DEMU: A DPDK-based Network Latency Emulator

Shuhei Aketa
Ritsumeikan University
Shiga, Japan
Email: saketa@asl.cs.ritsumei.ac.jp

Takahiro Hirofuchi* and Ryousei Takano
National Institute of Advanced Industrial
Science and Technology (AIST)
Tsukuba, Japan
Email: {t.hirofuchi, takano-ryousei}@aist.go.jp

***Abstract*—A network latency emulator allows IT architects to thoroughly investigate how network latencies impact workload performance. Software-based emulation tools have been widely used by researchers and engineers. It is possible to use commodity server computers for emulation and set up an emulation environment quickly without outstanding hardware cost. However, existing software-based tools built in the network stack of an operating system are not capable of supporting the bandwidth of today's standard interconnects (e.g., 10GbE) and emulating sub-milliseconds latencies likely caused by network virtualization in a datacenter. In this paper, we propose a network latency emulator (DEMU) supporting broad bandwidth traffic with sub-milliseconds accuracy, which is based on an emerging packet processing framework, DPDK. It avoids the overhead of the network stack by directly interacting with NIC hardware. Through experiments, we confirmed that DEMU can emulate latencies on the order of 10 µs for short-packet traffic at the line rate of 10GbE. The standard deviation of inserted delays was only 2-3 µs. This is a significant improvement from a network emulator built in the Linux Kernel (i.e., NetEm), which loses more than 50% of its packets for the same 10GbE traffic. For 1 Gbps traffic, the latency deviation of NetEm was approximately 20 µs, while that of our mechanism was 2 orders of magnitude smaller (i.e., only 0.3 µs).**

I. Introduction

In today's design process of data center networks, a tool to emulate network latencies is essential to find the best combination of various networking technologies. In order to achieve optimal performance of workloads running inside of a data center, it is necessary to achieve desired network latencies between particular communication endpoints in local area networks. A network latency emulator allows IT architects to thoroughly investigate how network latencies impact workload performance and then determine what latency values are acceptable and desired. From the viewpoints of the other network components, a network latency emulator can be seen as just a normal network link, but it behaves as if the network has a specific communication delay. IT architects can repeatedly try different network latencies without time-consuming preparation for experiments.

We consider that the advent of network virtualization also justifies the importance of network latency emulation techniques. In the notion of network function virtualization (NFV), network functions (e.g., network switch/router, load balancer, firewall and intrusion detection system) are implemented as software-based mechanisms on general-purpose hardware, instead of ASIC-based hardware appliances dedicated to each function. Although this software-based mechanism enables great flexibility in network configuration, the period of time to process a packet is orders of magnitude greater than ASIC-based systems. For example, we experienced that a firewall NFV appliance based on Lagopus [1] increases node-to-node latency by 40 to 60 µs. Similarly, an Open vSwitch [2] node adds 35 µs to network latency even when forwarding only 100 Mbps traffic [3], and a Linux router adds at least 20-30 µs [4]. In addition, virtualized network functions are sometimes chained to create a network service. A network slice created by programmable network infrastructure sometimes overlays multiple physical network segments. These also likely degrade end-to-end network latencies in a data center network. Even though network latencies are just tens of microseconds larger, latency-sensitive workloads very possibly suffer serious performance loss; including but not limited to high-speed transaction systems such as high frequency trading (HFT) servers, storage area network (SAN) systems serving high-volume random access requests, and high-performance computing tasks involving fine-grained synchronization among computation nodes.

Existing network emulation tools are categorized as hardware-based or software-based. Generally, hardware-based network emulators are equipped with custom-made ASICs or FPGAs, which enable highly-precise emulation of a wide range of network links including 10Gb Ethernet (10GbE) and faster interconnect technologies. For example, our hardware-based network emulator (GtrcNET-10p3, a successor of GNet-1 [5]) supports 10GbE and controls the transmission timing of each packet very accurately (i.e., ± 25.6 ns precision). A drawback of hardware-based emulators is generally that the cost of these products is high (i.e., tens of thousands USD). Since most hardware-based emulators are available as a proprietary commercial product, users cannot flexibly customize the features of these emulators, for example, to automate experiments and to log data.

