data Plane Development Kit that consists of libraries to accelerate packet processing workloads and user-space network card drivers running on a commodity hardware. A DPDK application runs as a user-space application using the pthread library, and it can process packets through a well-defined API without the intervention of the operating system kernel.

Figure 3 illustrates the overview of the emulation mechanism, where a series of packets is forwarding from NIC 0 to NIC 1. DEMU is composed of three types of threads: RX thread, worker thread, and TX thread. Normally two sets of three threads are used for bi-directional network emulation. Each thread is running on a dedicated logical CPU core, and it executes a busy loop to poll packets to be processed. As shown in Figure 3, DEMU 1) receives a packet from NIC 0 and records the arrival time, 2) drops the packet based on the given model and parameters, 3) keeps the packet in RX ring buffer until the transmission time, and 4) transmits the packet to NIC 1. The details of this process are described below.

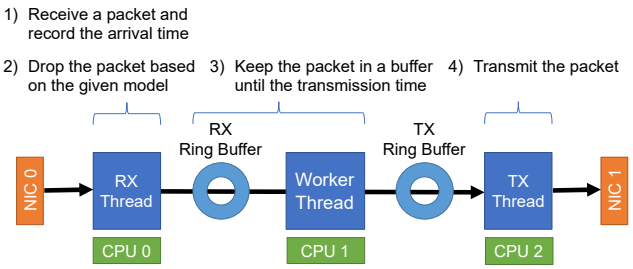


Figure 3: Design of DEMU

Figure 4 shows the pseudo code of the main loop of the RX thread. The RX thread receives at most 32 packets from NIC at a time by calling the *rte_eth_rx_burst()* function of the DPDK API. The RX thread makes a decision to drop each packet according to a packet loss model as described in Section 3.3. The *loss_event()* function implements several packet loss models and it returns a Boolean value, where the return value is true when the packet is dropped. If the packet is dropped, we just skip to send and discard it. Otherwise, the RX thread records the arrival time of each packet in a corresponding *rte_mbuf* object that is similar to the *sk_buff* in the Linux kernel. Then, the packet is forwarded to a worker thread via an RX ring buffer.

The worker thread dequeues packets from the RX ring buffer by calling the *rte_ring_dequeue_burst()* function. The worker thread continuously compares the current time and the scheduled transmission time (the arrival time plus the emulation delay) of the next packet, and it enqueues the packet to the TX ring buffer at scheduled transmission time.

The TX thread dequeues packets from the TX ring buffer and immediately transmits them to the NIC by calling the *rte_eth_tx_burst()* function.

```
static void demu_rx_loop(unsigned portid)
{
   struct rte_mbuf *rbuf[PKT_BURST_RX], *lbuf[PKT_BURST_RX];

   /* Receive packets and store them into rbuf. */
   nb_rx = rte_eth_rx_burst(…, rbuf, …);

   cnt = 0;
   for (i = 0; i < nb_rx; i++) {
      if (loss_event()) { /* Drop a packet based on the given model */
         cnt++;
         continue;
      }
      lbuf[i-cnt] = rbuf[i]; /* Pack the gap */
      lbuf[i-cnt]->udata64 = rte_rdtsc(); /* Record the arrival time */
   }

   /* Enqueue the packets into RX ring buffer */
   rte_ring_sp_enqueue_burst(…, lbuf, nb_rx-cnt, …);
}
```

Figure 4: Pseudo code of packet loss emulation

3.3 Packet Loss Emulation

DEMU supports two packet loss modes: random packet loss and burst packet loss.

The random packet loss mode decides to drop a packet when a pseudo random number is less than a given threshold, i.e., the packet loss probability parameter. This pseudo random number is extracted of the *rte_rand()* function that generates pseudo random numbers with uniform distribution. Note that the minimal packet loss probability is 10^{-7}.

We use the Gilbert-Elliott model to emulate the burstiness of packet losses. As shown in Section 3.1, this model has two states and four independent parameters: *h, k, p, r*. The user should set these parameters as command line parameters.

