Towards Portable End-to-End Network Performance Characterization of SmartNICs

Tong Xing[*]
The University of Edinburgh

Hesam Tajbakhsh
Dalhousie University

Israat Haque
Dalhousie University

Michio Honda
University of Edinburgh

Antonio Barbalace
The University of Edinburgh

ABSTRACT

Emerging accelerated network devices, including SmartNICs (SNICs) with general-purpose CPUs – such as ARM, create new sources of queuing and processing delay in the hardware, network stack, and application. We therefore argue for the need of a framework to measure the application and network stack performance in SNIC-based host systems.

This paper describes our methodology to build up such framework focusing on the regular Linux network stack executed on either host, or SNIC CPUs. Our framework shows how SNICs impact end-to-end latency, throughput, and multi-core scalability based on the SNIC architecture, SNIC CPU performance, and even the applications themselves.

CCS CONCEPTS

• **Networks → Network performance analysis**; • **Hardware → Networking hardware**.

ACM Reference Format:
Tong Xing, Hesam Tajbakhsh, Israat Haque, Michio Honda, and Antonio Barbalace. 2022. Towards Portable End-to-End Network Performance Characterization of SmartNICs. In *The Thirteenth ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '22), August 23–24, 2022, Virtual Event, Singapore.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3546591.3547528

1 INTRODUCTION

SmartNICs (SNICs), which are network interface cards (NICs) that integrate programmable or reconfigurable processing units on their own board, are becoming widely popular, counting several products available on the market, including Fidus Sidewinder [30], Cavium LiquidIO [4], Broadcom Stingray [2], and Mellanox BlueField [20]. There is a renewed interest in the research community towards understanding what applications may benefit from being offloaded to SNICs, with the goal of achieving higher compute capacity, lower latency, reduced power consumption, etc.

SNICs add a certain degree of heterogeneity that may exacerbate programmability within the host system. Ming *et al.*[17] considered CPU-based and FPGA-based SNICs, and built a software framework to ease their programmability. Microsoft Azure adopts FPGA-based smartNICs to reduce applications' network latencies, while introducing a software layer to ease programmability – as FPGAs are difficult to program to non-experts [8]. Nevertheless, the use of FPGA-based SNICs have been limited mainly to network function processing, including NetFPGA [31]. To overcome the intrinsic difficulties of FPGA-based SNICs, [9] adopts P4-enabled SNICs, where P4 cores relieve the server's host CPU(s) from network-related workloads. Similarly, [18] uses a MIPS based SNIC to offload microservice-based applications.

This paper focuses on general-purpose CPU-based SNICs because they can be programmed with familiar programming languages and run familiar software – even Linux, and advocates the need for a portable framework that enables the end-to-end performance characterization of SNIC systems. General-purpose CPU-based SNICs are increasingly popular and have not been fully studied in the literature. Specifically, in the context of regular applications that run atop a traditional operating system, like Linux, but are deployed on SNIC CPUs vs host CPUs. Since SNIC CPUs are different than host CPUs, and different SNICs adopt different hardware architectures, it is important to characterize their end-to-end network performance and compare that with what achievable on the host CPU. This will enable system designers to answer all of a set of questions, including what application should be moved (offloaded) from host to SNIC CPUs, for higher capacity, lower latency, reduced power, etc. This paper introduces our *initial prototype* of *a portable software-only framework* for the end-to-end network performance characterization of different SNIC, targeting widely-used fully-fledged operating systems, at the moment Linux. Such framework includes a collection of (new and existent) microbenchmarks and applications, as well as a measurement infrastructure.

Contributions. This paper makes two key contributions. *First*, we introduce a collection of test programs, and a set of portable measurement methodologies to characterize CPU-based SNIC performance, which exposes both the architectural and quantitative difference of SNICs. Our methodologies differ from previous ones [17] that use kernel-bypass techniques, because we believe many operators wish to deploy regular applications in CPU-based SNICs together with an widely-used Linux system, and their interest is the end-to-end performance that includes software overheads of both application(s) and OS, rather than the performance with the minimum software overheads based on custom, specialized applications or OSes (i.e., the maximum achievable by the hardware, which can be extrapolated in a product datasheet or based on previous studies).

The *second* contribution is a snapshot of the performance of popular SNICs currently available on the market – which demonstrate the portability of our framework. We examine Broadcom Stingray PS225 and Mellanox BlueField2, which adopt different architectures and SNIC CPUs, and indeed exhibit different performance characteristics. Using our measurement methodologies, we found that the end-to-end performance is impacted by the SNIC architecture, the SNIC CPU performance, but also by the applications themselves – even when their logic seems similar, suggesting that deploying latency-sensitive applications on SNIC CPUs requires careful evaluation.

[*]Tong Xing, Hesam Tajbakhsh, and Antonio Barbalace did part of this work when at Stevens Institute of Technology, Hoboken, NJ.