Software-based emulation tools (e.g., NetEm [6], dummynet [7], NIST Net [8]) 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 outstanding hardware cost. Conventional software-based tools have problems on the throughput

*Corresponding Author

978–1–5386–0728–2/17/$31.00 ©2017 IEEE

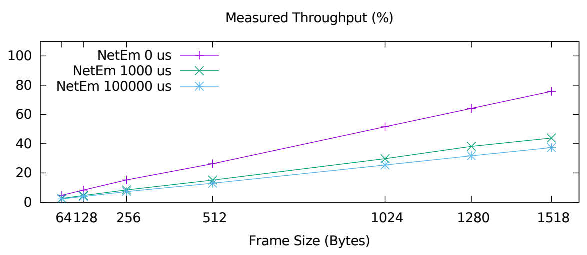


Fig. 1: Packet forwarding performance of **NetEm** for 10GbE network. The latency of emulation was configured to 0 µs, 1000 µs (1 ms) and 100000 µs (100 ms), respectively. In the case of 0 µs, no latency was introduced to assess bare forwarding performance of the NetEm driver. Measured throughput values are normalized by the theoretical maximum at each specified frame size.

of packet forwarding and the accuracy of emulation. For designing a data center network, it is necessary to investigate application performance impact of sub-milliseconds latency (for example, likely caused by network virtualization) in local area networks. As clearly shown in our experiments (Sections II and IV), however, an existing software-based emulator built in an operating system is not capable of creating such fine-grained latencies, especially for broad bandwidth traffic through today's standard interconnect links (e.g., 10GbE).

In this paper, we propose a network latency emulator supporting broad bandwidth traffic with sub-milliseconds accuracy. Instead of extending the network stack of an operating system, we developed an emulator program that directly interacts with the hardware interface of network interface cards, so as to avoid the overhead of the general-purpose network stack. We reduced the development cost of our emulator by using an emerging packet processing framework (Data Plane Development Kit, DPDK [9]). Through experiments, we confirmed that our DEMU (DPDK-based network latency EMUlator) can emulate latencies on the order of 10 µs for short-packet traffic at the 10GbE line rate with a 2-3 µs standard deviation. In contrast, a network emulator built in the Linux Kernel (i.e., NetEm) cannot perform latency emulation at all for the same traffic, which lost more than 50% of packets due to deficiency in performance. For 1 Gbps traffic, the latency deviation of NetEm was approximately 20 µs, while that of our mechanism was 2 orders of magnitude smaller (i.e., only 0.3 µs).

Section II briefly discusses the limitation of existing software-based emulation tools. Section III presents the design of our emulator and its implementation details. Section IV describes performance evaluation through experiments. Section V compares our study with prior work. Section VI concludes the paper.

II. Problem

Existing software-based network emulation tools have multiple issues including 1) seriously poor throughput incapable of supporting today's standard interconnects and 2) poor accuracy even for small bandwidth traffic. Due to space limitations, we focus this section on the first problem and then discuss the second problem in Section IV-C through comparison with our mechanism.

Through preliminary experiments, we confirmed that an existing software-based network emulation tool cannot meet the packet forwarding performance of 10GbE. We measured packet forwarding performance of NetEm, which is one of the most popular software-based network emulation tools. We set up a machine with two 10GbE NICs, installed Linux Kernel 4.4.0 and configured it to forward packets from one NIC to the other. We used our hardware packet generator to produce the maximum bandwidth of packets and measured the packet forwarding performance of the NetEm machine. See Section IV for details of experimental setup.