4 Experiments

4.1 Experimental Setting

We set up four Linux machines that were connected through 10GbE. The machines were desktop class PCs as shown in Figure 5. A network emulator, i.e., DEMU or NetEm, was running on one of them. DEMU is implemented on DPDK version 17.11.3. We only emulated packet losses except for those described in Section 4.3. The other two machines were used for sender and receiver programs. We used Iperf3 for generating TCP traffic. We used CUBIC TCP, which is the default TCP congestion control algorithm in Linux. Hyper threading was enabled. Hugepage is enabled for DPDK, and we assigned 12000 2MB-pages. We added the kernel command line parameter isolcpus=2-7 to exclusively allocate CPU cores for DEMU and the Linux kernel. By the default configuration, the both TSO, GSO, and GRO were enabled and the LRO was disabled on all machines. We changed the GRO configuration on an emulator machine. Note that DEMU does not use GSO and GRO because they are implemented in the Linux kernel. Table 1 summarizes the specifications of each machine.

Table 1 The specification of Linux machines

	Network Emulator	sender/receiver	
Processor	Intel Core i7-8700 (6-core, 3.20 GHz)	Intel Core2 Quad (4-core, 2.66GHz)	
Memory	32GB DDR4 2666MHz	16GB DDR2 800MHz	
NIC	Intel X520-DA2 + 10GBASE-SR/LR SFP+ transceiver	Intel X520-DA2 + 10GBASE-SR/LR SFP+ transceiver	
OS	Ubuntu 18.04 (Linux kernel 4.15.0)	CentOS 7.5.1804 (Linux kernel 3.10.0)	Ubuntu 18.04 (Linux kernel 4.15.0)
DPDK	17.11.3	-	

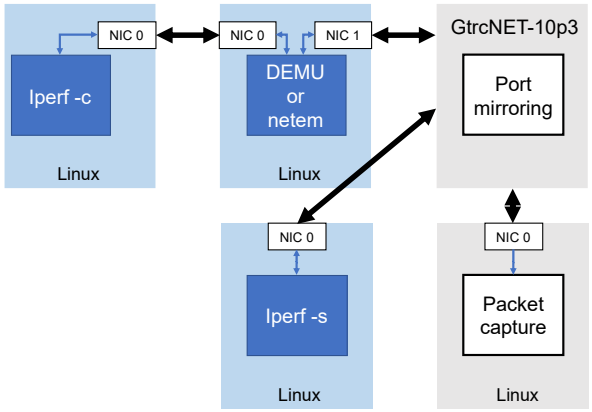


Figure 5: Experimental settings

We also used a hardware-based network testbed GtrcNET-10 [3] for capturing packets at 10 gigabit wire-speed. GtrcNET-10 has three 10GbE ports, and it is located between a receiver machine and a network emulator machine. The other port was used for transferring captured packets to a data analysis machine. In the machine, we observed the gap of TCP sequence numbers in each loss event from the captured data, and we calculated how many packets are dropped in a single loss event, i.e., a burst loss size.

4.2 Impact of Burst Packet Losses on TCP Performance

4.2.1 Verification of the Gilbert-Elliott model in DEMU

In this experiment, we demonstrate the accuracy of burstiness of packet losses in DEMU by measuring the burst loss size when Iperf3 was running for 30 seconds. We varied the parameter r from 0.05 to 0.9 with a packet loss probability of 10^{-4}. The parameter r indicates the transition probability from the Bad state to the Good state in the Gilbert-Elliott model as described in Section 3.1. It controls the burstiness of packet loss in such way that the burstiness increases as r decreases. Figure 6 shows CDF curves of the burst packet sizes. It is expected that a curve approaches the upper left corner as the burstiness decreases; it approaches the bottom right corner as the burstiness increases. The result shows that DEMU behaves as we have assumed.

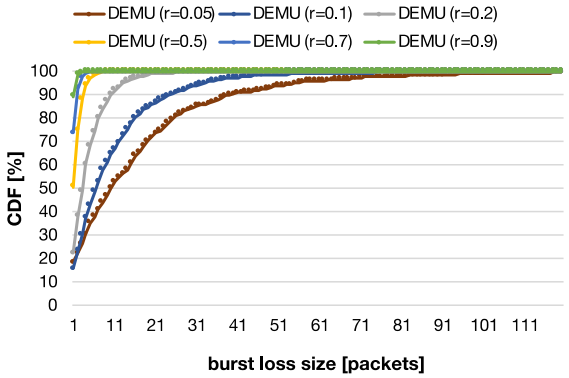


Figure 6: Comparison of the burst loss size under burst packet loss conditions. The parameter *r* is configured from 0.05 to 0.9.

