How to Measure the Killer Microsecond

Mia Primorac Edouard Bugnion Katerina Argyraki
EPFL
first.last@epfl.ch

ABSTRACT

Datacenter-networking research requires tools to both generate traffic and accurately measure latency and throughput. While hardware-based tools have long existed commercially, they are primarily used to validate ASICs and lack flexibility, *e.g.,* to study new protocols. They are also too expensive for academics. The recent development of kernel-bypass networking and advanced NIC features such as hardware timestamping have created new opportunities for accurate latency measurements. This paper compares these two approaches, and in particular whether commodity servers and NICs, when properly configured, can measure the latency distributions as precisely as specialized hardware.

Our work shows that well-designed commodity solutions can capture subtle differences in the tail latency of stateless UDP traffic. We use hardware devices as the ground truth, both to measure latency and to forward traffic. We compare the ground truth with observations that combine five latency-measuring clients and five different port forwarding solutions and configurations. State-of-the-art software such as MoonGen that uses NIC hardware timestamping provides sufficient visibility into tail latencies to study the effect of subtle operating system configuration changes. We also observe that the kernel-bypass-based TRex software, that only relies on the CPU to timestamp traffic, can also provide solid results when NIC timestamps are not available for a particular protocol or device.

CCS Concepts

•General and reference → Measurement; •Networks → Network measurement; Middle boxes / network appliances; **•Hardware →** *Networking hardware;*

Keywords

microsecond latency

1. INTRODUCTION

Network researchers need tools to generate traffic and measure latency and throughput. The ideal tool would combine low cost, flexibility, and accuracy: it would be inexpensive to obtain and usable with commodity components; enable the generation of arbitrary traffic patterns and the testing of arbitrary protocols; and provide latency – including tail-latency – measurements at the µs-scale. An eager client for such a tool today would be the community researching network function virtualization (NFV), whose goal is to study the latency and throughput of network functions [18, 14].

The industry has traditionally used hardware-based tools [21, 28], which provide accuracy, but neither flexibility nor low cost: they are excellent for validating Application Specific Integrated Circuits (ASICs) using standardized approaches [7], but they cannot test arbitrary protocols, and they are too expensive for most researchers. For the price of a hardware traffic generator that is able to saturate a link with tens of Gbps, one can buy tens of commodity servers with multiple NICs.

Researchers, on the other hand, typically use software tools, which provide low cost and flexibility, but their accuracy is unclear, if not downright questionable [6]. We believe we are not the only ones who have experienced the frustration of using software traffic generation and measurement – because that is the only option – while worrying about noise and repeatability, especially when the Linux networking stack and socket-based interface are involved [6]. With datacenter and cloud operators chasing the killer microsecond [5], researchers increasingly report results in µs-scale tail latencies [23, 30, 27]; but such results can be trusted only if they are obtained with a tool that provides accuracy at the same scale.

Kernel bypass, as the means to faster I/O [1, 25], and hardware timestamping, now increasingly available in commodity Network Interface Cards (NICs), are creating new opportunities for building better traffic generators and measurement tools. For instance, MoonGen – a scriptable, high-speed packet generator built on top of Intel DPDK (Data Plane Development Kit) can provide precise latency measurements while executing user-provided Lua scripts per packet [13]. It relies on many modern NICs having the hardware-based packet timestamping tailored to the precise requirements of IEEE 1588 time synchronization. In addition, some NICs such as the Intel 82580 [19] provide hardware support to timestamp all received packets. Unfortunately, outgoing packets must still be timestamped in software by the application. Recent work shows that precise RTT measurements with hardware timestamps can be highly beneficial even for datacenter congestion control [24, 8, 22]. However, the precision of the NIC hardware timestamps has its limits [13, 22] and, up to our knowledge, has not yet been evaluated against a commercial hardware appliance.