Figure 1 shows the throughput of NetEm with different frame sizes (selected as noted in RFC 2544 [10]). Measured values are normalized by the theoretical maximum at each frame size. Even though the configuration with no added latency produced best results, NetEm could not achieve the maximum throughput of 10GbE for the standard frame size (i.e., 1518 bytes), which was only 7.4 Gbps (i.e., 75% of the theoretical maximum). In the case of the shortest frame size (i.e., 64 bytes), it achieved only 0.3 Gbps (i.e., 4.7%).

The existing software-based emulation tool lacks the capability to support 10GbE traffic. It suffers very serious performance loss especially for short-packet traffic. As shown in Figure 1, there was a clear linear correlation between frame size and throughput. This means that the numbers of processed packets per second were roughly the same even though the frame sizes were different. NetEm failed to decrease per-packet processing cost for shorter packets, likely because of high per-packet processing cost [11].

III. DEMU

We carefully designed our latency emulation mechanism to make it capable of forwarding packets at the line rate of a high-speed network (i.e., 10GbE in mind). The emulation accuracy we designed for is the one that should be sufficient to investigate performance impacts of software-based network functions, i.e., sub-milliseconds order or if possible microseconds order. Our latency emulation mechanism directly manipulates RX/TX ring buffers of the NICs without interacting with the protocol stack of the Linux kernel. We used a programming framework for packet processing, DPDK. It provides an abstracted API to hide device-specific control interfaces of NICs. DPDK includes useful programming libraries such as queue and hash functions, which enabled us to save considerable effort to optimize the performance of DEMU.

DEMU runs on a machine with two physical network interfaces, and forwards packets from one NIC to the other NIC and vice versa. In terms of network topology, DEMU works transparently; it inserts a transmission delay when forwarding each packet, but does not modify its content. One of the intended use-cases is that a software developer investigates whether and to what extent a workload is tolerate against a transmission delay of a virtualized network.

In this paper, we focus on the emulation of network latency in data center networks. We envision that the future version of DEMU will be used to study wireless networks or wide-

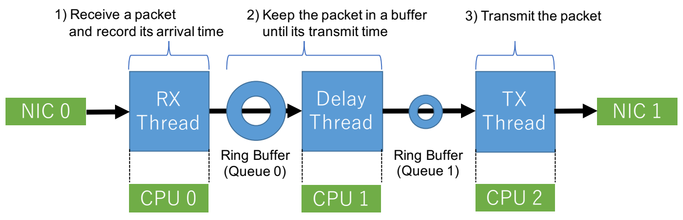


Fig. 2: Design Overview of DEMU

area networks where packet loss and limited bandwidth are also dominant. DEMU will be extended to emulate packet loss and bandwidth limitation by implementing a packet drop mechanism based on stochastic models.

A. Design Overview

Figure 2 illustrates the overview of the emulation mechanism forwarding packets from NIC 0 to NIC 1. Although DEMU supports bi-directional packet forwarding, the figure depicts one-way direction for ease of understanding. To emulate a network delay, DEMU 1) receives a packet and records its arrival time at one NIC, 2) keeps the packet in a buffer and 3) then transmits the packet from the other NIC after the emulation delay elapses. Each step is implemented in a dedicated thread assigned to an individual CPU core, so as to maximize performance on recent multi-core platforms.

Actually, our initial implementation of this mechanism as a single-thread program experienced serious packet loss for 10GbE traffic. We found that the RX ring buffer of a NIC frequently overflowed since the thread could not remove packets from the RX ring buffer while doing other tasks (i.e., enqueuing/dequeuing packets into/from a delay buffer and transmitting packets from the other NIC). Similarly, the TX ring buffer occasionally emptied since the thread failed to continuously supply packets to be transmitted.

B. Threads

A DEMU program is composed of 3 types of threads, i.e., RX thread, delay thread and TX thread. All the threads execute a busy loop to poll packets to be processed. Currently, a pair of one RX thread, one delay thread and one TX thread works for each direction of packet forwarding.

