Making Sense of Using a SmartNIC to Reduce Datacenter Tax from SLO and TCO Perspectives

Jinghan Huang*, Jiaqi Lou*,Yan Sun*, Tianchen Wang*, Eun Kyung Lee†, Nam Sung Kim*
*University of Illinois at Urbana-Champaign, †IBM Research
{jinghan4, jiaqil6, yans3, tw12, nskim}@illinois.edu {eunkyung.lee}@us.ibm.com

2023 IEEE International Symposium on Workload Characterization (IISWC) | 979-8-3503-0317-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/IISWC59245.2023.00025

Abstract

The speed of network interfaces has rapidly increased, while the performance and energy efficiency of CPUs have not, due to the demise of Dennard scaling. As a result, functions processing network packets have become responsible for a rapidly increasing portion of the datacenter tax. To tackle this problem, the industry has developed SmartNICs (SNICs) integrating conventional NICs with inexpensive and energy-efficient processors that can efficiently execute functions widely used by network-intensive datacenter applications. With such processors, the SNICs promise to reduce the total cost of ownership (TCO) for datacenters by increasing energy efficiency of servers and/or decreasing the number of expensive server CPU cores. In this paper, to make sense of using SNICs, we focus on analyzing energy efficiency of a server with an SNIC, especially under service level objective (SLO) constraints which matter for many datacenter applications. To this end, we not only measure the system-wide power consumption of a server but also devise a custom hardware setup that allows us to isolate the power consumption of the SNIC from that of the server. This helps us better understand the impact of using SNICs on energy efficiency of servers. Second, we take popular TCP/UDP, DPDK, and RDMA-based functions, and prepare them to run on an SNIC processor and a server CPU. Then, we measure maximum throughput, tail latency, system-wide energy efficiency of executing the functions on the SNIC processor and the server CPU, respectively. Lastly, based on analyses of the measurements, we make five key observations and three strategies to better use and design SmartNICs in the future.

1. Introduction

The speed of datacenter networks has increased rapidly. Hyperscalers, such as Google, Amazon, and Microsoft, had widely used the 100 Gbps Ethernet, and they have started to deploy the 400 Gbps Ethernet in 2020 [54]. To keep up with the increasing speed of networks, a server needs to process hundreds of millions of packets per second using various functions. Meanwhile, the single-thread performance of server CPUs has remained stagnant since the demise of Dennard scaling. This demands the server to use more CPU cores and cycles solely for processing network packets, which imposes a high *datacenter tax* [36]. For example, a CPU core operating at 4 GHz needs to process a network packet with a 20-byte payload every 6.4 cycles to sustain a throughput of 100 Gbps. Also, the improvement in energy efficiency of CPUs has been stagnant, which is further exacerbated by the inherent inefficiencies of the CPUs due to von Neumann overheads. Consequently, processing network packets consumes more CPU resources and energy at higher network speeds, which gives rise to a higher total cost of ownership (TCO) for datacenters.

Promising to reduce the TCO, the industry has developed various SmartNICs (SNICs) that integrate conventional NICs with CPU, FPGA, and/or ASIC accelerators. For example, NVIDIA BlueField-2 [56] is an SNIC that integrates a standard NIC with a processor comprising three accelerators attached to eight energy-efficient Arm CPU cores through the (on-chip) PCIe interconnect. These CPU and accelerators aim to efficiently execute network packet processing functions commonly used by datacenter applications. This motivated certain hyperscalers to explore the use of SNICs, because offloading such functions from server CPUs to SNIC processors can reduce energy consumed by servers and/or the number of server CPU cores needed for executing the datacenter applications. However, prior research [11, 81] argues that it is challenging for PCIe-attached accelerators to efficiently execute latency-sensitive functions processing small microsecond-scale tasks, which are prevalent in modern datacenter applications [71], due to long latency of the PCIe interconnect. Besides, energy-efficient Arm CPU cores may not be powerful enough to process such tasks at high rates. In other words, it is still elusive whether SNICs can truly reduce the TCO while satisfying service level objective (SLO) constraints, which matters for many datacenter applications.

In this paper, to determine whether SNICs can be useful for reducing the TCO under SLO constraints, we take BlueField-2, a state-of-the-art CPU-based commodity SNIC on the market. Then, we comprehensively compare maximum throughput, tail latency, and system-wide energy efficiency achieved by the BlueField-2 processor with those attained by an Intel Xeon CPU. To this end, we run 13 TCP/UDP, DPDK, and RDMA-based functions on the SNIC processor and the Intel Xeon CPU that serves as the host CPU to the SNIC. Furthermore, to better understand overall energy efficiency of servers deploying SNICs, we exploit a state-of-the-art power measurement feature in modern servers and develop a custom power measurement setup. They allow us to accurately measure the system-wide power consumption of a server with an SNIC and isolate the power consumption of the SNIC from the system-wide power consumption of the server, respectively. This helps

2835-2238/23/$31.00 ©2023 IEEE
DOI 10.1109/IISWC59245.2023.00025

users determine what functions should be offloaded to SNIC processor based on different SLO targets. Our evaluations show that running the functions on the SNIC processor gives (1) 0.1∼3.5× maximum throughput, (2) 0.1∼13.8× 99^{th}-percentile (p99) latency, and (3) 0.2∼3.8× system-wide energy efficiency of running them on the host CPU. Based on our comprehensive evaluations and in-depth analyses, we make the following key **Observations**:

(O1) The SNIC CPU often provides lower maximum throughput and higher p99 latency than the host CPU, especially when it is required to run a complex networking stack, such as TCP/UDP. This is because the SNIC CPU is not capable enough to efficiently run both the functions and the networking stack at the same time.

(O2) The SNIC accelerators do not always provide higher maximum throughput and lower p99 latency than the host CPU, especially when the host CPU can utilize its ISA extensions for given functions. This is because the host CPU backed by a more powerful memory subsystem can more efficiently accelerate some functions with its acceleration hardware features and software libraries than the SNIC accelerators.

(O3) The SNIC accelerators cannot achieve the line rate (*e.g.*, 100 Gbps for BlueField-2). As such, the host CPU cores need to be reserved and ready to immediately process network packets that the SNIC processor cannot process at high network packet rates.