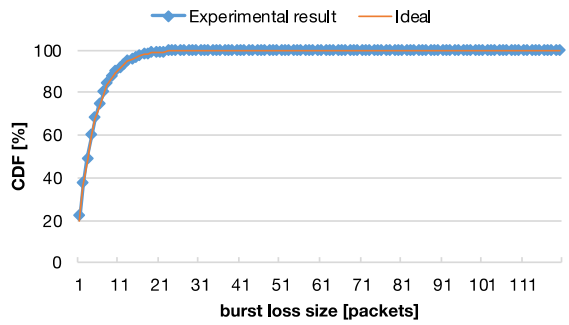


Figure 7: Comparison of the burst loss size in DEMU (r=0.1) between the experimental result and the ideal value.

We focus on one simple case to discuss the accuracy in detail. Figure 7 compares the ideal and measured CDFs, where $r = 0.1$. The ideal probability density function (PDF) of burst loss size can be calculated by $p(n) = r \times (1-r)^{(n-1)}$ and the corresponding CDF can be expressed as the integral of its PDF. Figure 7 shows that both curves are almost identical. The average error of the measured burst loss sizes from the ideal values is 1.62×10^{-2}%. As shown in Table 2, the average error decreases as r increases.

With respect to the accuracy of mean burst loss size, i.e., *E(B)*, in DEMU, in this experiment, the ideal *E(B)* equals to $1/r$ as described in Section 3.1. Table 3 compares the ideal *E(B)* with the measured *E(B)* obtained in the previous experiment. The measured *E(B)s* are almost equivalent to the ideal *E(B)s* except for when $r = 0.05$. This gap is due to the sampling number of packet loss events being small compared with the other cases resulting in a lager error. The number of packet loss events is about 600 in the case of $r = 0.05$. We measured E(B) when Iperf3 was running for 360 seconds again, and we obtained the measured *E(B)* is 19.4. In Table 3, $r = 0.05^{*}$ denotes this case.

Table 2 Error of the experimental burst loss size from the ideal value

Configurations	Error [%]
DEMU (r=0.01)	3.36×10^{-1}
DEMU (r=0.05)	6.93×10^{-2}
DEMU (r=0.1)	1.62×10^{-2}
DEMU (r=0.2)	1.50×10^{-3}
DEMU (r=0.5)	1.21×10^{-3}
DEMU (r=0.7)	1.22×10^{-3}
DEMU (r=0.9)	1.25×10^{-3}
DEMU (r=1)	6.52×10^{-4}
DEMU (random)	8.67×10^{-4}
NetEm (w/GRO)	2.65×10^{-1}

Table 3 Mean burst loss size *E(B)* in DEMU

Parameter *r*	0.05	0.05*	0.1	0.2	0.5	0.7	0.9
Ideal *E(B)*	20	20	10	5	2	1.43	1.11
Measured *E(B)*	16.9	19.4	10.7	5.02	1.97	1.57	1.13
Measured/Ideal	0.84	0.97	1.07	1.00	0.99	1.10	1.02

4.2.2 TCP goodput under various packet loss conditions

We investigate the impact of both the packet loss probability and the burstiness on TCP performance. We measured TCP goodputs with Iperf3 for 300 seconds under several emulated packet loss conditions by using NetEm and DEMU. The average TCP goodput was calculated from five trials for each condition.

The results are shown in Figure 8. We used CUBIC TCP. Thus, TCP goodput should decrease as the packet loss probability increases. All results follow this expectation. TCP goodputs were almost the same when the packet loss probabilities were ≤ 0.001. Focusing on NetEm, the goodputs are quite different whether GRO is enabled or disabled. This is because the burstinesses are different. We describe this in section 4.4. On the other hand, the TCP goodputs of DEMU depends on the parameter *r*, which is a factor of the burstiness of packet loss. The burstiness linearly increases as the parameter r decreases. When the loss probability is 10^{-4}, the TCP goodputs except for DEMU (r=0.05), DEMU (r=0.1), and DEMU (r=0.2) become by almost half. In addition, TCP goodput of DEMU (random) is quite similar to that of DEMU (r=1). DEMU

(random) is theoretically equivalent to DEMU (r=1-p), therefore, these results become almost equivalent when the packet loss probability is set to small enough. We also found an unexpected performance drop when we used DEMU (r=0.01) with the loss probability of 10^{-4}. We will discuss about it later.