A RX thread retrieves packets from the RX buffer of a NIC by calling the `rte_eth_rx_burst()` function of the DPDK API. The RX thread records the arrival time of packets with the TSC clock of CPU. In the DPDK framework, the information of a packet is maintained in a `rte_mbuf` object that is similar to `sk_buff` of Linux. The program uses a 64bit unsigned integer value of a `rte_mbuf` object for the arrival time.

Next, the thread forwards the packets to a delay thread via a software ring buffer (Queue 0 in Figure 2). The program uses a lock-less queue of DPDK for the software ring buffer. The size of the queue is fixed. When the enqueuing function (i.e., `rte_ring_enqueue_burst()`) fails due to overflow, the thread discards the packet so as to avoid the unexpected increase of an emulated delay. The appropriate size of the queue depends on the bandwidth of a network and the delay of an emulation. For example, a 10GbE network transfers 15M packets per second at maximum. When emulating 1ms delay, DEMU needs to preserve 15K packets in the queue.

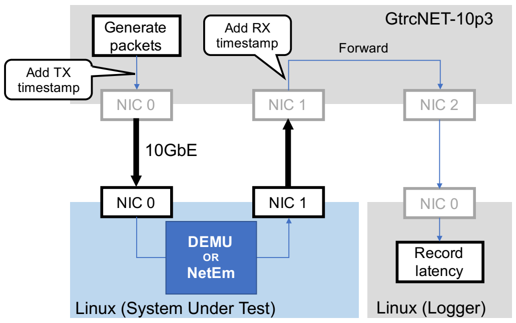


Fig. 3: Network topology of experiments

TABLE I: The specification of the Linux machine (System Under Test)

Processor	Xeon E5-2430L 2.00GHz x 2
Mother Board	Supermicro X9DBL-iF
Memory	DDR3-1333 8GB x 6
NIC	Intel X520-DA2
OS	Ubuntu 16.04.1 (Kernel 4.4.0-42)

A delay thread dequeues packets from the software ring buffer by calling the `rte_ring_dequeue_burst()` function. The delay thread reads the current time from the TSC clock. If the current time exceeds the expected transmission time of the packets (i.e., exceeds the arrival time plus the emulation delay), the thread enqueues the packets to the next software ring buffer (Queue 1). Otherwise, the thread continuously checks the current time again until the transmission time arrives.

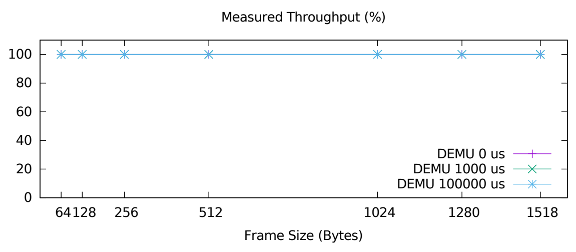
The TX thread dequeues packets from the software ring buffer and immediately transmits them to the NIC by calling the `rte_eth_tx_burst()` function. If the NIC does not consume its TX buffer due to network congestion, the thread fails to transmit packets and discards them.

C. Performance Optimization

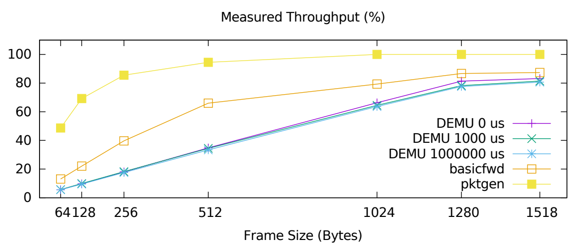
To obtain the best performance, we carefully optimized various software and hardware parameters. Otherwise, we observed sporadic outlier values in measured latencies (e.g., sudden spikes of several milliseconds). The Linux kernel was configured to reserve CPU cores for DEMU; the Linux process scheduler neither executes other tasks or handles hardware interrupts on the reserved CPU cores. All the power saving mechanisms of CPU processors such as DVFS and C-State were disabled. The 1-GB page mode of CPU was used for allocating memory objects of DPDK, instead of the 4-KB and 2-MB page modes.