	Stingray	BlueField2
CPU	8-core ARMv8 A72 (64bit) at 3.0 GHz	8-core ARMv8 A72 (64bit) at 2.5 GHz
Caches	16MB total cache (8MB L2 + 8MB L3)	1MB L2 cache per 2 cores, 6MB L3 cache
Memory	16GB Two channels on-board DDR4-1600	16GB Two channels on-board DDR4-1600
Network IF	10/25GbE, RoCE	25/50/100GbE, RoCE
Host IF	8x PCIe Gen 3	16x PCIe Gen 4

Table 1: Technical specifications of the SNICs used in the experiments (Broadcom Stingray PS225, and Mellanox BlueField2).

2 BACKGROUND AND MOTIVATION

A *smartNIC* (SNIC) is a unique technology where a processing unit is deployed into a network interface card (NIC). Such a processing unit can be programmable, like a multicore CPU, or reconfigurable, like an FPGA. Adding processing units to NICs enables a series of advantages, including early packet processing, and offloading network-related processing from the host processor to the NIC CPU/FPGA [7], and others [12]. In this paper, with the goal of reducing confusion, we will refer to traditional NICs without any programmable or reconfigurable processing unit as *regular NICs*, while we will use the term NIC generically to refer to the network interface hardware. Thus, in our terminology, both SNIC and *regular NIC* integrate a NIC.

SNICs and *regular NICs*, as well as other high-bandwidth peripheral devices, including storage drives and video cards, are connected to the host CPU via the Peripheral Component Interconnect Express bus – PCIe. PCIe is the de-facto standard peripheral bus in the computer industry, actively under development. It is a high-bandwidth, low latency, low power serial bus, introduced in the 2000s [25], now nearing its 5th generation. Today, PCIe generation 3 is the most widely deployed, while a few machines with PCIe generation 4 are already available on the market.

Transferring data via the PCIe bus has a non negligible per-transaction cost. Hence, while convenient for large-data transfers, small data-transfers have a cost in the µs scale [21] independently of the size. Therefore, if a network application needs to make many (small) data transfers through PCIe, it can be a good candidate to run on SNIC, which may lead to a reduction of the PCIe overheads of multiple data movements.

Running an application on SNICs CPUs has also other advantages, in fact, far less energy is consumed due to the lower SNICs' power consumption (limited by the PCIe bus at around $12W$ per card). SNICs are already used to offload data-center services, like distributed storage [12], freeing host CPUs from such overheads, and increasing the server capacities of processing data-center's customers requests – that can be actually monetized, hence increasing revenues. Moreover, applications running on a SNIC are isolated from the applications running on the host CPU, so the system's administrators can provide SLA for certain services.

2.1 A Categorization of SmartNICs

Figure 1 shows several existent *SmartNIC* architectures. As mentioned already, SNICs may integrate general-purpose CPUs (e.g., ARM, MIPS) or FPGAs, while since quite a long time NIC's ASICs integrate special-purpose processing units (accelerators) for simple packet-processing, including encryption/decryption or reassembly [1, 23] (not shown in figure).

SNIC architectures can be categorized based on their processing units' connection to the actual network interface, i.e., based on the network packet flow. Thus, similarly to [17], we discern SNICs with on-path processing units from SNICs with off-path processing units, and hybrids exist.

A SNIC with on-path FPGA is characterized by an FPGA-based network interface, Figure 1a and Figure 1b; and includes low-level network packet processing in the FPGA itself. However, the two configurations are quite different. The first, which can be realized with any Xilinx Zynq-based products (e.g., Fidus Sidewinder [30]) has an on-path FPGA and an off-path CPU. The FPGA directly interfaces with the Ethernet network and also with PCIe, in principle, providing all traditional NIC's functionalities. The second has an FPGA placed before the actual network interface – the FPGA can provide services instead of the host machine, faster. Microsoft Catapult project implemented this configuration [5]. Figure 1c and Figure 1d depict a SNICs with an off-path CPU and with an on-path CPU. Examples of the first are the Broadcom Stingray [2] and the Mellanox Bluefield [20], while the Liquid IO [4] is an example of the latter. Note the Mellanox Bluefield can also be configured such as in Figure 1d. Independently of the architecture, all such SNICs are being used in an increasing number of research works [3, 8, 9, 11, 13, 15, 17, 19, 27, 29, 31].

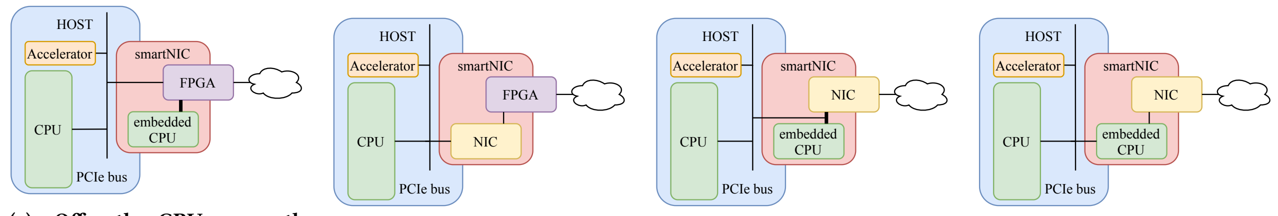
2.2 Research Question(s)

Clearly, SNICs with an on-path CPU or FPGA, Figure 1d and Figure 1b respectively, provide the lowest latencies in network packet/request processing when compared to the same processing executed on the host CPU due to the proximity to the network interface – this has been shown in different publications, including [17, 24, 27, 28]. Moreover, FPGA-based SNICs, such as the architectures in Figure 1a and in Figure 1b, have been studied extensively in the literature [3, 14, 31] demonstrating faster processing for specific applications. Only a few works have been published about off-path SNICs [27], whose full performance advantages are still not clear, especially when compared to the host network latencies (and achievable bandwidths), nor contrasted between models. Moreover, as the number of SNIC products is increasing, a portable way to easily characterize them may be beneficial for the community.