We ask the following two questions:
(1) *How close do state-of-the-art commodity solutions get to bridging the gap between hardware and software and providing accurate µs-scale tail latency measurements?*
(2) *Are the measurements sufficiently accurate to study the latency distribution of software network functions?*

We answer the first question based on a simple observation: the latency of a constant-rate flow going through an ASIC-based switch is expected to be constant. We first use a proprietary hardware-based measuring device to confirm

Previously published in SIGCOMM KBNets'17

Tool	Characteristics	Latency Measurements	Measured at Granularity
Spirent [28]	commercial hardware appliance, commonly used for standardized RFC2544 performance tests	FPGA-based or proprietary	$10ns$, 1µs, 5µs
MoonGen [13]	Dataplane using DPDK and Lua	Determined by the NIC leveraging IEEE 1588 support (when available)	$10ns$ (hardware), $100ns$ (software)
TRex [10]	DPDK dataplane	Determined by the CPU	$100ns$
netperf [2]	socket-based interface	Determined by the CPU	$100ns$

Table 1: Overview of evaluated traffic generators.

that it indeed hardly varies. We then use this measured latency as the ground truth and determine up to which percentile different software tools measure it correctly.

We answer the second question by sending constant-rate flows through a software network function that simply forward packets. We use different hardware and software tools to observe the impact of operating system (OS) configurations on the network function's tail latency up to the 99.9999^{th} percentile. We quantify the mismatch between the hardware ground truth and other tools.

We also contribute the following results:

- The use of NIC-based hardware timestamps on commodity NICs, such as Intel X710 10GbE, provides accurate readings up to the 99.99^{th} percentile, but not beyond.
- A tuned DPDK solution such as TRex introduces 5µs to 10µs overhead in readings, yet does allow to study the impact of operating-system configuration changes in network forwarding devices.
- POSIX-based solutions that rely on blocking I/O introduce almost 20µs overhead at 50^{th} percentile and have a 50µs long tail, hence should be avoided when measuring µs-scale latencies.
- Our study suggests that bidirectional hardware support is highly beneficial to accurately measure µs-scale latencies.

2. TOOL OVERVIEW

Table 1 lists the traffic generation and measurement tools that we consider in this paper. Our goal is not a comparison of all the available tools – we consider only a subset that we deemed sufficient for understanding where kernel bypass lands between traditional hardware and software tools when it comes to latency measurements.

Spirent [28] represents state-of-the-art hardware-based tools. It was designed to accurately measure ns-scale latency, but it is customized for a fixed set of pre-defined, standardized tests such as the ones specified in RFC 2544 [7]. It is possible to configure traffic generation to some extent, through GUI or scripts written in high-level languages, some of which require an extra license that bears a substantial cost.

MoonGen [13] represents state-of-the-art software tools that leverage kernel bypass and hardware timestamping at the NIC. It is built on top of DPDK and LuaJIT, and it is fully scriptable. Its best reported performance result is 178.5 Mpps with 64-byte packets running on twelve CPU cores at 2 GHz and twelve 10Gbps links while executing user-provided Lua scripts per packet.

TRex [10] is a software tool that leverages kernel bypass as well, but it relies on software timestamps. We use the stateless version, whose best reported performance result is that it can generate 10-20 Mpps with 64-byte packets while running on one core.

Finally, netperf [2] represents traditional software tools that use blocking POSIX API and conventional network drivers. Even though it was designed to measure performance and not as a full-fledged traffic generator, netperf can generate constant-rate UDP traffic of configurable message size, burst size, and inter-message time. This flexibility is good enough for assessing the benefit of kernel bypass over traditional I/O for latency measurements.

3. EXPERIMENTAL SETUP

We now describe our experimental setup, including the configuration of any hardware and software tools.

3.1 Hardware setup

We use four devices: a Cisco SG500X-48 switch [9] ("HW switch"), an FPGA-based Spirent SPT-3U chassis [28] ("Spirent"), and two x86 machines, one acting as a software traffic generator and measurement node ("SW generator"), the other as a network function ("NF"). The nodes are connected with two meters long 10GbE cables. The x86 machines are dual socket Intel Xeon CPU E5-2699 v4 @ 2.20GHz with hyperthreading disabled, each with two Intel x710 10GBE NICs [20].

We experiment with the four configurations, run one at a time, depicted in Figure 1 and enumerated accordingly:

1. Spirent + HW switch: to measure the true latency of the Cisco switch.
2. SW generator + HW switch: to measure the accuracy of latency measurements achievable with software tools.
3. Spirent + NF: to measure the true latency of our network function.
4. SW generator + NF: to determine whether software tools can accurately measure the latency of our network function.