IV. Evaluation

In this section, the performance of our proposed mechanism was evaluated through various experiments. To demonstrate the advantage of DEMU, we compared both the throughput of packet forwarding and the accuracy of emulated latencies between NetEm and DEMU.



(a) 10GbE network



(b) 100GbE network

Fig. 4: Packet forwarding performance of DEMU. Measured throughput values are normalized by the theoretical maximum at each specified frame size. (a) In the left graph, since DEMU achieved the exact maximum throughput of 10GbE with tested parameters, the lines and the points in the graph overlap. (b) In the right graph, the graph of pktgen shows the performance of the used packet generator. The graphs of DEMU partially overlap because of close values. basicfwd is a DPDK skeleton program simply forwarding frames from one NIC to another.

A. Setup of experiments

We used our hardware-based network test system GtrcNET-10p3[1]. GtrcNET-10 is equipped with three 10GbE interface ports and customized FPGAs. It is configured to generate packets at a given speed and embed timestamps in packets for latency measurement. Figure 3 illustrates the network topology of experiments. First, GtrcNET-10 transmits packets from its first port. When transmitting a packet, it embeds the current time (i.e., TX timestamp) in its payload. The data format of a timestamp conforms to RFC 1305 [12]. These packets pass through a system under test (i.e., DEMU or NetEm in our experiments). After that, GtrcNET-10 receives a packet from the second port and embeds the current time at another field of its payload (i.e., RX timestamp). Then, it forwards the packet to the third port. Finally, a data logger machine captures all the packets and calculates the latency of each packet referring its TX and RX timestamps. The clock resolution of GtrcNET-10p3 is 25.6 ns. We truncated fractional parts of 10 ns in the following results.

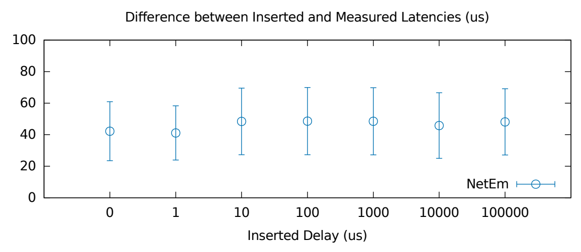
We set up a Linux machine to execute DPDK or NetEm. Table I summarizes the specifications of this machine. The version of DPDK used was 16.7.02. For experiments using NetEm, the Linux machine was configured as an IP router, not an Ethernet bridge. According to [11], the router mode of the Linux network stack achieves better throughput of packet forwarding than the bridge mode; the bridge mode unnecessarily clones descriptor objects, which results in higher CPU cost. We also carefully configured the parameters of NetEm and the Linux network stack to have the best performance.

B. Throughput

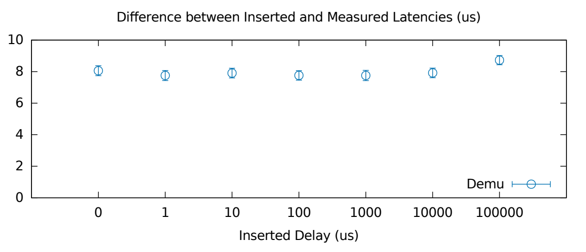
First, we evaluated the packet forwarding performance of NetEm and DEMU. GtrcNET-10 generated Ethernet frames at the maximum throughput of 10GbE and reported the actual throughput achieved by a system under test. Figure 1 shows the results of NetEm. As discussed in Section II, the forwarding performance of NetEm is totally incapable of supporting packet transmission rates in 10GbE networks. Conversely, as shown in Figure 4a, we observed that DEMU achieved the maximum theoretical throughput of 10GbE for each frame size (e.g., 7.143 Gbps for 64-byte frames and 9.844 Gbps for 1518-byte frames). It should be noted that GtrcNET-10 reports measured throughput values in the unit of Gbps and rounds off these values to the third decimal place. As far as we observed with this precision, the forwarding performance of DEMU exactly matched the theoretical maximum. Although the main focus of this paper is the accurate emulation of sub-milliseconds latencies, we increased the inserted latency up to 100 ms to assess how DEMU performs in more severe conditions. The forwarding performance of DEMU never degraded at any tested latencies. In summary, DEMU is capable of handling 10GbE traffic without any packet loss even while emulating any tested network latencies. In terms of forwarding performance, DEMU dramatically outperforms NetEm.