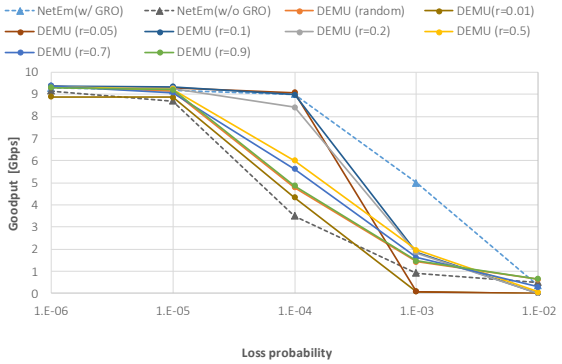


Figure 8: TCP goodput under several emulation settings: NetEm, DEMU with the random loss mode, and DEMU with the burst loss mode. The loss probability is configured from 10^{-6} to 10^{-2}.

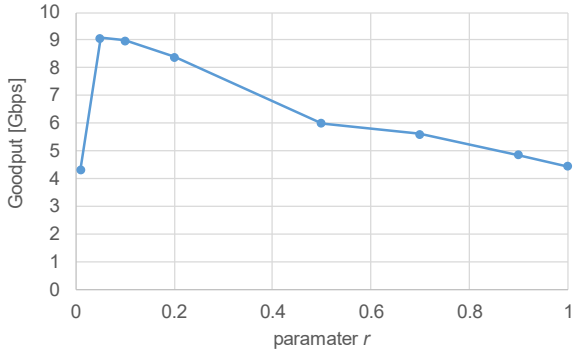


Figure 9: TCP goodput with the loss probability of 10^{-4}. The parameter *r* was configured from 0.01 to 1

We describe the impact of the burstiness of packet losses on TCP performance in detail. Figure 9 shows the relationship between TCP goodputs and the parameter *r*, where the loss probability is 10^{-4}. Generally speaking, TCP goodput increases in proportion to the burstiness of packet loss. However, the TCP goodput with $r = 0.05$ significantly decreases. In this case, the TCP communication was completely stalled, and it never recovered again. This phenomenon is the same as DEMU (r=0.05) with the loss probability in Figure 9. We believe that this was because a heavy retransmission timeout occurred and will be investigated further in the future.

4.3 Effectiveness of TCP Offload Engine

This subsection investigates the effectiveness of TCP Offload Engine (TOE): TCP Segmentation Offload (TSO), Generic Segmentation Offload (GSO), and Generic Receive Offload (GRO). We show that the obtained goodput strongly depends on whether GRO is enabled. In this experiment, we used NetEm and conducted five scenario experiences, where the combinations of TCP offload engine are configured at both NIC0 and NIC1 in the emulator machine. We refer the configurations as *w/ all* (all TOEs are enabled), *w/o GRO* (GRO is disabled), *w/o GSO* (GSO is disabled), *w/o TSO* (TSO is disabled), and *w/o all* (all TOEs are disabled). The emulated network delays were 0, 5 or 10 milliseconds. Iperf3 was running for 300 seconds and the average TCP goodput is calculated from five trials. Figure 10 and Figure 11 show the average goodputs with the maximum window sizes of 4 MB (the default configuration in the Linux kernel) and 100 MB, respectively.

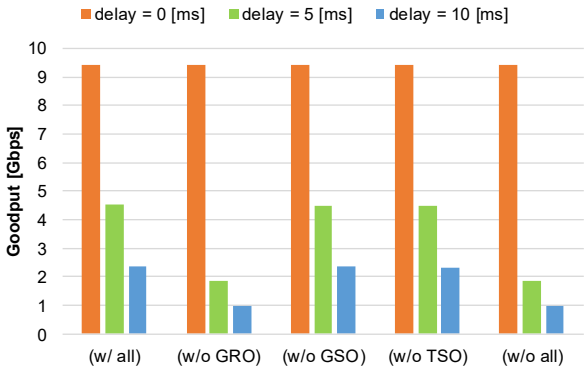


Figure 10: TCP goodput under five TCP offload engine settings. The maximum window size is 4MB.