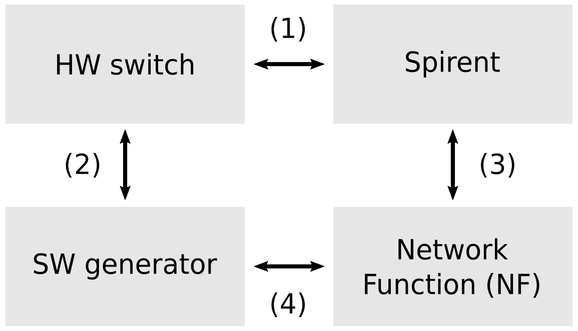


Figure 1: Experimental setup.

In all experiments, we use two distinct physical ports from each device, and each port is both sending and receiving traffic (so, we have two independent end-to-end flows). In the case of the x86 machines, the two ports are located in different NICs, but attached to the same CPU socket's PCIe root complex.

3.2 Software setup

The "NF" machine implements port forwarding. The OS is Fedora Linux 23 with kernel version 4.4.9. The forwarding software uses DPDK version 17.02 and consists of two forwarding streams in opposite directions, running on two cores that each have a dedicated RX and TX queue. Unless otherwise stated, we configure the machine to minimize jitter: we disable all the power-saving options such C-states and P-states, NUMA balancing, transparent huge pages, kernel audit, and interrupt moderation, and we run all the cores at the nominal frequency (but not TurboBoost).

The "SW generator" machine has similar configuration, but runs one of the following programs:

MoonGen with hardware timestamping: We use the latest version from GitHub [12]. Hardware timestamping was designed for IEEE 1588 [11] time synchronization, and it can only timestamp one outstanding single packet at a time due to resynchronization requirements, therefore can do only sampling of the latency distribution. It uses a separate hardware queue for the non-timestamped traffic.

MoonGen with software timestamping: MoonGen also works with software timestamps. In this case, there is no sampling limitation – the latency of all packets can be captured. We keep a 100ns-granularity histogram.

Netperf: We use the standard netperf tool, but replaced its histogram implementation with our own, more fine-grained one (100ns). We used UDP request-response benchmark with histograms and inter packet time control enabled.

TRex: We use the latest stateless version. We created a control plane experiment to fit our benchmark requirements. As with netperf, we replaced its original coarse-grain histogram implementation (10µs granularity) with our own (100ns granularity).

We further isolate the CPUs on which we run the traffic generators, and pin the forwarding tasks to these CPUs. We also make sure the cores, ports, and allocated memory are on the same socket.

In all experiments, the (hardware or software) traffic generator produces two independent UDP flows of 64-byte packets, each one at a rate 1Gbps. We calibrate all the tools to the same line rate using Spirent as a sink. We report the data from 5 independent runs of each experiment. Each run executes the benchmark for 120 seconds after a warm-up of 30 seconds.

Measurement granularity depends on the tool. The Spirent chassis has 16 adjustable-size histogram buckets, which we set after calibration to 10ns in §4.1 and between 1µs and 5µs in §4.2. Hardware timestamps in MoonGen have the precision of 10ns. The software solutions (MoonGen-SW, TRex, netperf) keep a 100ns-granularity histogram of latencies as measured using the processor's cycle counter.

4. RESULTS

We now answer our two basic questions: when measuring µs-scale latency, how far are state-of-the-art software tools from traditional hardware-based tools (§4.1)? and are software tools accurate enough for measuring the latency of software network functions (§4.2)?

4.1 Closing the HW/SW gap

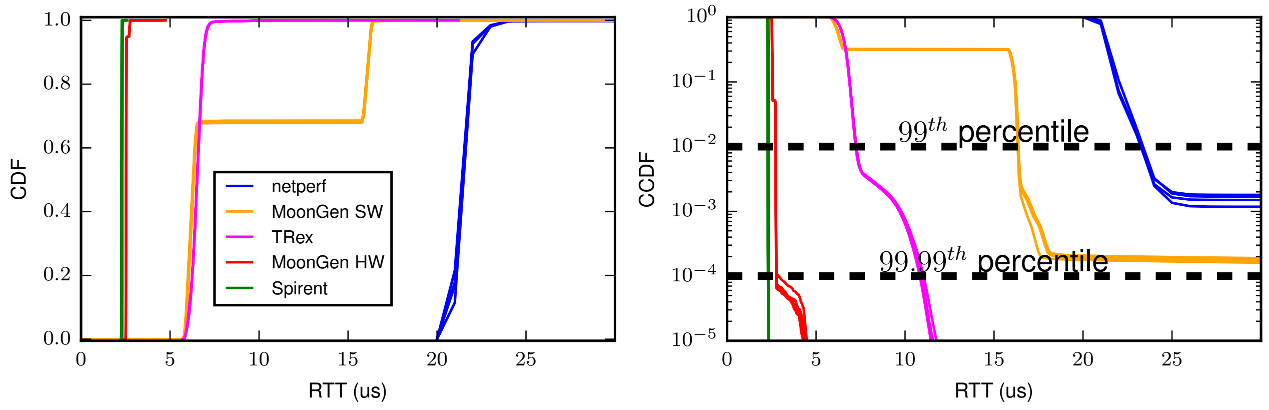
To answer the first question, we use configurations (1) and (2) to measure the latency of the HW switch. The idea is that any modern ASIC-based switch is expected to offer per-packet latency of a couple µs with insignificant jitter; we use configuration (1) to confirm this, and configuration (2) to test whether the software tools can measure µs-scale latencies.

