A Black-Box Approach for Estimating Utilization of Polled IO Network Functions

Harshit Gupta* Abhigyan Sharma† Alex Zelezniak† Minsung Jang† Umakishore Ramachandran*
*Georgia Institute of Technology †AT&T Labs Research

Abstract

Cloud management tasks such as performance diagnosis, workload placement, and power management depend critically on estimating the utilization of an application. But, it is challenging to measure actual utilization for polled IO network functions (NFs) without code instrumentation. We ask if CPU events (e.g., data cache misses) measured using hardware performance counters are good at estimating utilization for polled-IO NFs. We find a strong correlation between several CPU events and NF utilization for three representative types of network functions. Inspired by this finding, we explore the possibility of computing a universal estimation function that maps selected CPU events to NF utilization estimates for a wide-range of NFs, traffic profiles and traffic loads. Our NF-specific estimators and universal estimators achieve absolute estimation errors below 6% and 10% respectively.

1 Introduction

Network functions (NFs) such as routers, firewalls, and proxies use polled network IO in userspace for high performance [10, 13, 21]. Like any application, they require common cloud management tasks such as performance debugging [17], workload placement [16] and power management [20]. These tasks often depend on *NF utilization*: the percentage of the peak traffic supported by the NF that the traffic being processed by an NF represents, other factors such as hardware and traffic patterns remaining unchanged. But, due to polling, these NFs always report 100% CPU core utilization, even when they are not processing any packets! There are no server-level techniques to estimate the utilization of black-box NFs. The only approach is to instrument NF code to report cycles spent in packet processing and those spent in empty polls [27]. However, a telco provider such as AT&T deploys dozens of such NFs from vendors across the industry [12], which makes it difficult to instrument all NFs to collect a common set of statistics from all of them.

We propose an approach that requires no modifications to NFs or their network drivers. Our idea is to use CPU events, e.g., branch misses, or L3 cache misses, collected using hardware performance counters [1]. CPU events have ubiquitous support in server CPUs and software utilities [7, 28]. Given a limited number of hardware counters in a CPU, the challenge is to choose which CPU events to monitor and how to map the events' counter values to NF utilization values. At the outset, there are two extreme possibilities for this line of research. It is possible that one finds a single CPU event (out of hundreds of events in a modern CPU [8]) whose counter values can be mapped to NF utilization for all NFs for all traffic profiles. The other extreme would be if no combinations of CPU events are found to accurately estimate the utilization of any NF for common traffic scenarios. The reality, as we show in this paper, lies somewhere in the middle.

Our primary contribution is an empirical study of NF utilization estimation using CPU events with three NFs from different layers of the stack: a layer-3 forwarder [11], a layer-4 load balancer [24], and a layer-7 deep-packet inspection function [4]. We subject these NFs to a range of workloads with varying packet sizes, connection lengths and traffic volumes for collecting data on counter values for all CPU events. We use the events' data to train a variety of *estimation functions* or *estimators* that map counter values for a small number of CPU events to an NF's utilization. Our evaluation of these estimators reveals following insights:

- Simple *local* (NF-specific) estimators achieve an absolute error below 6% using just 3 CPU events.
- A simple *universal* estimator for all NFs based on 3 events can achieve an absolute error of less than 10%.
- The accuracy achieved by local estimators on an unseen test profile is close to that on profiles seen previously.
- An estimator can achieve estimation errors close to the best while measuring hardware counters for 10 ms only.

We conclude with a discussion on designing improved universal estimators based on hierarchical estimation functions, include additional kinds of NFs virtualized NFs and evaluation of use cases such as workload management and performance debugging.

2 Goal and approach

2.1 A "top" for polled IO NFs

We provide background on DPDK – a framework for building polled I/O NFs, and discuss prior work on instrumentation-based techniques to assist management of NFs.

Anatomy of a DPDK application: DPDK [10] is a toolkit for developing high-performance NFs. DPDK applications use polled IO from userspace, a technique also used by other high-performance NFs [2, 21]. This approach eliminates the overheads of context switching, packet copying and interrupt handling to maximize performance. But, polling results in 100% CPU use for the cores that the NF is running on.