(O4) Even for the same function, depending on inputs/configurations, major operation types, and algorithms, the SNIC processor performs better or worse than the host CPU. That is, we cannot choose to use the SNIC processor simply based on given functions.

(O5) The SNIC processor does improve energy efficiency for some functions, but not always and not considerably, especially when we account for the system-wide energy consumption of a server. An SNIC is a PCIe-attached device that needs to be used as one of the subsystems of a server, while the server including the host CPU, DRAM, and other subsystem components consumes a considerable amount of (idle) power. As such, energy efficiency of a server is often dominated by throughput of the server regardless of whether the host CPU or the SNIC processor processes a given function.

Note that these observations may not be universal as they are based on our setup with specific SNIC, server configurations, and functions. Nonetheless, comparing the SNIC with other commodity SNICs (*e.g.*, [25, 47]), we expect to make similar conclusions with these SNICs because of the power and form-factor constraints that limit the capabilities of SNIC processors and their subsystems.

The remainder of this paper is organized as follows. Sec. 2 describes the SNIC architecture, accelerator subsystem, and operation modes. Sec. 3 presents the evaluation methodology. Sec. 4 shows evaluation results and analyses. Sec. 5 analyzes TCO and discusses strategies to better use and design SNICs in the future. Sec. 6 reviews related work. Finally, Sec. 7 concludes this paper.

2. Background

2.1. Overall SNIC Architecture

Recently, the industry has developed various SNICs to offload network packet processing functions common in datacenter applications from the host CPU, as part of an effort to reduce the datacenter tax. Generally, an SNIC integrates a standard NIC with a processor comprising inexpensive and energy-efficient CPU, FPGA, and/or ASIC accelerators. For example, NVIDIA BlueField-2 [56] is a CPU-based SNIC that integrates NVIDIA ConnectX-6 Dx [58], a standard 100 Gbps NIC, with eight Arm CPU cores; private L1 and shared L2 caches; cache coherent on-chip interconnect; DRAM and PCIe controllers; on-board DRAM as main memory; and three accelerators, *i.e.*, regular expression matching, compression/decompression, cryptography (Fig. 1). AMD SN1000 [78] integrates a CPU-based SNIC with FPGA in a single chip. Lastly, an SNIC is connected to the host CPU through the PCIe interconnect.

2.2. SNIC Accelerators

ConnectX-6 Dx in BlueField-2 already has accelerators for basic networking functions, such as generic routing, encapsulation tunneling, connection tracking, as well as an embedded switch ('eSwitch' in Fig. 1). In addition, BlueField-2 provides three **A**ccelerators: **(A1)** regular expression matching (REM), **(A2)** cryptography, and **(A3)** compression accelerators. It is advertised that BlueField-2 provides more accelerators such as SNAP (Software-defined Network Accelerated Processing) for storage virtualization. However, we do not evaluate them due to lack of publicly available documents describing how to use them.

(A1) REM is a widely deployed function for network security. For instance, some malware can hide malicious code (*i.e.*, some specific string patterns) in network packets [70]. REM checks whether ingress network packets contain such patterns based on programmed regular expression rules, and then it drops the packets containing matching patterns. Due to the inherent complexity of REM, it is essential to use a highly efficient accelerator for applications that are required to use an REM function at high packet rates. BlueField-2

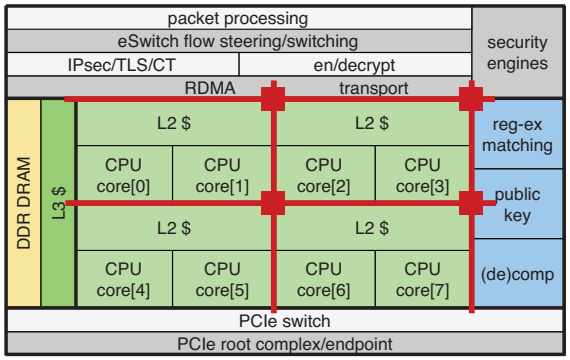


Figure 1: BlueField-2 architecture where the gray blocks represents the components of ConnectX-6 Dx.

provides an REM accelerator capable of handling multiple REM tasks through batching. Before a given application uses the REM accelerator, it programs a compiled rule set to the accelerator through DOCA (*i.e.*, BlueField-2 SDK) APIs [55], and then the BlueField-2 CPU is used to acquire ingress network packets from the network using DPDK, put the packets in buffers, and submit tasks with those buffers to the accelerator for processing the packets. For each task, the accelerator will return a list of network packets with matched patterns.

(A2) Cryptography is used by diverse applications, including `strongSwan` (open source IPsec-based VPN solution) [8] and `Nginx` (open source software for web serving, reverse proxying, caching, load balancing, media streaming, *etc.*) [19]. BlueField-2 provides an accelerator for 24 public key acceleration (PKA) algorithms, such as Rivest–Shamir–Adleman (RSA) [65], Advanced Encryption Standard (AES) [18], Digital Signature Algorithm (DSA) [51], Secure Hash Algorithm 2 (SHA-2) [52], and Elliptic-Curve Cryptography (ECC) [37]. To perform a cryptography function, the BlueField-2 CPU programs a specific memory region for a given PKA algorithm and (memory-mapped) command count register of the accelerator.

(A3) Compression is based on the Deflate algorithm [61] in BlueField-2. The compression accelerator can be used by both the BlueField-2 CPU and the host CPU. Similar to the REM accelerator, the BlueField-2 CPU first reads an uncompressed/compressed input file and writes data from the file to a buffer using DPDK APIs, and it submits the task with the buffer to the accelerator. Upon completion of the task, the accelerator will return the compressed/decompressed file to the buffer.

As described above, (A1) – (A3) rely on the BlueField-2 CPU to perform their respective functions. In contrast, the basic network function accelerators in ConnectX-6 Dx are bump-in-the-wire ones that do not involve the BlueField-2 CPU for performing their respective functions. Note that the host CPU can also access the REM and compression accelerators through the PCIe interconnect and offload the functions to the accelerators in a similar manner that it offloads functions to conventional PCIe-attached accelerators. This does not involve the BlueField-2 CPU, either. Instead, it is the host CPU that sets up DMA transfers to a specific MMIO address associated with a BlueField-2 accelerator.