Figure 2 shows the switch's latency distribution as reported by the different tools. The same data is presented in two different ways, akin to [23]: Figure 2(a) shows the cumulative distribution function (CDF) of the latency, while Figure 2(b) shows the complementary cumulative distribution function (CCDF), which provides a more explicit view of tail latency (shows which fraction of measurements exceed a given latency value). The CCDF is presented on a log scale to highlight the effects of tail latency. In case the figure is viewed in black and white, the labels are ordered by ascending accuracy, i.e., the top-most label (netperf) corresponds to the right-most (least accurate) latency distribution. We report data up to the 99.9999th percentile $(1 - 10^{-6})$.

First, we confirm that the HW switch provides stable, if not exceptionally low, latency, as expected from an ASIC-only datapath: Spirent reports minimum, mean, and maximum latency of 2.24µs, 2.26µs, and 2.52µs, respectively. The spread across more than 400 million measurements is, therefore, less than 300ns. The reported CCDF (left-most one in Figure 2(b)) is near vertical up to the 99.9999th percentile.

Second, we see that the combination of kernel bypass and hardware timestmaps comes very close to the ground truth provided by Spirent: MoonGen-HW reports minimum, median, and 99.99th percentile latency of 2.466 µs, 2.524 µs, and 2.723 µs, respectively. The reported CCDF (second from the left in Figure 2(b)) is less than a µs away from the ground truth up to the 99.99th percentile. Beyond that, however, the error increases. For instance, the 99.999th percentile $(1 - 10^{-5})$ latency is 4.379 µs, a noticeable increase, most likely due to the imperfect synchronization between the two different NICs of the SW generator (the one where each measured packet departs and the one where it arrives) [13].

Third, we see that kernel-bypass alone is not enough, hardware timestamps are necessary to get this close to the ground truth: MoonGen-SW reports latency between 5.225 µs and 7.1 µs for 60% of the measurements, but signifi-



(a) CDF of the latency

(b) Complementary CDF

Figure 2: Latency measurements of a hardware switch using different tools.

cantly higher for the rest. TRex does better, overlapping with MoonGen-SW for 60% of the measurements, including the median of 6.5 µs, but reporting stable latency up to the 99th percentile of 7.3 µs, only a 28% increase from the minimum value. We attribute the discrepancy between the two tools to MoonGen's use of Lua JIT within the datapath: unless hardware timestamps are available, having your software generator JIT-compiled comes at a high price.

Finally, we see the limits of using the standard POSIX API and conventional network drivers for latency measurements: Netperf (the right-most curve in both graphs) is at least 17µs off the ground truth. We attribute the gap to highly variable latency of interrupt dispatching and thread wakeups on multicore machines. Our conclusions are reproducible even with NIC ports directly connected. For more details please refer to our repository [4].

4.2 Measuring network functions

To answer the second question, we build on the insights of §4.1 to measure the latency distribution of our network function (DPDK port forwarding). We want to experiment with scenarios that introduce non-trivial latency and jitter, but are also realistic and interesting to the networking community. So, instead of introducing artificial latency and jitter ourselves, we consider four OS-level configurations that have latency implications:

1. **local**: a baseline "out-of-the-box" OS configuration, where the network function (CPU and memory) runs on the same NUMA socket that has the PCIe root complex of the NIC. The NIC/memory interactions are therefore all local to the same socket.