Instrumentation-based techniques: Cloud management of NFs (including both polled NFs and others) takes help from instrumentation of NFs, their network interface drivers and cloud infrastructures. Trifonov *et al.* [27] measure the number of empty polls to determine utilization and insert artificial delays between consecutive polls to reduce polling frequency. ENVI [5] uses a combination of NF instrumentation and cloud infrastructure instrumentation to inform elastic scaling decisions for NFs. But, cloud instrumentation needs OS support to measure CPU utilization, which does not exist today for polled IO NFs. Niccolini *et al.* [20] use NIC driver instrumentation to estimate queue lengths of NIC packet buffers. Queue lengths are used as a proxy for NF utilization in performing CPU power management. In general, these approaches are difficult to apply to unmodified polled IO NFs.

Goal: Our position is that getting a good utilization estimate for unmodified NFs can vastly simplify cloud management tasks such as performance diagnosis, workload management and power management for them. Towards this goal, we seek to create a utility like UNIX's `top` for polled IO NFs. Similar to `top`, this utility would list the polled IO NFs and their utilization on each of the cores they are running on.

2.2 NF utilization via CPU events

CPU events: Our intuition is that a polled IO NF that is processing packets is likely to generate a different type of activity on the CPU, e.g., data cache access due to arriving packets or branch prediction errors in the polling thread due to non-empty polls. A modern CPU defines hundreds of these events (e.g., 714 CPU events on an Intel Xeon E5-2680 v3 in our testbed), which can give a detailed picture of CPU activity. This motivates us to explore if there are some CPU events whose frequency is strongly correlated to NF's utilization.

Hardware performance counters: A CPU core has a small number of registers called *hardware performance counters*, each of which can be programmed to count a CPU event as shown in Figure 1. Such counters have long been available in server CPUs such as Intel and AMD [6, 25]. Support for using them is available in various utilities such as Linux perf [28] and Intel EMON [7]. Hardware counters have been widely used such as for application profiling (e.g., L1/L2/L3 cache misses) [1, 25] and modeling power use [18].

```
msr_fd = open("/dev/cpu/2/msr", O_RDWR | O_SYNC);
    // descriptor for registers on CPU core 2
c = 0x005300c4; // CPU event for branch hits
ret = pwrite(msr_fd, c, sizeof(c), 0x186);
    // configuration register 0x186 will record branch hits
...
ret = pread(msr_fd, &val, sizeof(val), 0xc1);
    // read counters from counter register 0xc1
```

Figure 1: Read/write to performance counter registers.

Advantages of CPU events: CPU events are deployment friendly for several reasons. They require no modification to NF, network drivers or cloud infrastructures. In comparison, existing instrumentation-based techniques would require considerable effort to instrument run dozens of NFs [12] and NIC types in a large ISP network. Further, hardware counters can be read within few tens of cycles [19] and hence can be used by operators on NFs running in production with minimal overhead to NFs.

Open questions: Are CPU events useful for estimating utilization for polled NFs? Specifically, (1) can we select a small number of CPU events whose values can be accurately mapped to an NF's utilization level? (2) Is the mapping function robust across variations in workload for that NF? (3) Is there a universal estimation function that can map a fixed set of CPU events and their counter values to utilization for all NFs with a low percentage error? (4) How long does it take to estimate an NF's utilization using this approach?

3 An empirical study of estimation functions

An *estimation function* (or an *estimator*) on a set of CPU events accepts as input the frequency of occurrences of those events at a CPU core and outputs a percentage value from 0 to 100 that represents the utilization for the NF running on that core. In this section, we define the types of estimation functions we evaluate, describe the experimental setup to collect data for our evaluation and present statistical metrics on the accuracy of our estimation functions.