2.3. Operation Modes

Based on how network packets flow within BlueField-2, we can operate it in one of the two Modes: **(M1)** on-path mode and **(M2)** off-path mode [57] (Fig. 2). Note that the operation mode does not change the physical connections between components within BlueField-2.

(M1) On-Path Mode. The BlueField-2 CPU controls the embedded switch such that all the ingress/egress network packets go through the BlueField-2 CPU and its subsystems first (Fig. 2(a)). Then, the BlueField-2 CPU serves as the control plane, configuring Open vSwitch (OvS) [62] to direct specific network packets to the BlueField-2 CPU or the host CPU. In addition, the BlueField-2 CPU can communicate with the host CPU and its subsystems through the PCIe interconnect.

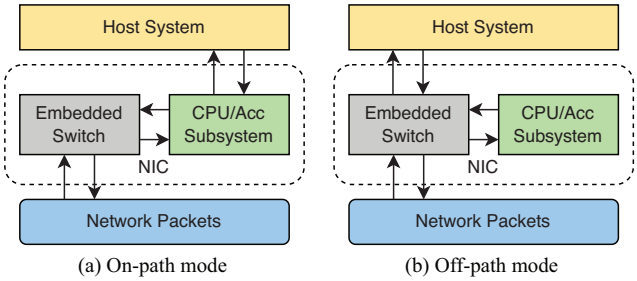


Figure 2: SNIC operation modes.

TABLE 1: Specifications of BlueField-2

	NVIDIA BlueField-2
CPU	8 ARMv8 A72 cores at 2.0 GHz
Accelerator	Regular Expression Matching (REM) Public Key Algorithm (Cryptography) Compression
Cache	256KB L1-D cache and 384KB L1-I cache per core 1MB L2 cache per 2 cores 6MB shared L3 cache
Memory	16GB on-board DDR4-3200
Network	Dual ports of 100 Gb/s (ConnectX-6 Dx)
PCIe	×16 Gen 4.0
OS	Ubuntu 20.04.5 LTS (Linux 5.4.0-1049-bluefield)

(M2) Off-Path Mode. The BlueField-2 CPU and its subsystems are exposed to other nodes including the host as an independent node connected to the Ethernet. Based on configured packet forwarding rules and destination Ethernet addresses, the embedded switch delivers network packets to either the BlueField-2 CPU or the host CPU directly.

In this paper, we evaluate the BlueField-2 processor only in the on-path mode, because the key accelerators including REM can be used only in the on-path mode, and NVIDIA discontinued the support for the off-path mode. A detailed specification of BlueField-2 used for our evaluation is summarized in Table 1.

3. Evaluation Setup and Methodology

3.1. System Setup

To perform our evaluation in this paper, we use two systems: one for a server and the other for a client. The configurations of these two systems are summarized in Table 2. The server is equipped with a 100 Gbps SNIC described in Table 1, while the client is equipped with a standard 100 Gbps NIC. In particular, for performance evaluation, when the host CPU executes functions, we use the Linux `userspace` governor [10] to set the operating frequency to 2.1 GHz (*i.e.*, the maximum operating frequency under

TABLE 2: System configurations.

	Client	Server
OS	Ubuntu 18.04.6 LTS (Linux 4.15.0)	Ubuntu 18.04.6 LTS (Linux 5.4.0)
Processor Model	Intel Xeon E5-2640 v3 (Broadwell)	Intel Xeon Gold 6140 (Skylake)
LLC	20 MB	24.75 MB
System Memory	32 GB DDR4-1866 2 DIMMs, 4 channels	128 GB DDR4-2666 8 DIMMs, 6 channels
NIC	ConnectX-6 Dx	BlueField-2
Driver Version	MLNX_OFED 5.8-1.0.1.1-LTS	

the TDP constraint), and disable Hyper Threading [31] and Turbo Boost [32] to reduce performance variations and interference. For energy evaluation, when the SNIC processor executes functions, we use the Linux `ondemand` governor to reduce the power consumption of the idle host CPU.

3.2. Power Measurement Setup

System Power Measurement. Advance Configuration and Power Interface (ACPI) is an industry standard that was developed to provide a unified interface for the operating system to monitor and manage power consumption of a server. We can access power sensors on the motherboard through the ACPI interface and Linux `lm-sensors` [5]. `IPMItool` [4] is a Linux software wrapper that offers a command line interface to manage the Intelligent Platform Management Interface (IPMI). Specifically, we use the Data Center Manageability Interface (DCMI), which is a component of `IPMItool`, to measure power consumption of the server using the primary power sensor integrated on the motherboard and controlled by the Baseboard Management Controller (BMC) in the server. It can measure power consumption of a server at 1Hz with $\pm$ 1W accuracy, but it can only measure the system-wide power consumption of the server including all the PCIe-attached devices. In other words, it cannot isolate the power consumption of a NIC from the system-wide power consumption of the server.

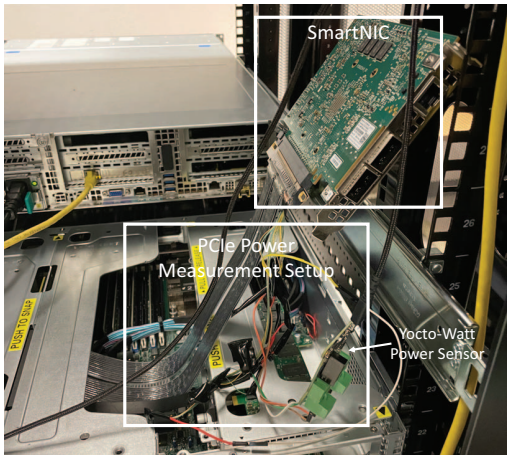


Figure 3: Evaluation system.

Note that our sampling rate for power measurement is sufficiently high for accurate measurements due to the following reasons. The motherboard provides larger capacitance than a single CPU [21]. This gives the time constant for power changes (or fluctuations) comparable to our sampling period. The hyperscalers also employ the same power measurement mechanism.