2. **remote**: also a baseline OS configuration, but the network function runs on the remote NUMA node relative to the PCIe root complex of the NIC. All NIC/memory interactions must therefore go through the QPI interface between sockets.

3. **local+isolset**: we augment "local" to further use the `isolcpu` and `taskset` features of the Linux scheduler to explicitly isolate the network function and ensure that no other application is ever scheduled on the same core.

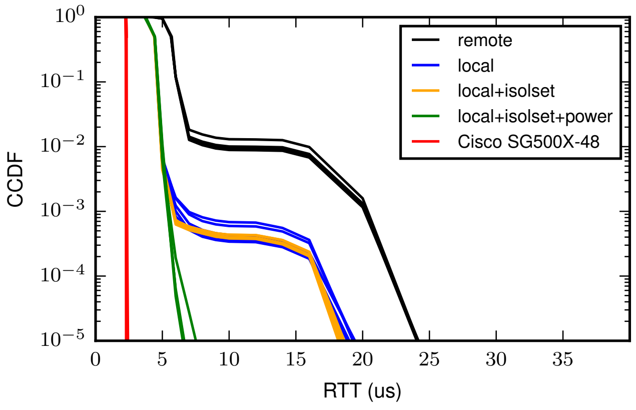
4. **local+isolset+power**: we further disable power-saving options including P-states (and TurboBoost), C-states and PCIe Active State Power Management.

5. **Cisco SG500X-48**: as a reference, we again show the hardware switch that forwards the same traffic between two physical ports.

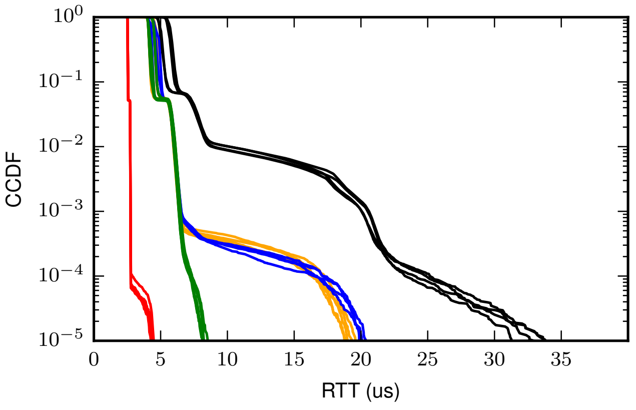
Figure 3 shows the latency distribution of the network function, for each of the four OS configurations, as reported by different tools. Each subfigure shows the latency CCDF of the network function for the four OS configurations, as well as the latency CCDF of the HW switch, which is used as a reference. Each subfigure reports data captured by a different tool; we omit netperf from this evaluation due to its limitations shown in Figure 2. The labels are ordered by ascending accuracy, i.e., the top-most label (remote) corresponds to the right-most (least accurate) distribution.

First, we establish the ground truth: Figure 3a shows the NF's latency CCDF as reported by Spirent (testbed configuration (3) in Figure 1), which is the most precise of the considered tools (Figure 2). We see that, while the NF is clearly slower than the HW switch, they can both deliver relatively low jitter. We also clearly see the impact of OS configuration on latency: when considering minimum, or even median latency, it is necessary and sufficient to ensure that the local socket is consistently used; when considering tail latency, however, it is essential to further control power settings. For instance, at the 99.99th percentile, the appropriate power settings reduce tail latency by a factor of 2.6, which is consistent with prior observations [23]. The "local+isolset+power" configuration has the lowest latency and jitter, with a minimum latency of 3.73 µs and a maximum latency of 10.72 µs, and a smooth CCDF near-vertical line between the two.

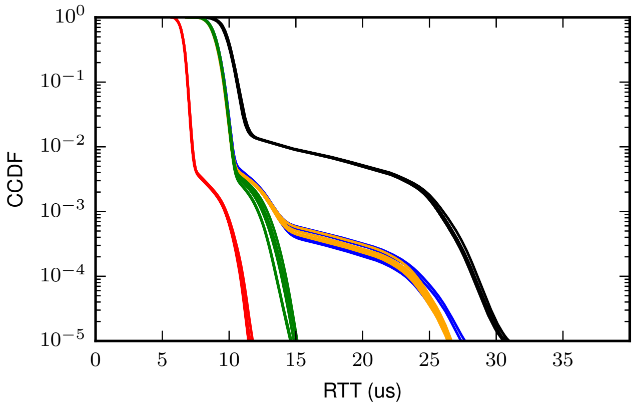
Next, we assess how well the software tools can measure the same NF latency: Figures 3b-3c-3d (testbed configuration (4) in Figure 1) show how MoonGen-HW, TRex, and MoonGen-SW, respectively, report gradually noisier latency distributions. Still, both MoonGen-HW and TRex are accurate enough to capture the impact of OS configuration on NF tail latency; TRex may be off by several µs in absolute terms, but it does captures correctly the relative overhead introduced by each OS configuration. MoonGen-SW (as well