3.1 Local and universal estimators

We evaluate estimation functions under these key assumptions. Since CPU events are dependent on the CPU architecture and its configuration, we assume that the CPU used for evaluation as well as its BIOS settings remain unchanged. In practice, the estimators can be updated upon generational changes in hardware or major configuration changes. An NF can run on one more cores on a physical server. However, each core is assumed to perform identical packet processing so that an

estimator for a single core can easily generalize to a multi-core NF [23, 30]. Henceforth, we assess the accuracy of single-core NFs. To assess the NF throughput corresponding to a 100% utilization, our evaluation assumes that the CPU is the bottleneck resource for the NF.

Let N be the set of NFs deployed and C_ALL be the set of CPU events that can be counted.

Local estimator A local estimator L_{nC} outputs the utilization of NF $n \in N$ given the counts of a set of CPU events $C \subset C\_ALL$.

Universal estimator A univeral estimator U_C outputs the utilization for any NF given the counts of a set of CPU event $C \subset C\_ALL$.

To train the estimators, we build a set of test profiles T_n for each NF $n \in N$. A test profile includes traffic characteristics such as packet sizes and flow lengths as well as an NF's internal configuration such as routing table sizes for a router or the size of memory cache for a proxy. For each traffic profile, the NF is subject to a set of loads L in the range 0-100, where a load of 0 means no traffic and a load of 100 means a traffic volume equal to the maximum throughput of the NF for that test profile. A local estimator L_{nC} is trained based on data from the set of experiments $T_n \times L$, and an universal estimator U_C is trained based on data from the set of experiments $\bigcup_n T_n \times L$.

3.2 Experimental methodology

Network functions: We have evaluated three NFs with varying amounts of computation and state.

A layer 3 forwarder (**L3FWD** for short) matches the destination IP of an incoming packet to the entries stored in its forwarding table and updates the layer 2 header before sending the packet to the next-hop router. The forwarder is a stateless NF because it does not maintain per-flow state. Our experiments are conducted with the L3FWD application included in the DPDK 18.02 release [11].

A layer 4 load balancer (**L4LB** for short) multiplexes connections to a pool of servers [24]. In order to cope with a changing pool of servers, it creates a flow table entry upon the arrival of the first packet of a new connection. The flow table entry stores a connection identifier as the key (e.g., the TCP/IP 5-tuple) and the chosen server as the value. We evaluate a homegrown load balancer implemented as a DPDK application using nearly 3K lines of code. It performs common flow table management tasks such as flow table resizing and deletion of flow table entries.

A deep packet inspector (**DPI** for short) identifies traffic matching a set of pre-configured rules. The rules can specify layer-3 and layer-4 header fields, application-layer rules, e.g., URL patterns, and strings in the packet data. It analyzes data sent over a connection by reconstructing its bytestream from individual packets. A DPI maintains per-flow state and performs non-trivial computation on each flow. We use an open-source DPI library nDPI [4] and integrated it with a DPDK forwarding application to test its network performance.

Test profiles: Our test profiles T_n for NFs depend on their respective performance metrics.

L3FWD's performance is typically measured in packets per second [15]. We stress the computation done by L3FWD by evaluating traffic profiles with minimal-sized packets (64B) as well as larger packet sizes (128B, 256B, 512B). L3FWD is configured to apply a hash-based lookup algorithm on a routing table configured with 10K entries.

An L4LB should support a high rate of both packets and connections [24]. Our test profiles vary the number of packets per connection from 10 to 1000. All packets are minimal-sized to test the packet throughput. L4LB uses a flow table with 10M entries, which is sufficient to store all flow entries.

A DPI should support a large number of connections and a high bandwidth (in bits/sec) while examining the packet payloads. Our test profiles include HTTP connections of varying length from 1 KB to 1024 KB. Smaller connections test the connection throughput whereas larger connections test the bandwidth. Our DPI NF is configured to perform transport protocol-based, hostname-based and IP-based matching.