Because 10GbE is not sufficient to measure the maximum limit of DEMU's performance, we conducted supplementary experiments using 100GbE interfaces. We prepared more powerful machines equipped with recent hardware components i.e., dual Intel Xeon E5-2630v4 processors and a Mellanox ConnectX-4 MCX416A-CCAT interface card with dual 100GbE ports. Since GtrcNET-10 does not support 100GbE, one machine is configured as a software-based network tester with Pktgen-DPDK. Another machine is used to run DEMU. We carefully tuned various parameters of the network tester. Figure 4b shows the results of this experiment. Although it was not possible to generate packets at the maximum theoretical throughput of 100GbE, such generated traffic was large enough to measure the maximum throughput of DEMU (See the graph of pktgen in Figure 4b). For 1280-byte and 1518-byte frames, DEMU achieved approximately 80% of the 100GbE bandwidth. The performance of DEMU, however, was seriously degraded for short packets. After investigating the reason of the performance loss by using instrumentation techniques, we found that packet drops occurred at the RX buffer of the NIC due to the high CPU cost of the RX function of the Mellanox DPDK driver (MLX5 PMD). Actually, we observed that a DPDK skeleton program (basicfwd in the figure) suffers similar performance loss for short packets. A study [13] reported more than 80% of a packet drop rate for 64-byte traffic with the same Mellanox NIC. Although further experiments are necessary to discuss the limit of

[1]A hardware-based network emulator, mentioned in Section I, is another feature of GtrcNET-10p3.

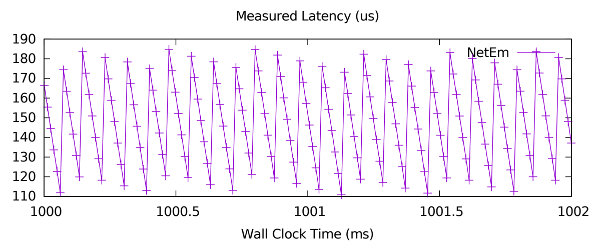


(a) NetEm

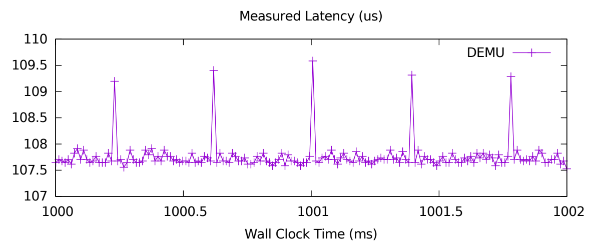


(b) DEMU

Fig. 5: The difference between an inserted delay and its measured latency for 1 Gbps traffic of 1518-byte frames. Error bars show standard deviation. Note the scale of the Y axis.



(a) NetEm



(b) DEMU

Fig. 6: The measured latency of each packet passing through the emulator inserting a 100-us delay in the case of 1 Gbps traffic composed of 1518-byte frames. Data in a period of 2 ms are plotted.

DEMU's performance, DEMU showed potential to support future datacenter networks beyond 10GbE.