SmartNIC Power Measurement. To precisely measure power consumption of the (S)NIC, we develop a custom power measurement setup (Fig. 3). This setup allows us to measure power consumption of a specific PCIe-attached device. It consists of (1) a PCIe riser card; (2) two Yocto-Watt [80] power sensors; and (3) a program that runs on another server to configure the power sensors, collect power traces, and store the traces. The two Yocto-Watt power sensors tap 12V and 3.3V power pins of a PCIe interface, respectively, and then measure the power consumption of a PCIe-attached device at 10Hz with $\pm$ 2mW accuracy. To validate the accuracy of our power measurement setup, we measure the system-wide power consumption of the server with and without the SNIC. We observe that the difference between the two power consumption values of the server with and without the SNIC is approximately matched with the power consumption value of the SNIC measured by our power measurement setup. Besides, the sampling rate and resolution of our power measurement setup for the SNIC is 10$\times$ and 500$\times$ higher than those of BMC for the server, respectively.

3.3. Microbenchmark

We take three microbenchmarks to evaluate the performance of networking stacks: UDP, DPDK, and RDMA that send/receive 64 B and 1 KB packets. These packet sizes are close to two representative packet sizes in datacenters [13].

UDP Microbenchmark. UDP is a connectionless protocol, and it is primarily used to establish low-latency and loss-tolerating connections between nodes since it does not guarantee packet delivery to the destination node. To measure throughput and per-packet round-trip latency, we run a UDP-based client-and-server program on eight host or SNIC CPU cores.

DPDK Microbenchmark. DPDK is widely used by datacenter networks for fast packet processing, with the support of user-space data plane libraries and polling-mode drivers. We run a simple ping-pong-style program [34] and `DPDK-Pktgen` [77] on one host or SNIC CPU core to measure per-packet round-trip latency and throughput, respectively. Since DPDK offers high packet processing efficiency, one host or SNIC CPU core can accomplish the 100 Gbps line rate for 1 KB packets.

RDMA Microbenchmark. RDMA offloads the networking stack to NIC hardware and allows clients to directly access the memory of servers, and it is now supported by most high-end NICs. We run `perftest` [44] to measure throughput and per-packet round-trip latency of both one-sided

TABLE 3: Benchmark. HC, SC, and SA denote host CPU, SNIC CPU, and SNIC accelerator.

Benchmark	Network Stack	Description & Configuration	Execution Platform		
			HC	SC	SA
Redis	TCP/UDP	In-memory key-value store with three workloads (A, B, C)	✓	✓	
Snort		Intrusion detection and prevention tool with three rulesets (img, fla, exe)	✓	✓	
NAT		Network address translation with ($10K, 1M$) entries	✓	✓	
BM25		Search engine ranking function with ($100, 1K$) documents	✓	✓	
Cryptography		Public key algorithm (AES, RSA, SHA)	✓	✓	✓
REM	DPDK	Regular expresion matching with three rulesets (img, fla, exe)	✓	✓	✓
Compression		Deflate compression algorithm with two test files (app, txt)	✓	✓	✓
OvS		Software-based switch with ($10\%, 100\%$) traffic load	✓	✓	✓
MICA	RDMA	Kernel-bypass key-value store with $(4, 32)$ batch size	✓	✓	
fio		Remote storage access through NVMe-oF with (Read, Write)	✓	✓	

(read/write) and two-sided (send/recv) RDMA operations. perftest is a test suite developed for evaluating latency and bandwidth of Infiniband and RoCEv2 networks. To exclude the potential influence of lost packets on throughput and latency [49], we use the default Reliable Connection (RC) transport mode. Similar to DPDK, we use only one host or SNIC CPU core.

3.4. Benchmark

We take ten benchmarks widely used by datacenter applications and run them on the following execution platforms: host CPU, SNIC CPU, and/or SNIC accelerator. Table 3 lists the benchmark names, used networking stacks, and execution platforms along with brief descriptions of the benchmarks. These benchmarks have been also used for evaluating SNICs in prior work [22, 46, 82, 84].

For each benchmark, we choose a networking stack commonly used for the benchmark. To compile the benchmark source code, we follow the standard compilation flow for Arm and Intel CPUs with the same optimization option. We run these benchmarks with popular data sets, configurations (*e.g.*, batch sizes), and operation types (*e.g.*, read or write) that are used by the prior work. As the SNIC CPU has 8 cores, we also use 8 host CPU cores in our evaluations unless specified otherwise. We ensure that the size of data sets fits into the main memory of the SNIC. This prevents the benchmarks from causing page faults, and thus allows the SNIC to accomplish its maximum performance for the benchmarks. The SNIC has a smaller cache capacity than the host CPU, but the benchmarks do not exhibit notable performance sensitivity to cache capacity since they serve either streaming or random memory accesses.

Note that three benchmarks, REM, Cryptography, and Compression in Table 3 are also supported by the BlueField-2 accelerators, and they are advertised as the functions widely used by datacenter applications. For these three benchmarks, we follow the instructions described in DOCA (*i.e.*, BlueField-2 SDK) [55], and then we sweep accelerator-specific parameters to get maximum performance.

TCP/UDP-based Benchmark. Redis [64] is a high-performance in-memory key-value store (KVS) widely used for real-time data store, caching and data streaming. We run a TCP-based Redis with YCSB [15]; we use workload_a (50% read and 50% write), workload_b (95% read and 5% write), and workload_c (100% read) for workloads, load the database with 30K records (1 KB per record), and perform 10 K operations.

Snort [67] is a popular programmable packet sniffing and logging tool based on libpcap. It is widely used as a lightweight network intrusion detection or prevention system. It can check packets to detect malicious network activities. It runs on the server processing UDP packets sent by iperf [17] running on the client. We use file_image, file_flash, and file_executable from a registered rule set (*i.e.*, snapshot 31470) [7].

NAT [53] denotes network address translation. NAT running on a router changes the destination IP address of ingress packets to forward them to a private network, and modifies the source IP address of egress packets to protect the connection to the private network. We set up NAT in a UDP-based server. For each given IP address, it finds a matched IP address in NAT entries. We run NAT for 10 K and 1 M entries, the content of which is randomly generated.