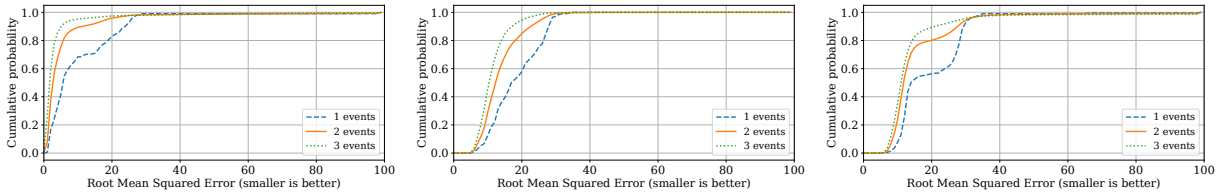
Estimators: We first generate ground truth values of NF utilization. We determine the peak load for an NF and a test profile using a binary search approach (similar to RFC 2544 [3]) under the constraint that the drop rate stays below 0.01%. We subject the NF to 10%, 20%, $\cdots$, 100% of the peak load. At each load, we collect all available CPU events using EMON [7] taking 4 sec measurements for each event.

We use linear regression to estimate the percentage utilization based on a feature set consisting of event counts of 1, 2 and 3 events. Among $\binom{|C_{ALL}|}{c}$ possible estimators with c event counts, we evaluate all estimators for c = 1 and c = 2. For c = 3, we evaluate estimators by uniformly sampling 1% of estimators from a total of 1.4M possible estimators. The aforementioned ground truth utilization data serves as the training dataset for the regression models. We perform 3-fold cross-validation on the training dataset and use the average root mean squared error [29] as the error metric.

Testbed: We conduct experiments on two servers, each equipped with 2x10G network interfaces (Intel 82599ES 10 Gbps SFI/SFP+ NIC) and two CPUs (Intel Xeon E5-2680 v3). The two servers running a traffic generator and an NF respectively are connected via a 10G switch. We use a scalable, open-source DPDK traffic generator TRex [26]. L4LB and DPI are tested using TRex's stateful mode, which can generate millions of new connections per second.

3.3 Estimation errors

Local estimators: We tabulate the errors for the best estimators in Table 1. We find that the best local estimators using 3



(a) L3FWD (b) L4LB (c) DPI

Figure 2: Distribution of estimator score (root mean squared error) for tested NFs.

Estimator	Root mean squared error		
	1 events	2 events	3 events
L3FWD Local	1.79	0.80	0.51
L4LB Local	6.55	5.23	5.17
DPI Local	8.56	6.64	5.98
Universal	13.86	10.98	9.89

Table 1: Cross-validation errors for the best estimators.

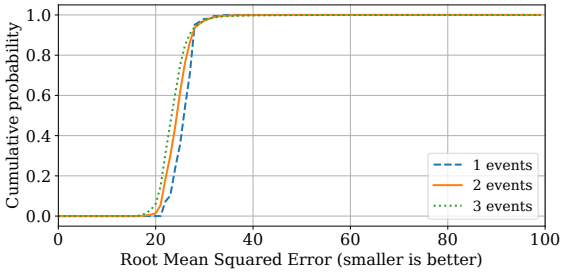


Figure 3: Error distribution for universal estimators.

CPU events for all NFs achieve an error rate of 0.5%, 5.2% and 6.0%. This finding shows that there exist simple local estimators for NFs that can achieve low error rates despite significant variations in test profiles. Stateful NFs (L4LB and DPI) have higher error rates than the stateless L3FWD, which is likely due to more complex processing and additional state maintained by these NFs. These error rates may be acceptable in many use cases such as power management and workload placement, which can design incorporate a utilization estimation error in making decisions.

We show the distribution of errors for local estimators of all NFs in Figure 2. The median error rates for single-event estimators are 6.5% , 17.8% and 14.9% for the three NFs which are considerably higher than those of the respective best estimator. This validates our data-driven approach to select the estimator functions, since randomly selecting CPU events is likely to yield higher errors. We also note that stateful NFs have higher errors than the stateless NF across the entire distribution, which further shows that stateful NFs are more difficult to model.

For all NFs, using two events to estimate the utilization yields lower error rates over a single event estimator across the entire CDF. Median error for estimators using two events is lower than single event estimators by 4.5%, 2.8% and 2.2%. The marginal improvement of adding the 3rd event is smaller in many cases. This trend suggests that using additional events in a linear regression model may yield smaller improvements.