3 RELATED WORK

The most related and complementary SNIC-based performance characterization is described in [17], where the authors characterized four different SNICs from Marvell, Broadcom, and NVIDIA/Mellanox (including on-path and off-path configurations) to understand benefits and challenges of deploying distributed applications such as key-value stores. However, such performance characterization doesn't provide an in-depth analysis of the latencies and bandwidths, nor their benchmarks are portable, which are the targets of our work. Recently, Liu et al. [16] come up with a fairly complete network and compute characterization of a specific SNIC, the BlueField-2. Differently, our work is a more in-depth and portable – applicable to all CPU based SNICs, characterization of the networking latency and contrasts them for different SNICs.

Hardware latencies, specifically PCIe, of NICs have been studied by several works, most noticeably by Neugebauer et al. [21]. Such work has been carried on in the context of FPGA-based SNICs, and although it is not portable, it is complementary to our analysis –



(a) Off-path CPU, on-path FPGA (b) On-path FPGA (c) Off-path CPU (d) On-path CPU

Figure 1: Several existent SmartNIC architectures.

which also aims at highlighting PCIe costs, but we don't focus on that, instead we look at the end-to-end numbers for SNICs and host CPUs, while comparing them.

Finally, a large amount of works explored what workloads are more suitable to run on SNICs. For example, the authors of [26] deployed an FPGA-based SNIC to accelerate the performance of a key-value store database application (memcached). Qiu *et al.*in [22] provided a tool for predicting the performance offloaded task into FGPA-based SNIC. Firestone *et al.*proposes to offload networking in a virtualized environment into SNICs. In [6], a P4-enabled SNIC is adopted for several applications such as KV-store, web server, and image transformer, where the SNIC is equipped with hundreds of RISC cores.

4 CHARACTERIZATION FRAMEWORK

Our goal is to build a software-only, portable (i.e., run on any SNIC with a traditional OS, like Linux, BSD, Windows, etc. without kernel modifications) SNIC performance characterization framework. Our measurement methodology for Linux is based on eBPF to identify the cost of different kernel subsystems during network request processing. In the current prototype, we take timestamps at least at the syscall interface and at the bottom of the network stack, as close to hardware as possible – using `tc-eBPF`. Since eBPF is under development for Windows and FreeBSD, we believe a similar methodology will be feasible in the near future.

We use the regular Linux network stack in our measurement, and this is a stark contrast to the existing work [17], which employs kernel-bypass techniques like DPDK and RDMA. This is because we expect SNIC CPUs usually run Linux and its network stack in reality, as it implements various network protocols and features that many kernel-bypass network stacks do not implement, such as tunneling protocols, TCP extensions, filters and queuing disciplines.

We develop microbenchmarks, including UDP-based echo client and server that implement our simple timestamp-based measurement protocol. In addition, we collected widely-used production applications such as Memcached, Redis, and a Function-as-a-Service (FaaS) runtime – OpenLambda, as well as corresponding workload generators. While this is a work in progress, we are planning to open-source the current prototype upon paper acceptance.

5 EXPERIMENTS

The use of SNICs could impact latency-critical applications or scenarios due to SNIC architecture and embedded CPU processing inside the SNIC. We set out to test these effects using our SNIC testbed and measurement tools described earlier. We run microbenchmarks to measure microscopic network data behavior; we then run real-application benchmarks to see end-to-end latency and efficiency of the SNIC CPUs.

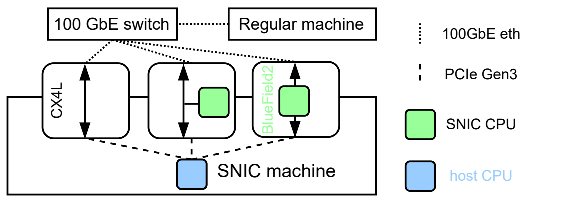


Figure 2: The SNIC Testbed. Details in Section 5.1.

5.1 Experimental Setup

Hardware: We use a pair of machines (Figure 2). The *SNIC machine* is equipped with two Xeon Gold 6230R CPUs. It installs a Mellanox ConnectX-4 Lx MT27710 100GbE NIC (CX-4Lx), Broadcom Stringray PS225 25GbE SNIC (Stingray) and Mellanox BlueField2 100GbE SNIC (BlueField2); details of these SNICs are summarised in Table 1. Unless otherwise stated, we configure BlueField2 in the on-path mode. The other machine, which we call *regular machine*, is equipped with a Xeon Silver 4110 CPU and installs a Mellanox ConnectX-4 MT27700 100GbE NIC (CX-4). All the NIC ports connect to a 100GbE Ethernet switch with a breakout cable for the Broadcom 25GbE SNIC.
Software: We install the Linux 5.4 kernel optimized for performance, which disables various debugging supports, retpline and netfilter, in all the hosts; embedded CPUs of Broadcom PS225 run Linux kernel 4.14.79 and those of BlueField2 SNIC run the Linux kernel 5.4; both are provided by the vendors. All the NICs disable interrupt mitigation to optimize for latency.