BM25 [66] is one of the most successful ranking algorithms for search engines. It can be used to compute relevance between input queries and database documents in distributed systems. We set up BM25 and run it on a UDP-based server with 100 and 1 K database documents, each with 10 words on average. The content of these documents is randomly generated. One query operation using BM25 is performed upon arrival of each packet.

Cryptography is commonly used by TCP/UDP-based applications, such as OpenSSL [6]. To measure the performance of diverse cryptography algorithms, we run AES, RSA, and SHA-1 that are used by OpenSSL locally on the server without processing TCP/UDP packets from the client. In such a case, we find that a single SNIC CPU core is sufficient to send commands to the cryptography

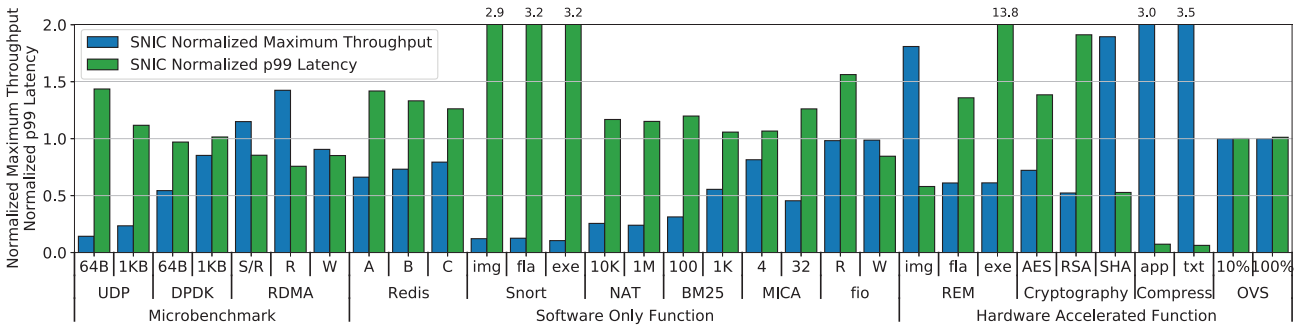


Figure 4: Maximum sustainable throughput and p99 latency of running functions on the SNIC processor, normalized to those of running the functions on the host CPU. For RDMA, we show the values only for processing 1 KB packets since the trends for processing 64 B and 1 KB packets are similar.

accelerator for its maximum throughput. We leverage the Intel RDRAND technology [24], a hardware-based random number generator to run these algorithms on the host CPU.

DPDK-based Benchmark. `REM` running on the host CPU or the SNIC accelerator processes packets sent by the client. `DPDK-Pktgen` running on the client sends PCAP traces (`CTU-Mixed-Capture-5` from Stratosphere [72]) comprising various sizes of packets to the server at ∼70 Gbps speed per client CPU core. The host CPU runs `Hyperscan` [74] as high-performance `REM`. Both the host CPU and the SNIC accelerator use the same three rule sets as `Snort`. To execute `REM` on the SNIC accelerator at its maximum sustainable throughput, we use two SNIC CPU cores for processing DPDK packets and supplying the packets to the SNIC accelerator.

`Compression` is evaluated with `Application3` and `Text1` from a compression benchmark suite [2] as input files and `dpdk-test-compress-perf` to measure the performance of the Deflate algorithm. The compression level is set to 9 to get the best compression ratio. For the host CPU, we also use the Intel Intelligent Storage Acceleration Library (ISA-L) [26], which leverages the Intel AVX technology [28] to accelerate compression. To evaluate performance of the `Deflate` algorithm with the host CPU, we use `TurboBench` [9]. We use two SNIC CPU cores to stage the input files and submit the tasks to the accelerator at its maximum throughput.

`OvS` is a virtual switch within a hypervisor on a server. As such, it plays an important role in datacenters because it is the foundation for building software-defined network (SDN) and network virtualization for multi-tenancy on top of physical networks. Although we can run both the data and control planes of `OvS` on a CPU, we offload the data plane of `OvS` to the embedded switch provided by both ConnextX-6 and the BlueField-2. In such a case, the host CPU and the SNIC CPU run only the control plane of `OvS`. We evaluate `OvS` in a similar manner as `REM` using `DPDK-Pktgen` at 10% and 100% of the line rate, but the packet size is fixed to the MTU (*i.e.*, 1500 B).

RDMA-based Benchmark. `MICA` [42] is a scalable, high-throughput KVS that uses a partitioned design and exploits the high thread-level parallelism of multi-core CPUs. Different from `Redis`, `MICA` leverages RDMA and enables application software to directly interface with NICs, bypassing the kernel and scheduling key-value requests to the most efficient CPU core. We run it with a 100% `GET` workload and batch sizes of 4 and 32.

`fio` [3] is a tool to evaluate performance of storage systems. In our setup, it runs on the server acting as a compute server and accesses a storage device in a remote storage server through NVMe over Fabrics (NVMe-oF) which is a protocol designed for accessing remote flash array storage over network fabrics, such as RDMA, FC, or TCP/IP. The server sends 64 KB block I/O requests to the storage server, running `fio` as an NVMe over RDMA workload with `iodepth` of 4. In our evaluation, the storage server runs `RAMDisk` to emulate a fast 16 GB block device and uses the NVMe-oF offloading engine in both ConnextX-6 NIC and BlueField-2.

4. Evaluation

Fig. 4 plots the maximum (sustainable) throughput and 99^{th}-percentile (p99) latency values of the SNIC processor running microbenchmarks and benchmarks (*i.e.*, functions), normalized to those of the host CPU running them. We classify functions, which can be accelerated by the SNIC accelerators, into the 'Hardware Accelerated Function' category and the remaining ones into the 'Software Only Function' category. We set the packet rate at which we get the maximum throughput for an evaluated function and a given packet size, and then measure the p99 latency at that rate. This measurement shows that the SNIC processor gives 0.1×–3.5× maximum throughput and 0.1∼13.8× p99 latency of what the host CPU can provide.

Key Observation 1. The SNIC CPU often provides lower maximum throughput and higher p99 latency than the host CPU, especially when it is required to run a complex networking stack, such as TCP/UDP.