C. Accuracy

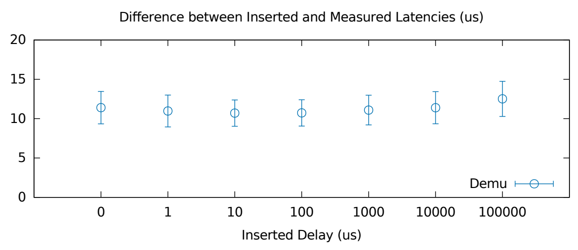
In order to compare the emulation accuracy of DEMU and NetEm under the same condition, we used 1 Gbps traffic composed of 1518-byte frames for experiments, since both emulators can handle this network traffic without packet loss. The duration of each experiment was 10 seconds; for this traffic, the number of generated packets in each experiment was set to 8000000. Figures 5a and 5b show the difference between an inserted delay and its actual measured latency, achieved by NetEm and DEMU, respectively. For NetEm, the average observed difference was about 40 µs. This value is approximately the minimum processing latency of NetEm required just for forwarding packets without emulating a delay. It is not possible to emulate a shorter network latency than 40 µs. In addition, we observed that the standard deviations of measured latencies were approximately 20 µs. Clearly, NetEm will not be suitable for precisely emulating sub-milliseconds latencies. On the other hand, the minimum processing latency of DEMU was only approximately 8 µs. The standard deviations of measured latencies were about 0.3 µs, which is 2 orders of magnitude smaller than that of NetEm. These results show that DEMU can emulate latencies on the order of 10 µs with a 1 µs resolution for 1 Gbps traffic.

Figures 6a and 6b show the measured latency of each packet over a period of 2 ms, in the experiments that NetEm and DEMU inserted a 100-us delay to 1 Gbps traffic of 1518-byte frames, respectively. These figures explain observed latency deviations. In the graph of DEMU, there were 1.5-us spikes in measured latencies at every 32 frames. This probably originates from the default burst size of the used DPDK NIC driver (IXGBE PMD). We are further investigating microscopic behavior of NIC queue operations. There will be a possibility that we can improve emulation accuracy by further optimizing the parameters.

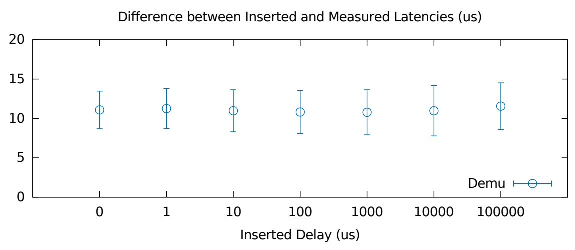
We also conducted experiments using network traffic at the line rate of 10GbE. Since NetEm cannot handle this rate of traffic, we discuss only the results of experiments using DEMU. Figures 7a and 7b show the results of the experiments using 1518-byte frames and 64-byte frames, respectively. In comparison to the results of 1 Gbps traffic, the minimum processing latency increased to 11-12 µs. The standard deviation of emulated latencies increased to 2-3 µs, which was still only 10% of that of NetEm handling 1 Gbps traffic.

D. Summary

We confirmed that DEMU is capable of inserting latency to a 10GbE link. Even though network traffic includes many short packets, it is possible to achieve the line rate of 10GbE. It can emulate a network latency that is larger than 12 µs with the standard deviation up to 2-3 µs. We consider that this resolution will allow us to emulate typical latencies of software-based network functions (e.g., several tens of microseconds). For example, we correctly estimated the performance impact of a virtual switch on a database server (memcached); if we replace a hardware switch with the virtual switch that incurs 30 µs of network latency (i.e., RTT 60 µs), our experiments using DEMU demonstrated that the performance of the database decreases to 81%. Conversely, the experiments using NetEm erroneously reported that the performance decreases to 54%,



(a) 1518-byte frames



(b) 64-byte frames

Fig. 7: The difference between an inserted delay and its measured latency, achieved by **DEMU** for line-rate 10GbE traffic.

which resulted in a serious overestimate of the performance impact.

V. Related Work

To our best knowledge, this paper is the first work thoroughly investigating the feasibility of network latency emulation based on an emerging packet processing framework. This work succeeded in discussing microseconds-order accuracy by means of our high-resolution network test system.