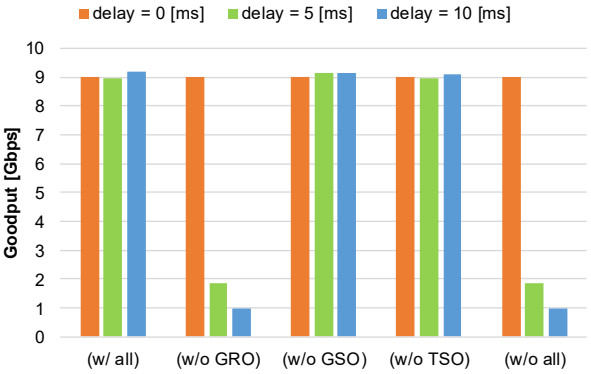


Figure 11: TCP goodput under five TCP offload engine settings. The maximum window size is 100MB.

Figure 10 shows that the goodput remarkably decreased as the delay increased in the case of the default window size. While the goodput reached close to the 10Gbps wire rate throughput with 0 millisecond delay, the goodputs with 5 and 10 milliseconds were much less than the wire rate throughput. This is mainly because the windows size was smaller than the bandwidth-delay product (BDP). Focusing on the TOE, a receiver side mechanism, i.e., GRO, significantly improves the performance when the delay was not 0 millisecond. The goodput was reduced by half when GRO was disabled. On the other hand, the effectiveness of sender side mechanisms, i.e., TSO and GSO, was negligible.

Figure 11 shows that the goodputs were not degraded regardless of the delay, except when GRO was disabled because the maximum window size of 100 MB was large enough to fill the BDP, i.e., 12.5 MB. However, the result of w/o GRO is quite similar in Figure 10. The performance degradation of w/o GRO is caused by packet losses in the emulator machine. In the case of w/ GRO, there were no packet losses regardless of delay. In the case of w/o GRO, however, the number of packet losses was proportional to the delay as shown in Figure 12. Through these experimental results, the advantage of GRO is clear from a performance standpoint and it is crucial when NetEm emulates a large BDP network.

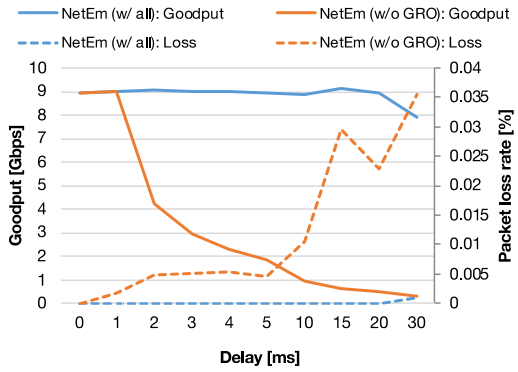


Figure 12: Performance degradation of NetEm (w/o GRO) when the emulated delay becomes large.

4.4 Negative Impact of GRO for Accurate Packet Loss Emulation

We investigate the impact of GRO for the accuracy of packet loss emulation. In this experiment, we measured TCP goodput under a random packet loss condition emulated by NetEm (w/ GRO), NetEm (w/o GRO), and DEMU, where the packet loss probability is 10^{-4}. We have observed that the average goodput is significantly different whether GRO is enabled or disabled, as shown in Figure 13. The goodput of NetEm (w/ GRO) was around 9 Gbps while those of NetEm (w/o GRO) and DEMU were around 4 Gbps.