Fig. 4 shows that the SNIC CPU offers 20.6%–89.5% lower maximum throughput and 1.1×–3.2× higher p99 latency

than the host CPU for TCP/UDP-based functions (*i.e.*, `Redis`, `Snort`, `NAT`, `BM25`). This is because the SNIC CPU is not capable enough to efficiently run both the functions and the networking stack at the same time. In contrast, the SNIC CPU delivers maximum throughput values similar to what the host CPU can provide for simple RDMA-based functions, such as `fio`. This is because the SNIC CPU offloads the RDMA stack to NIC hardware. Still, the SNIC CPU gives 19.5%–54.5% lower maximum throughput and 6.7%–26.2% higher p99 latency than the host CPU for more complex RDMA-based functions, such as `MICA`.

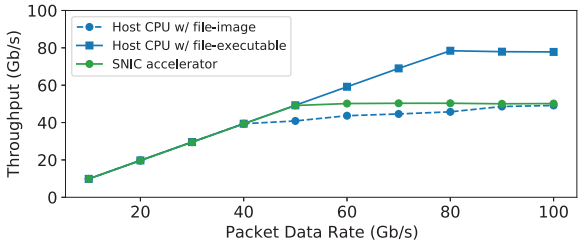
This observation is supported by our evaluation of the networking stack performance using the microbenchmarks. Specifically, the SNIC CPU provides 76.5%–85.7% lower maximum throughput and 1.1×–1.4× higher p99 latency than the host CPU for UDP. By contrast, the SNIC CPU accomplishes up to 1.4× higher maximum throughput than the host CPU and 14.6%–24.3% lower p99 latency than the host CPU for RDMA; the host CPU also exploits RDMA hardware in the (S)NIC, but it goes through a longer communication path to the hardware than the SNIC CPU [76]. Therefore, RDMA-based functions are more suitable to be offloaded to the SNIC CPU than TCP/UDP-based functions. Note that Fig. 4 shows the maximum throughput and p99 latency of processing only 1 KB packets for the RDMA microbenchmark, because they are similar to those of processing 64 B packets.

Key Observation 2. The SNIC accelerators do not always provide higher maximum throughput and lower p99 latency than the host CPU, especially when the host CPU can utilize its ISA extensions for given functions.

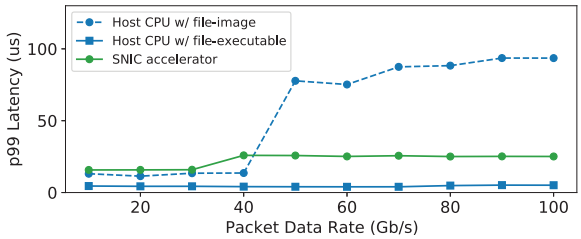
The SNIC accelerators provide up to 3.5× and 1.8× higher maximum throughput than the host CPU for `Compression` and `REM` (for specific input rule sets), respectively. However, the host CPU can offer 38.5% and 91.2% higher maximum throughput than the SNIC accelerator for `AES` and `RSA`, respectively, because it can efficiently accelerate them with the ISA extensions facilitated by Intel RDRAND technology. Although `SHA-1` is one of the cryptography algorithms, the host CPU offers 47.2% lower maximum throughput the SNIC accelerator. This is because the Intel RDRAND technology does not efficiently support `SHA-1`.

Note that the recent Intel Sapphire Rapids Xeon processor [60] is integrated with diverse accelerators, such as QAT [30], DSA [35], and IAA [29], for many functions including compression and cryptography. Since a server based on the Intel Sapphire Rapids Xeon processor is not available to us, we do not evaluate these accelerators in this paper. Yet, we expect these accelerators can provide higher performance than the SNIC accelerators as they are backed by a more powerful memory subsystem of the host CPU.

Delving deeper into analyzing performance of the SNIC accelerators, we take `REM` as a representative function accelerated by an SNIC accelerator. Fig. 5 plots throughput and p99 latency values of the host CPU and the SNIC accelerator running `REM` for two input rule sets (`file_image` and `file_executable`), as we sweep the rate of packets. We do not plot the throughput and p99 latency values for `file_flash` because the SNIC accelerator and the host CPU provide similar throughput and p99 latency values for both `file_flash` and `file_executable`. Besides, as the SNIC accelerator offers almost the same throughput and p99 latency for the two input rule sets, we show the throughput and p99 latency values of the SNIC accelerator for only one input rule set. Lastly, the Fig. 4 values are based on using PCAP traces comprising various sizes of packets (Sec. 3.4). Meanwhile, the Fig. 5 values are based on a packet size equal to the MTU to evaluate the throughput and p99 latency of the SNIC accelerator with the minimum overhead for processing packets.



(a) Throughput



(b) p99 latency

Figure 5: Throughput and p99 latency of the host CPU and the SNIC accelerator running `REM` versus packet data rates.

Key Observation 3. The SNIC accelerators cannot achieve the line rate (*e.g.*, 100 Gbps for BlueField-2).

Fig. 5 shows that the maximum throughput of the SNIC accelerator processing `REM` is capped to ∼50 Gbps (regardless of the input rule set). In contrast, that of the host CPU processing `REM` with `file_image` scales up to 78 Gbps with 8 cores (and handle up to 100 Gbps with 10 cores). The SNIC accelerator provides a few times higher throughput than the host CPU for `Compression`, but it can provide only a maximum throughput of ∼50 Gbps. In such a case, host CPU cores need to be reserved and ready to immediately process packets when the SNIC accelerator cannot process at high network packet rates. Nonetheless, the current SNIC does not support any efficient mechanism that can determine whether the SNIC accelerator is processing all the ingress packets or not and redirect all or some of the packets to the host CPU without notably increasing p99 latency. Furthermore, even with such a mechanism, it is