5.2 Basic End-to-End Latency

To see the effect of SNICs on the packets that traverse them, we wrote custom echo server and client programs that run on host or SNIC CPUs. Since queues form in various stages in the host network stack and SNIC, it is crucial to measure latency with *bursts* of packets. Our client thus sends one or more UDP datagrams in a batch and waits for the packets *echoed* by the server. These programs use `sendmmsg()` and `recvmmsg()` to minimize the context switches. We run the client and server programs in either the SNIC machine or regular machine.

Table 2 shows RTTs measured at the client; we discuss the memcd, Redis and Lambda columns later. RTTs between the host CPUs over different NICs are similar (18.22–18.73µs and 22.42–23.99µs for 64B and 1024B datagrams, respectively) when CX-4Lx or Stingray is used (first and second rows) in the SNIC machine. This is reasonable because Stingray adopts the *off-path* architecture (Figure 1). When the client or server runs in Stingray's SNIC CPU (third row), RTT is slightly longer (17.69–18.66µs for 64B datagram and 24.05–24.73µs

NIC	Clie. CPU	**64B [µs]**	**1KB [µs]**	**memcd 64B [µs]**	**memcd (1KB) [µs]**	**Redis (64B) [µs]**	**Redis (1KB) [µs]**	**Lambda (64B) [µs]**	**Lambda (1KB) [µs]**
CX-4Lx	host	**18.22 (18.42)**	**23.28 (22.42)**	**28.36 (25.58)**	**28.87 (25.85)**	**40.84 (38.32)**	**41.10 (38.51)**	**(3330)**	**(3792)**
Stingray	host	**18.73 (18.31)**	**22.37 (23.99)**	**27.68 (26.49)**	**28.37 (27.62)**	**43.09 (38.65)**	**43.36 (40.18)**	**(3384)**	**(3756)**
Stingray	SNIC	**17.69 (18.66)**	**24.05 (24.73)**	**29.05 (29.14)**	**31.91 (30.48)**	**43.69 (37.38)**	**46.50 (39.01)**	**(2519)**	**(3253)**
BlueField2	host	**21.13 (20.40)**	**23.02 (22.02)**	**29.26 (27.53)**	**29.91 (28.04)**	**44.13 (40.24)**	**45.47 (44.86)**	**(3415)**	**(3794)**
BlueFIeld2	SNIC	**39.15 (45.80)**	**41.85 (49.13)**	**67.78 (70.71)**	**73.25 (75.81)**	**77.01 (80.33)**	**83.68 (88.70)**	**(4242)**	**(4280)**
Offpath-BlueField2	host	**19.57 (19.45)**	**21.60 (21.38)**	**28.28 (27.11)**	**29.14 (27.85)**	**43.71 (39.49)**	**45.20 (39.80)**	**(3351)**	**(3787)**
Offpath-BlueField2	SNIC	**37.94 (46.21)**	**39.10 (49.79)**	**67.22 (55.56)**	**69.54 (60.64)**	**82.87 (67.11)**	**77.16 (67.91)**	**(4211)**	**(4274)**

Table 2: RTTs between the client and server applications. Numbers in parentheses are RTTs when the client and server are reversed.

for 1024B datagrams), because the SNIC CPU is slower than the host CPU.

The results with BlueField2 SNICs highlights the architectural implication of the *on-path* SNICs. When the server runs on the host CPU of the SNIC machine (fourth row), RTT is longer than those over the other NICs, because the traffic is mediated by the SNIC CPU. On the other hand, RTT increases even when the client or server runs in the SNIC CPU (fifth row) and thus the distance to the output port is shorter. This is likely because the SNIC CPU is slow. In addition, we observe significant RTT difference depending whether the SNIC CPU runs the client (39.15–41.85µs) or server (45.80–49.13µs) program. Figure 4 shows latency results with different packets sizes. We take a closer inspection of the CPU times spent in the SNIC software in the later section. We also ran the tests with BlueField2 in the off-path mode (the last two rows), and observe lower latency than the on-path cases. Those results confirm that the impact of the SNIC architecture or configuration.

Figure 3a shows RTTs over increasing numbers of the batch size for 1472B packets; Figure 3b shows those for the opposite request direction. Since packets are queued at various places such as the NIC queues and PCIe bus, RTT increases with the batch size. BlueField2 SNIC CPUs clearly exhibit higher RTT increase. This can be again explained by the on-path architecture of SNIC.

5.3 Latency Breakdown

To shed light on the software and hardware overhead inside SNICs, we measure the breakdown of the end-to-end latency using timestamps taken from several vantage points and embedded in the packet: TX system call (1) and low-level packet output routine (2) in the echo client kernel, low-level packet input routine (3) and output routine (4) in the echo server kernel, low-level packet input routine (5) and RX system call (6) in the echo client. Therefore, for example, we can derive the time elapsed in the echo client's network stack for TX and RX from (2) - (1) and (6) - (5), respectively.