Universal estimators: We test the accuracy of universal estimators built using data collected across all NFs and their profiles. All data is collected at a fixed CPU frequency (1.2 GHz). We use an equal number of data points for each of the NFs in training and validating the universal estimators.

The best universal estimator (Table 1) using 3 events has an error rate of 9.9%, which is higher than the error of local estimators, especially that of L3FWD. Figure 3 shows the distribution of errors for universal estimators, where the median error rates are 26.4%, 22.0%, 19.2% for estimators based on 1, 2 and 3 events respectively. Given that there are no techniques to estimate utilization of unmodified polled IO NFs, these estimators represent an advancement of the state of the art. But, due to error rates of 10% or more, we need to be conservative in using results from these universal estimators.

4 Evaluation

We conduct a preliminary evaluation to test the accuracy of estimators developed in Section 3 on previously unseen profiles. We also measure the impact of the duration of counter measurement on the overall accuracy.

Experimental setup: We use test profiles with heterogeneous traffic characteristics. L3FWD's profile contains a mix of packet sizes from 64B to 278B. L4LB's profile contains a mix of flows of length 10 to 100 packets. DPI's profile is an HTTP browsing workload included with TRex. For each NF and its test profile, we calculate the peak load (as described in Section 3.3) and test each NF at loads of 10%, 20%, $\cdots$, 100%. For each NF, we selected the local estimator with 2 events which showed the least error (Table 1) for evaluation. CPU events of the chosen estimators' are shown in Table 2.

We treat the duration of counter measurement as a parameter, which is varied from 10 us to 1 s in this experiment. We measure the counter values 100 times for each value of the

NF	CPU events
L3FWD	ILD_STALL.LCP
	L2_TRANS.RFO
L4LB	L1D_PEND_MISS.REQUEST_FB_FULL
	BR_INST_EXEC.TAKEN_CONDITIONAL
DPI	CYCLE_ACTIVITY.CYCLES_L2_PENDING
	MEM_LOAD_UOPS_RETIRED.L2_MISS

Table 2: Events chosen for building evaluated estimators [8].

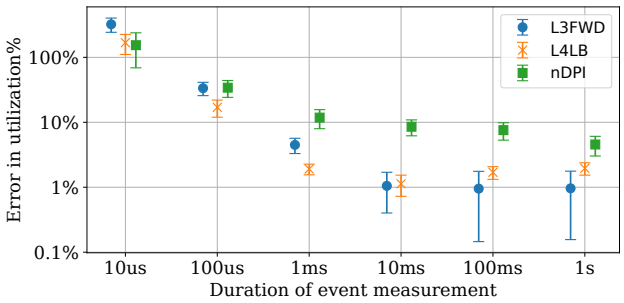


Figure 4: Utilization estimation errors on unseen test profiles.

parameter. For each measurement, we calculate the absolute error between the actual utilization and the utilization estimated by the local estimator. Figure 4 shows the average and the standard deviation of this error across all loads and all measurements of the counter values.

Results: Estimation errors are the least for L3FWD, followed by L4LB and DPI, which is consistent with the trend observed earlier. The average error of these estimations for a 1 s counter measurement are 0.9%, 1.9% and 4.5% respectively, which are lower relative to the cross-validation errors in Table 1. This result suggests that our estimators can generalize beyond the specific profiles they are trained on. But, we need experiments with real traffic traces to strengthen this claim.

The interval of counter measurements affects estimation errors. A 10 us measurement interval has an average error of more than 100%. But, increasing the measurement interval beyond 1 ms has smaller improvements for all NFs. What the best value of this interval is may depend on the use case. Workload placement at minutes or hours timescale can use counter measurements over a few seconds for greater accuracy. But, an aggressive CPU power management policy may want an up to date utilization, for which a millisecond measurement may be a better choice despite its higher error rates.

Our linear regression-based estimator is fast in the common case (median = 0.3 ms). Thus, the estimator is only a small fraction of the overhead provided that the counters are measured for 1 ms or longer.