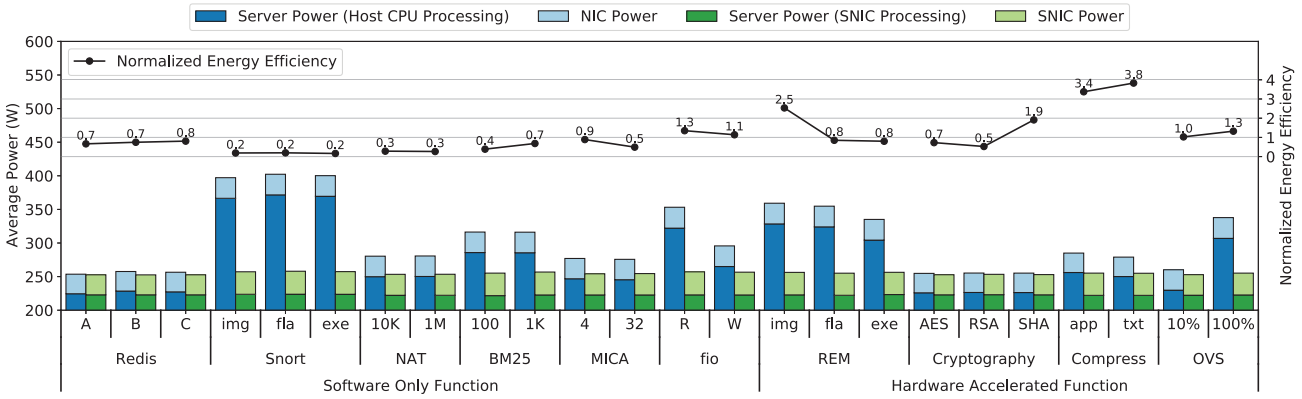


Figure 6: Average power consumption and normalized energy efficiency. The energy efficiency is throughput divided by the system-wide energy consumption. We normalize the energy efficiency of the SNIC processor processing functions to the host CPU processing the functions.

hard to put these reserved host CPU cores into sleep mode to reduce the power consumption of the host CPU because of the impact of waking them up on p99 latency [45, 79].

Key Observation 4. Even for the same function, depending on inputs, configurations, algorithms, and major operation types, the SNIC processor performs better or worse than the host CPU.

Fig. 4 shows that the SNIC accelerator can achieve 1.8× higher maximum throughput than the host CPU for `REM` with `file_image`. However, it can accomplish only 0.6× throughput of the host CPU for `REM` with `file_flash` and `file_executable`. Fig. 5 also shows that for `file_executable`, the host CPU provides up to a throughput of 78 Gbps with 8 cores while the SNIC accelerator delivers up to a throughput of 50 Gbps (Key Observation 3). Besides, the host CPU offers 5.1μs for p99 latency at the maximum throughput point, whereas the SNIC accelerator provides 25.1μs. That is, the host CPU can provide 1.6× higher throughput and 1.6×–5× lower latency for `REM` with this particular input rule set. However, for `file_image`, the host CPU gives only a maximum throughput of ∼40 Gbps when a reasonable p99 latency value is considered. The p99 latency increases dramatically after the packet rate exceeds ∼40 Gbps, which is depicted by dotted segments of the lines in Fig. 5. In contrast, the SNIC accelerator delivers almost the same p99 latency for both the input rule sets. Up to a packet rate of 40 Gbps, the host CPU offers slightly lower p99 latency than the SNIC accelerator.

For `Cryptography`, the host CPU executes the `AES` and `RSA` algorithms more efficiently than the SNIC accelerator although it processes `SHA-1` less efficiently (Key Observation 2). Both the host and SNIC CPUs give almost the same maximum throughput for `fio`. However, the host CPU provides 36.0% lower p99 latency than the SNIC CPU for `fio`, when performing `read` operations, whereas it offers 18.2% higher p99 latency when executing `write` operations.

For `BM25` and `MICA`, the SNIC CPU currently provides worse performance than the host CPU. Nonetheless, the performance of the SNIC CPU relative to the host CPU notably varies depending on the input and batch sizes of `BM25` and `MICA`, respectively. If the SNIC CPU becomes more powerful in the future, it may outperform the host CPU for certain input and batch sizes for such functions. That is, we cannot choose to use the SNIC processor solely based on given functions.

Fig. 6 plots average power consumption values of the host CPU and the SNIC processor processing functions, breakdown of the server power consumption between the (S)NIC and the rest of the server, and energy efficiency (*i.e.*, throughput divided by energy consumption) of the host CPU and the SNIC processor processing the functions. We measure the power consumption when a given function runs at the maximum throughput points shown in Fig. 4. When the server is idle, the power consumption values of the server and the SNIC are 252 Watts and 29 Watts, respectively. Active power consumption is the power consumption of the server running a function minus the idle power consumption of the server. For the active power consumption, the server and the SNIC can consume up to 150.6 Watts and 5.4 Watts, respectively. The SNIC processor gives 0.2×–3.8× energy efficiency of the host CPU.

Key Observation 5. The SNIC processor does improve energy efficiency for some functions, but not always and not considerably, especially when we account for the system-wide energy consumption of a server.

The SNIC processor does offer 1.1–1.3×, 2.5×, 1.9×, and 3.4–3.8×, higher energy efficiency for `fio`, `REM` (only with `file_image`), `SHA-1`, and `Compression`, respectively, than the host CPU. Except for `fio`, it gives higher energy efficiency only when it accomplishes higher throughput with its accelerators. Nonetheless, the SNIC processor does not improve the energy efficiency notably except for `Compress` because of the following reason. An SNIC is

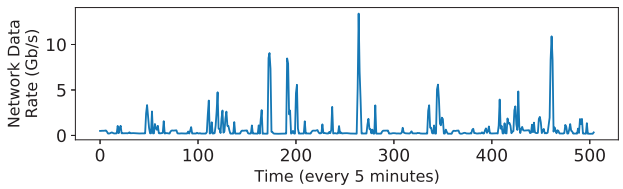


Figure 7: Network data rates over time.

a PCIe-attached device that needs to be used as one of the subsystems of a server. As such, although the SNIC processor processes a given function, the server including the host CPU, DRAM, and other subsystem components consumes a considerable amount of (idle) power.