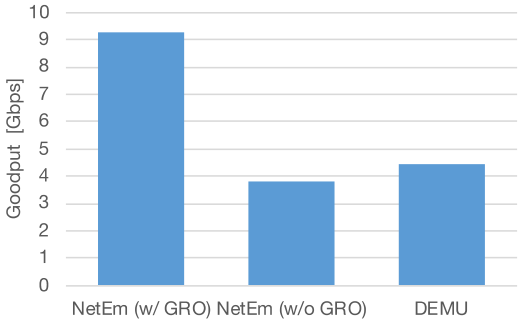


Figure 13: TCP goodput under a random packet loss condition. The packet loss probability is 10^{-4}

We investigate where the difference comes from despite the total number of packet losses were almost the same in all cases. Our assumption is that GRO makes the burst loss size larger. NetEm emulates a packet loss in such way that it drops an aggregated large packet by GRO instead of a single normal packet, and eventually, multiple packets are dropped in a single loss event. Figure 14 depicts the cumulative distribution function (CDF) of the burst loss size in a packet loss event. The burst loss size should be one in a random packet loss condition. The CDF curve of NetEm (w/ GRO) shows the Poisson distribution. On the other hand, the CDF curve of NetEm (w/ GRO) holds on to 100%. This result indicates that DEMU emulates random packet losses correctly, namely almost all of the burst loss size is one. On the contrary, NetEm (w/ GRO) does not emulate random packet losses correctly, and many burst losses occurred as we had expected above. Moreover, we analyze the error of the measured burst loss sizes from the ideal values. The average errors of DEMU and NetEm (w/ GRO) are 8.67×10^{-4} and 2.65×10^{-1}, respectively. In other words, the accuracy of DEMU is 306 times higher than that of NetEm (w/ GRO).

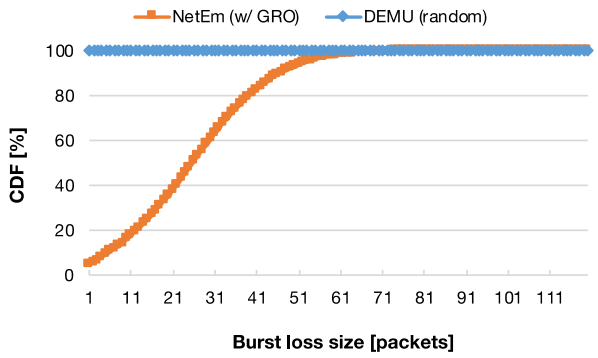


Figure 14: Comparison of the burst loss size between NetEm (w/ GRO) and DEMU under a random loss condition

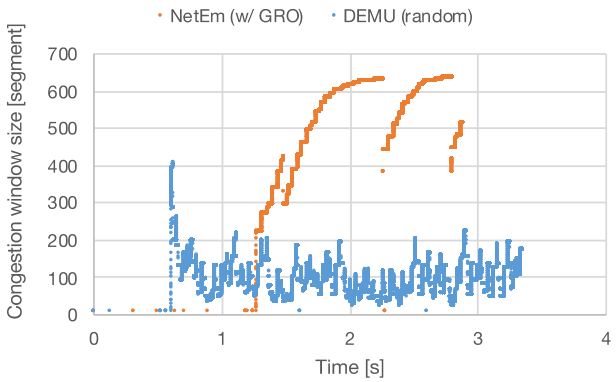


Figure 15: Comparison of the variation of TCP congestion window between NetEm (w/ GRO) and DEMU.

Inaccuracy of emulated burstiness makes it difficult to achieve correct performance evaluation. As shown in Figure 15, the behaviors of the cwnd in the sender machine when DEMU or NetEm (w/ GRO) were running in the emulator machine. DEMU generated many isolated packet losses. The OS frequently detected packet loss events resulting in a low cwnd. On the other hand, NetEm (w/ GRO) dropped multiple packets in each packet loss event. The OS detected fewer packet loss events, and the cwnd was increased compared to DEMU. Therefore, the burstiness results in the difference of TCP performance even though the total number of packet loss were the same in both cases.

5 Conclusion and Future Work

We have designed and implemented an accurate packet loss emulation in a DPDK-based network emulator, DEMU. We have demonstrated the effectiveness through experiences on a 10GbE network. For random packet loss emulation, DEMU is able to emulate packet losses with 305 times higher accuracy compared with NetEm (w/ GRO).