We use `tc-eBPF` to take a timestamp at the low-level packet output and input routine and embed it in the UDP packet payload, except for the kernel running on a Stingray SNIC CPU; since the supported kernel version is too old, we directly patched the device driver.

Software overheads: Table 3 shows the RTT, software overheads and time spent in hardware for a 1024B packet. When the echo client runs in the Bluefield2 SNIC CPU (the bottom row) or the echo server does so (numbers in parentheses), the end-to-end latency is much higher than other cases, because the performance of the SNIC CPU is low and thus results in long processing delay in the SNIC. The

NIC	CPU	Clie.	Serv.	HW	RTT [µs]
CX-4Lx	host	3.7 (3.3)	4.0 (3.2)	15.6 (16.1)	23.3 (22.6)
Stingray	host	5.5 (3.4)	4.1 (3.3)	14.3 (15.7)	23.9 (22.4)
Stingray	SNIC	9.2 (3.4)	4.1 (7.7)	11.4 (12.9)	24.7 (24.0)
BlueField2	host	3.1 (3.1)	4.0 (2.8)	15.9 (17.1)	23.0 (23.0)
BlueField2	SNIC	22.5 (3.4)	4.1 (16.0)	22.5 (22.5)	49.1 (41.9)

Table 3: Software overheads, hardware time and RTT between the echo client and server for a 1024B message. Numbers in parentheses are times when the client runs in the regular host and the server runs in SNIC machine.

Stingray SNIC CPU (third row) also exhibits higher software overheads than the host CPU, but lower than the Bluefield2 SNIC CPU.

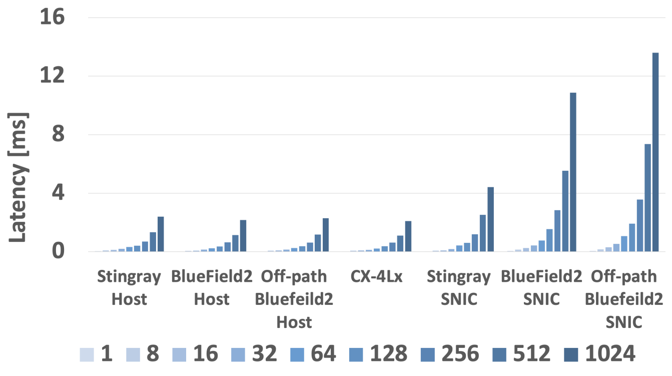
Hardware overheads: We can derive the time spent (mostly) in hardware, including the NICs and wire, from RTT and software overheads measured above. The Stingray SNIC exhibits lower hardware time with the SNIC CPU ($24.7-4.1-9.2=11.4$µs) than the host CPU ($23.9-5.5-4.1=14.3$µs), which matchesour expectation, because the SNIC CPU resides closer to the physical ports than the host CPU. On the other hand, the Bluefield2 SNIC shows longer hardware time with the SNIC CPU (22.5µs) than the host CPU (15.9µs). Those results indicate the importance of the measurement, because the actual performance on SNICs can differ from the intuition based on the hardware architecture, such as on-path or off-path SNIC CPUs.

Figure 4 plots RTT with smaller packets, and as shown in the bottom parts, software overheads are high in SNIC CPUs, similar to the 1024B packet cases. We also measured on the BlueField2 with the off-path configuration, but did not observe significant difference.

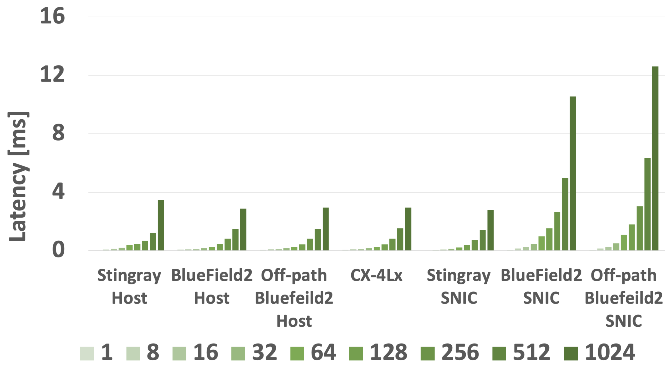
Finally, we analyze communication latency between the host and SNIC CPUs inside the host system (SNIC machine), measured using the same `sendmmsg()/recvmmsg()` programs. Figure 3c plots that latency for different batch sizes. BlueField2 exhibits higher latency, which appears to be mainly due to the slow SNIC CPU.