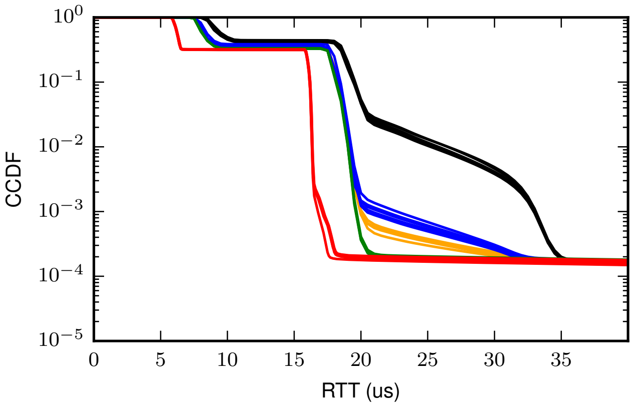
(a) client: Spirent



(b) client: MoonGen-HW



(c) client: TRex



(d) client: MoonGen-SW

Figure 3: Effect of different OS and application configurations of the software port-forwarding application, as measured by different client tools. The hardware switch is added for comparison.

as netperf, not shown), on the other hand, does not.

5. RELATED WORK

Many tools and systems can measure latency beyond MoonGen, TRex and Spirent: Pktgen [29] is built on top of DPDK, similar to Moongen and TRex. Caliper [16] and OFLOPS [26] are built on top of the NetFPGA platform [17]; we used Spirent instead.

Zilberman et al. [31] recently focused on dissecting one-way latency from the wire to the application up to the *ns*-scale latency. Their methodology relies on Data Aggregation and Generation (DAG) cards that have precision comparable to Spirent's. They used a latency-optimized NIC that relies on a proprietary driver and a proprietary kernel bypass framework [15], and a Solarflare NIC [3].

While they focused on the best possible case in terms of hardware and software setup on the measured side, we evaluated commonly-used techniques in academia and industry on the measuring side. Both sets of results show that kernel-bypass improves tail latency by $20 - 40\mu s$.

6. CONCLUSION

We evaluate different software- and hardware-based latency measuring tools. While dedicated hardware devices provide the most precision, NIC-based timestamping developed for IEEE 1588 can also provide precise measurements and are much more easily integrated into flexible packet generators such as MoonGen.

We then study the impact of operating system settings on a simple DPDK-based port forwarder. One can observe the impact of these configuration on tail latencies using either dedicated hardware, NIC-based timestamps, or kernel-bypass based software with CPU timestamps.

Our results clearly show the benefit of measuring latency of packet requests and responses, and more generally of remote procedure calls, within the NIC as opposed to CPU. We focus on µs- instead of *ns*-scale measurements for comparing software- and hardware-based tools, because the former is a "gray area" where optimized software techniques might still stand a chance.

While many modern NICs support hardware-based timestamping, the implementation is narrowly tailored to the precise requirements of IEEE 1588 time synchronization, *i.e.,* matching one outstanding PTP timestamping packet with its reply. A more flexible implementation, integrating bidirectional timestamping into arbitrary protocols and packet formats, would be highly beneficial.

Acknowledgments

The authors thank Intel Corp, who gave us access to the hardware we used, and in particular Mesut A. Ergin, Ren Wang and Charlie Tai. This research is suppored in part by an Intel grant, a VMware grant, and the Microsoft-EPFL Joint Research Center.

7. REFERENCES

[1] Data Plane Development Kit. http://dpdk.org/. Last accessed: 2017-03-01.

[2] Netperf. http://www.netperf.org/netperf/. Last accessed: 2017-03-08.

[3] Solarflare Flareon Ultra SFN8522-PLUS. https://www.solarflare.com/Media/Default/PDFs/SF-116323-CD-LATEST_Solarflare_SFN8522-PLUS_Product_Brief.pdf.