5 Conclusions and future work

Network cloud management and automation require a common interface to report utilization of NFs. To this end, we explore a CPU event-based estimation approach that is applicable to black-box NFs.

Improved universal estimators: Our experiments show that CPU events are not a "silver bullet" to the complex problem of estimating utilization of polled NFs. Simple universal estimators based on up to three CPU events have an error rate of 10% in the best case. However, simple local estimators do estimate utilization within a percentage point in some cases. We plan to explore universal estimators based on a few tens of CPU events, which can be still be collected in a few milliseconds. Such an estimator could be practical for scenarios that do not need sub-millisecond response time. We can apply non-linear estimation techniques such as those based on neural networks to improve universal estimators.

Virtualized and/or cross-core processing: We evaluated stateless and stateful NFs and found the estimation errors to be higher for stateful NFs. We plan to explore two more kinds of NFs in the future: virtualized NFs and NFs that perform cross-core processing. Virtualized polled IO NFs using network interfaces that support SR-IOV (a form of a virtual PCI device) can perform close to an NF that has direct physical access to network interfaces [9]. We are studying whether estimators need to be rebuilt if an NF is run in a VM on an SR-IOV interface. Many multi-core NFs dedicate one or more threads to poll hardware queues for Rx and multiplex them among worker threads via software queues [14]. Worker threads poll these software queues to read packets. Unlike NFs we have evaluated, these NFs may need different estimator functions for Rx threads and worker threads. Determining whether Rx or worker thread is the bottleneck is a challenge for such NFs.

Use cases: We see a number of applications of CPU event-based estimators. A top-like utility for NFs can be made available for system administrators. With community help, the utility can be packaged with a set of estimators for common CPUs, so that the utility can be used out of the box. For cloud orchestration tasks, the utility would implement a northbound API. Cloud orchestration software (e.g., OpenStack [22]) can query that API to optimize resource allocation (number of cores), workload placement and load distribution. Power management using frequency scaling and/or sleep states seems an urgent need for polled IO NFs, which run constantly at 100% CPU utilization [20]. A key question for frequency scaling is whether estimators trained at a given CPU frequency work at other frequencies using simple interpolation. Using CPU events to improve the trade-off between power consumption and performance (latency, jitter and packet loss) is an interesting area of exploration.

A Workshop discussion

The central question before the workshop audience is whether our effort to develop a CPU event-based universal estimator worthwhile or is it easier to define a common API and have NF vendors implement and maintain that API for their respective NFs. There appears to be preliminary work in DPDK community towards the second direction. In the long run, other automated techniques are possible such an intelligent compiler to perform automated NF instrumentation, which can simplify the task of implementing a utilization API.

A limitation of this approach is that the accuracy of utilization estimators cannot be increased arbitrarily by using complex machine-learning models as doing so would affect the end-to-end estimation latency. Such high latency might be unsuitable for critical use-cases like power management. Secondly, the behavior of a universal estimator is dependent on the NFs used to train it. Even in the cross-validation experiments, each estimator was trained on datapoints from all NFs. We haven't tested the accuracy of a universal estimator on an unseen network function. Furthermore, an estimator might have a high error if the traffic profile it is subjected to has significantly different characteristics than the profiles in training data. We rely on the assumption that a network provider like AT&T would have insight about representative profiles for a given NF.

References

[1] Glenn Ammons, Thomas Ball, and James R. Larus. Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling. *SIGPLAN Not.*, 32(5):85–96, May 1997.

[2] Anuj Kalia and Michael Kaminsky and David Andersen. Datacenter RPCs can be General and Fast. In *USENIX NSDI*, 2019.

[3] Scott Bradner and Jim McQuaid. RFC2544-Benchmarking Methodology for Network Interconnect Devices. *URL: ftp://ftp. rfc-editor. org/in-notes/rfc2544. txt (March 1999)*, 1999.

[4] NDPI by NTOP. Open Source Deep Packet Inspection Software Toolkit. https://github.com/ntop/nDPI. Accessed: 2019-05-14.

[5] Lianjie Cao, Puneet Sharma, Sonia Fahmy, and Vinay Saxena. ENVI: Elastic Resource Flexing for Network Function Virtualization. In *USENIX HotCloud*, 2017.