5.4 Application Performance

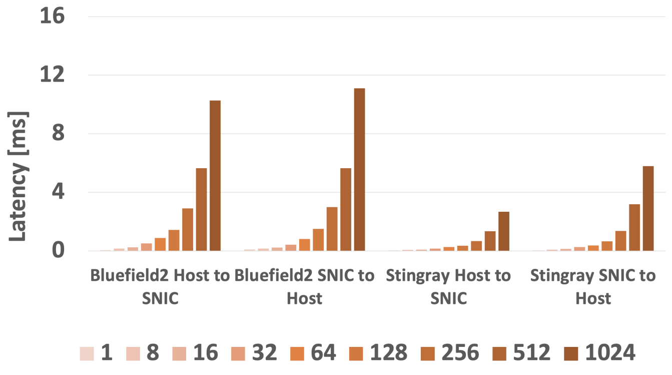
What is the application performance on SNICs? We chosen memcached and Redis, popular in-memory key-value stores, because those applications incur reasonable software overhead to implement application-level communication protocol over UDP (memcached) or TCP (Redis), consistency guarantee using locking and request parser, in addition to their data structures (e.g., hash tables). We further examine an HTTP server executed in OpenLambda [10] as a serverless function. We thus believe those applications are suitable to characterize the impact of latency caused by different SNIC architectures on



(a) Latency



(b) Latency in reverse directions.



(c) latency between host and SNIC CPUs inside the SNIC machine.

Figure 3: RTT with different batch sizes.

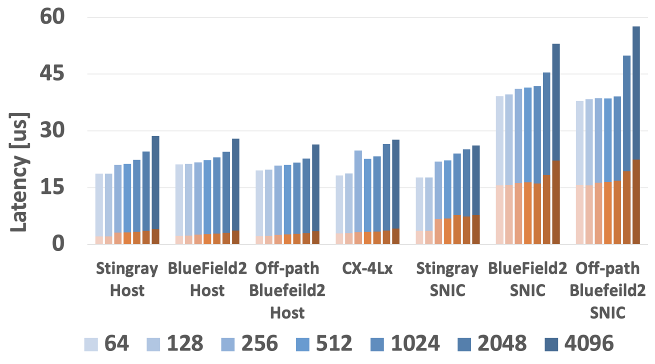


Figure 4: RTT with various packet sizes. Bottom parts indicate software overheads. Off-path BlueField2 means the off-path CPU configuration.

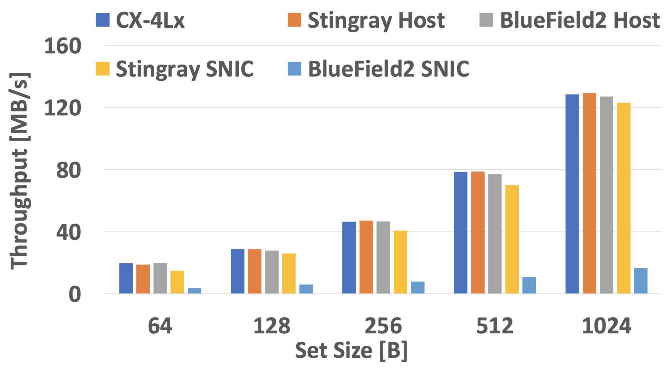
end-to-end performance. It should be noted that bandwidth demands of those applications never exceed the physical link capacity.

As before, we set off our experiment by simple RTT measurement with continual back-to-back requests. In Table 2, "memcd", "Redis" and "Lambda" columns show RTTs of memcached, Redis and OpenLambda, respectively. To be consistent with the other columns discussed earlier, when those applications run on the SNIC machine, numbers are represented in parentheses as they are *servers*; for example, when memcached runs on the Stingray CPU in the SNIC machine and its client runs in the regular machine, RTT with 64B messages is 29.14µs, and when memcached runs in the regular machine, RTT with the same message size is 29.05µs. We do not run OpenLambda in the regular machine.

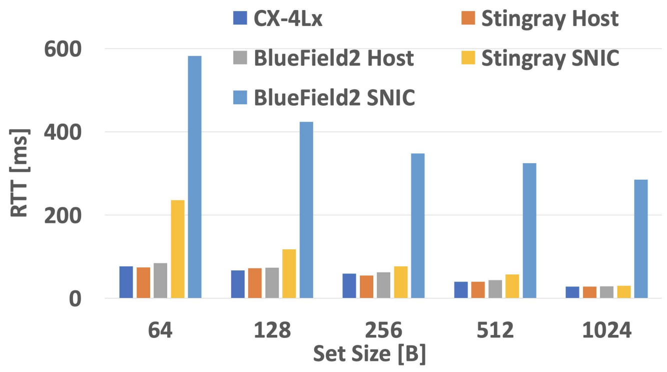
memcached and Redis use `set` requests for a 64B and 1024B object, and OpemLambda uses an HTTP request. For memcached and Redis, except for BlueField2 SNIC CPU cases, results are unsurprising, adding moderate (less than 10µs) latency to the simpler echo applications. The BlueField2 SNIC CPU cases are worth noting. Their latency can be roughly two (Redis) or three (memcached) times higher than the same application running on the host CPUs. OpenLambda latency is dominated by the serverless framework overheads, which are orders of magnitude higher than memcached or Redis. Our results indicate a caveat of running applications in SNIC CPUs, which is to incur high latency even in an unloaded situation, and the degree of latency penalty can largely differ depending on the application.