As described in Section 4.2, the TCP goodput depends on the degree of burstiness of packet losses. A loss-based congestion control algorithm largely reduces its congestion windows size when it detects of a series of packet losses. Although random loss can provide a worst-case performance with a given packet loss probability, burst packet loss is more characteristic of network performance. Thus, packet loss emulation with accurate burstiness control is necessary for evaluating the performance of TCP communication. To the best of our knowledge, there is no empirical study that investigates burst packet loss in a software-based network emulator. DEMU enables us to comprehensively evaluate the impact of burst packet losses.

We also present doubts regarding the use of TCP offload engine in a software-based network emulator. Through our experiment, we conclude that GRO is harmful for accurate packet loss emulation. Although GRO is useful to improve packet processing throughput over a large bandwidth-delay product network, as shown in Section 4.3, it introduces an unexpected effect into a packet loss emulation as discussed in Section 4.4.

The functionalities and the usability of DEMU are not enough for practical use. For future work, we plan to implement a function to replay packet losses that are recorded on an actual network for conducting more realistic and reliable experiments. Other future work is the dynamic re-configuration of parameters at runtime instead of restarting with the different parameters.

The source code is available from https://github.com/ryousei/demu.

REFERENCES

[1] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. 1999. "Measurement and modeling of the temporal dependence in packet loss." 18th IEEE International Conference on Computer Communications (INFOCOM'99), pp.345-352.

[2] ESnet. 2018. "Network Tuning: TCP Issues Explained: Packet Loss." https://fasterdata.es.net/network-tuning/tcp-issues-explained/packet-loss/

[3] Y. Kodama, T. Kudoh, R. Takano, H. Sato, O. Tatebe, and S. Sekiguchi. 2004. "GNET-1: Gigabit Ethernet network testbed." IEEE International Conference on Cluster Computing (CLUSTER'04), pp.185–192.

[4] S. Hemminger. 2005. "Network emulation with NetEm." linux.conf.au, pp. 18-23.

[5] L. Rizzo. 1997. "Dummynet: a simple approach to the evaluation of network protocols." ACM SIGCOMM Computer Communication Review 27.1, 31-41.

[6] M. Carson and D. Santay. 2003. "NIST Net: a Linux-based network emulation tool." ACM SIGCOMM Computer Communication Review 33.3, pp.111-126.

[7] H. K. Kalitay and M. K. Nambiarz. 2011. "Designing WANem: A Wide Area Network emulator tool." Third International Conference on Communication Systems and Networks (COMSNETS'11), pp. 1-4.

[8] S. Aketa, T. Hirofuchi and R. Takano. 2017. "DEMU: A DPDK-based network latency emulator." IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN'17), pp. 1-6.

[9] R. Lübke, P. Büschel, D. Schuster and A. Schill. 2014. "Measuring accuracy and performance of network emulators." IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom'14), pp. 63-65.

[10] A. Jurgelionis, J. Laulajainen, M. Hirvonen and A. I. Wang. 2011. "An Empirical Study of NetEm Network Emulation Functionalities." 20th International Conference on Computer Communications and Networks (ICCCN'11), pp. 1-6.

[11] A. Clark. 2003. "Packet Loss Distributions and Packet Loss Models." ITU-T Contribution Com 12-D97-E.

[12] Intel Data Plane Development Kit (DPDK). http://dpdk.org/

[13] S. Hemminger. 2017. " XNetEm Network Emulation using XDP." NetDev 2.2, pp. 1-7

[14] S. Pachghare. 2012. "TOE: TCP Offload Engine on NIC & Packet Capture Misinterpretations." http://www.thesubodh.com/2012/05/toe-tcp-offload-engine-on-nic-packet.html

[15] S. Ha, I. Rhee, and L. Xu. 2008. "CUBIC: a new TCP-friendly high-speed TCP variant." ACM SIGOPS operating systems review 42.5. pp.64-74.

[16] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. 2016. "BBR: Congestion-based congestion control. "

[17] E. N. Gilbert. 1960. "Capacity of a burst-noise channel." The Bell System Technical Journal 39.5, pp.1253-1265.

[18] G. Haßlinger and O. Hohlfeld. 2008. "The Gilbert-Elliott model for packet loss in real time services on the Internet." 14th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB'08), pp. 1–15.

[19] S. Salsano, F. Ludovici, A. Ordine, and D. Giannuzzi. 2012. "Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel." http://netgroup.uniroma2.it/NetemCLG