[6] AMD Corporation. Instruction-Based Sampling: A New Performance Analysis Technique for AMD Family 10h Processors. http://developer.amd.com/wordpress/media/2012/10/AMD_IBS_paper_EN.pdf. Accessed: 2019-05-14.

[7] Intel Corporation. EMON: User's guide. https://software.intel.com/en-us/download/emon-users-guide. Accessed: 2019-05-14.

[8] Intel Corporation. Intel Processor Events Reference. https://download.01.org/perfmon/index/. Accessed: 2019-05-14.

[9] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan. High Performance Network Virtualization with SR-IOV. *Journal of Parallel and Distributed Computing*, 72(11):1471–1480, 2012.

[10] DPDK. DPDK Data Plane Development Kit. https://www.dpdk.org. Accessed: 2019-05-14.

[11] DPDK. DPDK L3 Forwarding Sample Application. https://doc.dpdk.org/guides/sample_app_ug/l3_forward.html. Accessed: 2019-05-14.

[12] ETSI. Network Functions Virtualisation. https://portal.etsi.org/NFV/NFV_White_Paper.pdf. Accessed: 2019-05-14.

[13] S Gallenmüller. Comparison of Memory Mapping Techniques for High-speed Packet Processing. *Technical University of Munich*, 2014.

[14] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. SoftNIC: A Software NIC to Augment Hardware. *Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-155*, 2015.

[15] Intel. DPDK Performance Reports. https://core.dpdk.org/perf-reports/. Accessed: 2019-05-14.

[16] Gueyoung Jung, Matti A Hiltunen, Kaustubh R Joshi, Richard D Schlichting, and Calton Pu. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In *IEEE ICDCS*, 2010.

[17] Kai Li, Ming Shen, and Chuanpeng Zhong. I/O System Performance Debugging using Model-driven Anomaly Characterization. In *USENIX FAST*, 2005.

[18] Min Yeol Lim, Allan Porterfield, and Robert Fowler. SoftPower: Fine-grain Power Estimations using Performance Counters. In *ACM HPDC*, 2010.

[19] Yan Liu. Optimizing PAPI for Low-Overhead Counter Measurement. 2017.

[20] Luca Niccolini, Gianluca Iannaccone, Sylvia Ratnasamy, Jaideep Chandrashekar, and Luigi Rizzo. Building a Power-proportional Software Router. In *USENIX ATC*, 2012.

[21] NTOP. PF_RING. https://github.com/ntop/PF_RING. Accessed: 2019-05-14.

[22] Openstack. Openstack. https://www.openstack.org. Accessed: 2019-05-14.

[23] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. NetBricks: Taking the V out of NFV. In *USENIX OSDI*, 2016.

[24] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, et al. Ananta: Cloud Scale Load Balancing. In *ACM SIGCOMM*, 2013.

[25] Patrick Fay Thomas Willhalm, Roman Dementiev. Intel Performance Counter Monitor: A better way to measure CPU Utilization. https://software.intel.com/en-us/articles/intel-performance-counter-monitor. Accessed: 2019-05-14.

[26] TRex. Cisco Systems. TRex: Cisco's Realistic Traffic Generator. https://trex-tgn.cisco.com. Accessed: 2019-05-14.

[27] Hristo Georgiev Trifonov. Traffic-aware Adaptive Polling Mechanism for High Performance Packet Processing. *University of Limerick Institutional Repository*, 2017.

[28] Linux Perf Wiki. perf: Linux Profiling with Performance Counters. https://perf.wiki.kernel.org/index.php/Main_Page. Accessed: 2019-05-14.

[29] Wikipedia. Root-mean-square deviation. https://en.wikipedia.org/wiki/Root-mean-square_deviation. Accessed: 2019-05-14.

[30] Wei Zhang, Abhigyan Sharma, Kaustubh Joshi, and Timothy Wood. Hardware-assisted Isolation in a Multi-tenant Function-based Dataplane. In *ACM SOSR*, 2018.