We further extend our experiment with parallel requests. Figure 5a and Figure 5b plot throughput and average latency of memcached that runs in the SNIC machine, respectively. We issue 1000 `set` requests in parallel. The server uses a single host or SNIC CPU core, whereas the client uses all the eight CPU cores to avoid being saturated. Throughput with the Stingray SNIC CPU is close to that with the host CPU (Figure 5a), but with much higher latency (Figure 5b) particularly for small objects. This indicates that the Stingray core is almost saturated at those request rates (i.e., arriving requests are queued); data movement itself is not a bottleneck, as the degree of latency increase from the host CPU cases declines when the object sizes increase. BlueField2 SNIC CPU exhibits very low throughput with larger degrees of latency increase than Stingray cases. This further confirms the possibility of unpredictable high latency caused by "offloading" applications to SNICs, which would jeopardize the whole service, because memcached is typically used as a latency-critical component of a larger task (e.g., generating a response to the web client).

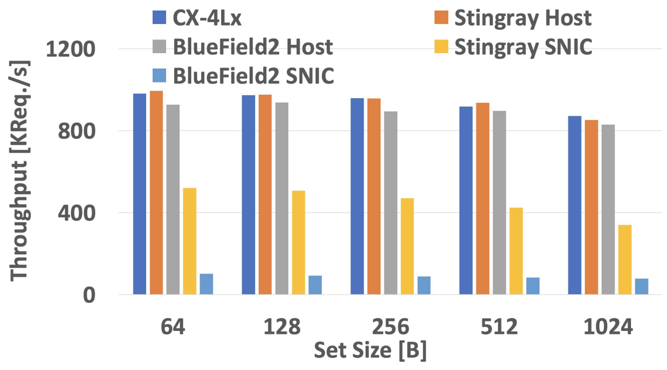
Figure 5c plots Redis throughput with 15 parallel TCP connections or requests. As in memcached, the BlueField2 SNIC CPU exhibits the lowest throughput by large margins. Also, even when the host CPU is used, throughput is relatively low due to its on-path SNIC architecture. On the other hand, Stingray SNIC highlights the weakness of its SNIC CPU, demonstrating around half of the throughput compared to the host CPU. Interestingly, this was not the case in memcached (Figure 5a). This observation further confirms that the impact of SNIC CPU could differ depending on the applications.



(a) Memcached throughput.



(b) Memcached latency.



(c) Redis throughput.

Figure 5: Redis and Memcached performance.

Finally, we measure the throughput and latency of an HTTP server executed in the OpenLambda serverless framework based on containers and plot the results in Figure 6. Since we have already observed high serverless function software overheads earlier in this section, we focus on multi-core scalability. In the experiment, the client (`wrk`) issues parallel HTTP requests or TCP connections, whose ratio is 100 × the number of server cores. The throughput increases and latency decreases when more server cores are used. As a notable observation, for throughput, ARM cores on SNIC exhibits significantly lower multi-core scalability compared to x86 cores on the host.

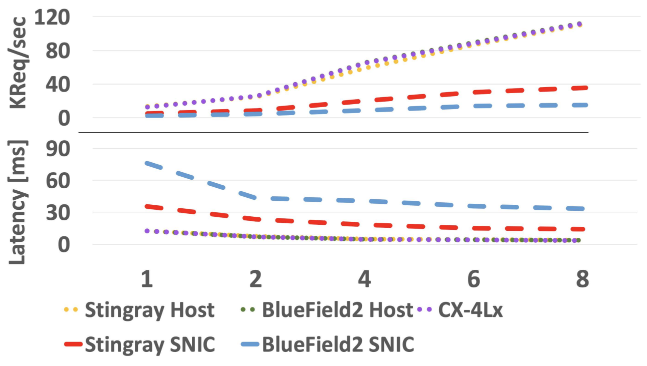


Figure 6: Throughput (top) and latency (bottom) of an HTTP server executed as an OpenLambda serverless function. X axis is the number of CPU cores.

6 CONCLUSION AND LESSONS LEARNED

This paper advocated the need for a measurement framework for Linux host systems equipped with CPU-based SNICs. Since we believe that many users wish to use a matured, feature-rich Linux network stack with regular socket APIs, we specifically focus on the performance on in-kernel network stack in SNICs. This is a stark contrast to existing work that characterizes the performance on SNICs with kernel-bypass network stacks [17]. We designed and implemented a framework using a custom network application and eBPF, and found the following SNIC performance characteristics:

- SNIC CPU performance can impact the end-to-end latency by a factor of two.
- Off-path CPUs do not always result in lower latency to and from the host CPUs.
- Even between similar applications, the degree of performance impact by SNICs can largely differ.
- Multicore-scalability of the application can be impaired when ARM SNIC CPUs are used.

ACKNOWLEDGMENTS

We are grateful to the anonymous APSys reviewers. Michio Honda was in part supported by EPSRC grant EP/V053418/1.