Netmap [14] is another programming framework providing an API to develop high-performance packet processing programs. In [15], the authors of netmap presented a network emulation program (TLEM) based netmap. Their motivation also supports our work; network I/O stacks of commodity operating systems, designed tens of years ago, cannot efficiently handle emerging high-speed network technologies, and network emulators implemented in operating system kernels are not flexible to add new features. TLEM is composed of multiple pipelined stages running on multiple CPU cores, which is similar to the design of DEMU achieving the line rate of 10GbE and beyond. Although estimating that TLEM will be accurate to 50 µs and be capable of forwarding 40 Gbps traffic of 1518-byte frames, their paper, supposedly published as preliminary work, does not include any detailed evaluations using real network links. According to [16], in terms of latency and throughput, DPDK achieved better performance than netmap. It will be fruitful to compare DEMU and TLEM in order to pursue high-performance, noise-less network emulation techniques at the software layer.

VI. Conclusion

In this paper, we proposed a DPDK-based network latency emulator, DEMU. It avoids the overhead of general-purposed network stack by directly interacting with the hardware interface of network interface cards. Through experiments, we confirmed that DEMU is capable of forwarding short-packet traffic at the line rate of 10GbE and the deviation of inserted latencies was only 2-3 µs. We consider that this performance will be sufficient to investigate the impact of network virtualization on application performance in datacenters.

The authors would like to thank Dr. Jason Haga and other colleagues for their invaluable feedback. This work is partially supported by the Vertically Integrated Center for Technologies of Optical Routing toward Ideal Energy Savings (VICTORIES) pursuing the Project for Developing Innovation Systems of the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan.

References

[1] Y. Nakajima, T. Hibi, H. Takahashi, H. Masutani, K. Shimano, and M. Fukui, "Scalable, high-performance, elastic software OpenFlow switch in userspace for wide-area network," Open Networking Summit 2014, pp. 1–2, Mar 2014.

[2] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI2015)*. USENIX Association, 2015, pp. 117–130.

[3] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An open framework for Openflow switch evaluation," in *Proceedings of the 13th International Conference on Passive and Active Measurement (PAM2012)*. Springer-Verlag, 2012, pp. 85–95.

[4] R. Bolla and R. Bruschi, "Linux software router: data plane optimization and performance evaluation," *Journal of Networks*, vol. 2, no. 3, pp. 6–17, 2007.

[5] Y. Kodama, T. Kudoh, R. Takano, H. Sato, O. Tatebe, and S. Sekiguchi, "GNET-1: Gigabit Ethernet network testbed," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, Sep 2004, pp. 185–192.

[6] S. Hemminger, "Network emulation with NetEm," in *Australia's National Linux Conference 2005 (LCA2005)*, Apr 2005, pp. 1–9.

[7] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," *SIGCOMM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, Jan. 1997.

[8] M. Carson and D. Santay, "NIST Net: A Linux-based network emulation tool," *SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 111–126, Jul. 2003.

[9] DPDK: Data Plane Development Kit, http://dpdk.org/, 10 2016.

[10] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," RFC 2544, Internet Engineering Task Force, Mar. 1999.

[11] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Assessing soft- and hardware bottlenecks in pc-based packet forwarding systems," *IARIA ICN 2015*, 2015.

[12] D. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305, Internet Engineering Task Force, Mar. 1992.

[13] X. Wu, P. Li, Y. Ran, and Y. Luo, "Network measurement for 100Gbps links using multicore processors," in *The 3rd Innovating the Network for Data-Intensive Science (INDIS2016)*, Nov 2016, pp. 1–10.

[14] L. Rizzo, "Netmap: A novel framework for fast packet I/O," in *Proceedings of 2012 USENIX Annual Technical Conference (USENIX ATC'12)*, Jun 2012, pp. 101–112.

[15] L. Rizzo, G. Lettieri, and V. Maffione, "Very high speed link emulation with TLEM," in *Proceedings of the 22nd IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN2016)*, June 2016, pp. 1–6.

[16] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *Proceedings of the Eleventh ACM/IEEE Symposium (ANCS2015)*, 2015, pp. 29–38.