[4] Repository. https://github.com/MihaelaMia/measure-killer-us, 2017.

[5] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan. Attack of the killer microseconds. *Commun. ACM*, 60(4):48–54, Mar. 2017.

[6] A. Botta, A. Dainotti, and A. Pescapè. Do you trust your software-based traffic generator? *IEEE Communications Magazine*, 48(9):158–165, 2010.

[7] S. Bradner and J. McQuaid. Benchmarking Methodology for Network Interconnect Devices. IETF RFC 2544, Mar. 1999.

[8] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: congestion-based congestion control. *Commun. ACM*, 60(2):58–66, 2017.

[9] Cisco Systems. Cisco SG500X-48 48-Port GB with 4-Port 10-GB Stackable Managed Switch.

[10] Cisco Systems. TRex: Cisco's realistic traffic generator. https://trex-tgn.cisco.com. Last accessed: 2017-03-01.

[11] J. Eidson and K. Lee. IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In *Sensors for Industry Conference, 2002. 2nd ISA/IEEE.*

[12] P. Emmerich. Moongen's GitHub repository (commit ef3aa3f). https://github.com/emmericp/MoonGen. Last accessed: 2017-03-01.

[13] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *IMC*, pages 275–287, 2015.

[14] European Telecommunications Standards Institute. Network Functions Virtualisation – Introductory White Paper. http://portal.etsi.org/NFV/NFV_White_Paper.pdf, 2012.

[15] Exablaze. ExaNIC X10. https://exablaze.com/exanic-x10. Last accessed: 2017-08-30.

[16] M. Ghobadi, M. Labrecque, G. Salmon, K. Aasaraai, S. H. Yeganeh, Y. Ganjali, and J. G. Steffan. Caliper: a tool to generate precise and closed-loop traffic. In *SIGCOMM*, pages 445–446, 2010.

[17] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown. NetFPGA - An Open Platform for Teaching How to Build Gigabit-Rate Network Switches and Routers. *IEEE Trans. Education*, 51(3):364–369, 2008.

[18] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[19] Intel Corporation. Intel 82580EB/82580DB Gigabit Ethernet Controller Datasheet. http://www.intel.com/content/www/us/en/embedded/products/networking/82580-eb-db-gbe-controller-datasheet.html. Revision: 2.7, September 2015.

[20] Intel Corporation. Intel Ethernet Controller 710 Series Datasheet. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xl710-10-40-controller-datasheet.pdf. Revision: 2.9, April 2017.

[21] Ixia. Ixia traffic generator. https://www.ixiacom.com. Last accessed: 2017-03-01.

[22] C. Lee, C. Park, K. Jang, S. B. Moon, and D. Han. Accurate Latency-based Congestion Feedback for Datacenters. In *USENIX ATC*, pages 403–415, 2015.

[23] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble. Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency. In *SOCC*, pages 9:1–9:14, 2014.

[24] R. Mittal, V. T. Lam, N. Dukkipati, E. R. Blem, H. M. G. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. TIMELY: RTT-based Congestion Control for the Datacenter. In *SIGCOMM*, pages 537–550, 2015.

[25] L. Rizzo. netmap: A Novel Framework for Fast Packet I/O. In *USENIX ATC*, pages 101–112, 2012.

[26] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. OFLOPS: An Open Framework for OpenFlow Switch Evaluation. In *PAM*, pages 85–95, 2012.

[27] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout. It's Time for Low Latency. In *HOTOS-XIII*, 2011.

[28] Spirent Communications. Spirent test modules and chassis. https://www.spirent.com/Products/TestCenter/Platforms/Modules. Last accessed: 2017-03-01.

[29] D. Turull, P. Sjödin, and R. Olsson. Pktgen: Measuring performance on high speed networks. *Computer Communications*, 82:39–48, 2016.

[30] Y. Zhang, D. Meisner, J. Mars, and L. Tang. Treadmill: Attributing the Source of Tail Latency through Precise Load Testing and Statistical Inference. In *ISCA*, pages 456–468, 2016.

[31] N. Zilberman, M. P. Grosvenor, D. A. Popescu, N. M. Bojan, G. Antichi, M. WÃşjcik, and A. W. Moore. Where Has My Time Gone? In *PAM*, pages 201–214, 2017.