REFERENCES

[1] H. Agrawal, Y. Dutta, and S. Malik. 2012. Performance Analysis of Offloading IPsec Processing to Hardware Based Accelerators. In *2012 International Symposium on Electronic System Design (ISED)*. 291–294.
[2] Broadcom. 2018 (accessed June 30, 2020). *Stingray PS225*.
[3] Adrian Caulfield, Paolo Costa, and Manya Ghobadi. 2018. Beyond SmartNICs: Towards a Fully Programmable Cloud. In *IEEE International Conference on High Performance Switching and Routing*.
[4] Cavium. (accessed June 30, 2020). *LiquidIO-II 10/25G Smart NIC Family*.
[5] D. Chiou. 2017. The microsoft catapult project. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*. 124–124.
[6] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. 2019. λ-NIC: Interactive Serverless Compute on SmartNICs. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos* (Beijing, China) *(SIGCOMM Posters and Demos '19)*. Association for Computing Machinery, New York, NY, USA, 151–152. https://doi.org/10.1145/3342280.3342341
[7] Kevin Deierling. 2018 (accessed June 30, 2020). *Defining SmartNIC: What is a SmartNIC and How to Choose the Best One.*
[8] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking: SmartNICs in the public cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 51–66.

[9] Michael Galles and Francis Matus. 2021. Pensando Distributed Services Architecture. *IEEE Micro* 41, 2 (2021), 43–49.

[10] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2016. Serverless computation with openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*.

[11] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M. Swift, and T. V. Lakshman. 2017. UNO: Uniflying Host and Smart NIC Offload for Flexible Packet Processing. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 506–519. https://doi.org/10.1145/3127479.3132252

[12] Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R. K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and Anirudh Badam. 2020. LeapIO: Efficient and Portable Virtual NVMe Storage on ARM SoCs.

[13] Junnan Li, Zhigang Sun, Jinli Yan, Xiangrui Yang, Yue Jiang, and Wei Quan. 2019. DrawerPipe: A Reconfigurable Pipeline for Network Processing on FPGA-Based SmartNIC. *Electronics* 9, 1 (Dec 2019), 59. https://doi.org/10.3390/electronics9010059

[14] Junnan Li, Zhigang Sun, Jinli Yan, Xiangrui Yang, Yue Jiang, and Wei Quan. 2020. DrawerPipe: A reconfigurable pipeline for network processing on FPGA-based SmartNIC. *Electronics* 9, 1 (2020), 59.

[15] Huan Liu, Zhiliang Qiu, Weitao Pan, Jun Li, Ling Zheng, and Ya Gao. 2020. Low-Cost and Programmable CRC Implementation based on FPGA (Extended Version). (5 2020). https://doi.org/10.36227/techrxiv.12181494.v2

[16] Jianshen Liu, Carlos Maltzahn, Craig Ulmer, and Matthew Leon Curry. 2021. Performance Characteristics of the BlueField-2 SmartNIC. arXiv:2105.06619 [cs.NI]

[17] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading Distributed Applications onto SmartNICs Using IPipe. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 318–333. https://doi.org/10.1145/3341302.3342079

[18] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 363–378.

[19] Layong Larry Luo and TEG Tencent. 2018. Towards converged SmartNIC architecture for bare metal & public clouds.

[20] Mellanox. 2019 (accessed June 30, 2020). *BlueField SmartNIC for Ethernet High Performance Ethernet Network Adapter Cards*.

[21] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. 2018. Understanding PCIe Performance for End Host Networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 327–341. https://doi.org/10.1145/3230543.3230560

[22] Yiming Qiu, Qiao Kang, Ming Liu, and Ang Chen. 2020. Clara: Performance Clarity for SmartNIC Offloading. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. 16–22.

[23] Pravin Shinde, Antoine Kaufmann, Timothy Roscoe, and Stefan Kaestle. 2013. We Need to Talk About NICs. In *14th Workshop on Hot Topics in Operating Systems (HotOS XIV)*. USENIX Association, Santa Ana Pueblo, NM.

[24] Raffaele Sommese. 2018. *Boosting the performance of NFV services with SmartNIC.* Ph. D. Dissertation. Politecnico di Torino.

[25] Madhur Tj. 2017 (accessed June 30, 2020). *What's The Difference Between In PCI Express Gen 1 vs. Gen 2 vs. Gen 3 vs. Gen 4? | DESKDECODE.COM.*

[26] Yuta Tokusashi and Hiroki Matsutani. 2016. A Multilevel NOSQL Cache Design Combining In-NIC and In-Kernel Caches. In *2016 IEEE 24th Annual Symposium on High-Performance Interconnects (HOTI)*. 60–67. https://doi.org/10.1109/HOTI.2016.022

[27] Maroun Tork, Lina Maudlej, and Mark Silberstein. 2020. Lynx: A SmartNIC-Driven Accelerator-Centric Architecture for Network Servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 117–131. https://doi.org/10.1145/3373376.3378528

[28] Shuhe Wang, Zili Meng, Chen Sun, Minhu Wang, Mingwei Xu, Jun Bi, Tong Yang, Qun Huang, and Hongxin Hu. 2020. SmartChain: Enabling high-performance service chain partition between SmartNIC and CPU. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[29] X. Wang, Y. Niu, F. Liu, and Z. Xu. 2020. When FPGA Meets Cloud: A First Look at Performance. *IEEE Transactions on Cloud Computing* (2020), 1–1.

[30] Xilinx. 2019 (accessed June 30, 2020). *Zynq UltraScale+ Device Technical Reference Manual.*

[31] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. 2014. NetFPGA SUME: Toward 100 Gbps as Research Commodity. *IEEE Micro* 34, 5 (2014), 32–41. https://doi.org/10.1109/MM.2014.61