5. Discussion

5.1. Analysis with a Datacenter Network Trace

To assess the energy efficiency benefit of deploying SNICs to datacenter servers, we take a network trace from a hyperscaler (Fig. 7) and measure average throughput, p99 latency, and average power consumption of running `REM` on both the host CPU and the SNIC accelerator. For this measurement, we modify `DPDK-Pktgen` to send packets, following the packet rate distribution of the network trace, and use the MTU packet size and the `file_executable` rule set. The average throughput, p99 latency, and average power are shown in Table 4. The average packet and data rates of this trace are relatively low and similar to values reported by prior work [13, 83]. As such, both the host CPU and the SNIC accelerator show the same average throughput, but the SNIC accelerator gives ∼3× longer p99 latency than the host CPU. If a given application using the `REM` function has to meet a certain SLO (*e.g.*, p99 latency) constraint based on the performance of the host CPU running this application, the SNIC is not suitable for the application due to notably higher p99 latency. Even if we relax the latency constraint, the overall power reduction is only 9%.

5.2. TCO Analysis

Table 5 compares the 5-year TCO of two servers with a comparable standard NIC and an SNIC, including the costs of the servers and and consumed electricity. We estimate that the costs of a server without any NIC, an SNIC (BlueField-2: MBF2M516A-CEEOT), and a NIC (ConnectX-6 DX: MCX623106AC-CDAT) are $6,287, $1,817 and $1,478, respectively, based on the market price of the Intel Xeon CPU [27] and other components such as DDR4-2666 DIMMs and motherboard [1]. That is, the servers with the SNIC and NIC cost $8,098 and $7,759, respectively. We make the same assumption as recent work [12, 48, 59] to estimate the TCO for the server lifetime (*i.e.*, 5 years) and the power cost (*i.e.*, $0.162 per kWh). We choose `REM`, `fio`, `OVS`, and `Compress` for this analysis, because (1) accelerating `REM` is one of the most advertised SNIC use cases; (2) the host CPU and SNIC CPU provide similar throughput for `fio`; (3) the host CPU and SNIC accelerator provide similar throughput for `OVS`; and (4) the SNIC accelerator gives far higher throughput than the host CPU for `Compress` in our evaluations. We see that using the SNIC increases the TCO by 2.5% for `REM` while decreasing the TCO by 2.7%, 1.7%, and 70.7% for `fio`, `OVS`, and `Compress`, respectively. Note that this TCO analysis is just for our setup. Since hyperscalers often make their own servers and have more purchasing power for components, they may make different conclusions on the TCO benefit of using SNICs.

TABLE 4: `REM` trace from a major hyperscaler.

	Host Processing	SNIC Processing
Throughput (Gb/s)	0.76	0.76
p99 Latency (μs)	5.07	17.43
Average Power (W)	278.30	254.50

5.3. Strategy for Better Utilizing SNIC

Strategy 1. The SNIC needs better hardware support for offloading the networking stack from the SNIC CPU to dedicated SNIC hardware.

Key Observation 1 states that execution of the TCP/UDP stacks alone consumes a large percentage of SNIC CPU cycles and limits performance of the functions. To fully exploit the performance and energy efficiency benefits of using the SNIC processor for offloading TCP/UDP-based functions, we suggest that SNIC should provide more hardware support for the TCP/UDP networking stack. While fully offloading the TCP/UDP networking stack to hardware is challenging, prior work proposes to partially offload the TCP stack to a NIC. For example, `FlexTOE` [69] decomposes the TCP datapath into fine-grained modules and offloads them to an SNIC, while `AccelTCP` [50] offloads the TCP connection setup and tear-down entirely to the NIC.

Strategy 2. The SNIC needs more intelligent policies to determine functions to offload to the SNIC processor.

Key Observations 2 and 4 indicate that we must consider inputs, configurations, algorithms, and major operation types to determine whether offloading a given function to a SNIC processor can be beneficial. To this end, we may use an automation tool such as `Clara` [63] that can predict performance of a given function running on an SNIC for various configurations and operation types in advance.

Strategy 3. The SNIC needs more efficient mechanisms that can balance the load between the SNIC processor and the host CPU.

Key Observations 2, 3 and 4 demonstrate that the host and SNIC have their respective advantages on different functions, and even the SNIC accelerators cannot deliver the line rate throughput and/or meet a given SLO. Monitoring the performance statistics of an SNIC processor running functions, such as throughput and latency, a load balancer can distribute network packets between the host CPU and

TABLE 5: TCO analysis for fio, OVS, REM, and Compress.

Application	fio		OVS		REM		Compress	
	SNIC	NIC	SNIC	NIC	SNIC	NIC	SNIC	NIC
Servers needed	10	10	10	10	10	10	10	35
Power per server (W)	257	343	255	328	255	268	255	269
Power use per server (kWh)	11260	15023	11178	14349	11147	11743	11169	11773
Power cost per server	$1824	$2434	$1811	$2325	$1806	$1902	$1809	$1907
5-year TCO	$99223	$101928	$99088	$100835	$99038	$96613	$99074	$338320
TCO savings using SNIC	2.7%		1.7%		-2.5%		70.7%	

the SNIC processor. When the SNIC processor processes a given function at low packet rates, it will offer higher energy efficiency than the host CPU. If the SNIC processor needs to process the function at high packet rates, the load balancer may quickly redirect all or some of the network packets to the host CPU to prevent potential SLO violations. However, our preliminary investigation reveals that a load balancer implemented with current mechanisms in BlueField-2 consumes most of the SNIC CPU cycles simply to monitor packets at high rates and it cannot redirect packets fast enough to meet SLO constraints. Therefore, we need better (hardware-based) mechanisms for an efficient and fast load balancer in future SNICs.

6. Related Work

SNIC Use Case Exploration. Over the last few years, many studies have explored the use of SNICs for various purposes. Some studies proposed to strategically offload certain CPU-intensive functions to CPU- and FPGA-based SNICs and demonstrated that SNICs could reduce consumption of host CPU resources, accelerate certain specialized operations, and improve energy efficiency [14, 16, 20, 22, 33, 38, 43, 68, 73, 75]. For instance, IO-TCP offloads certain TCP operations (*e.g.*, storage I/O and packet transfer) to SNICs for online content delivery applications [40]. LineFS offloads certain CPU-intensive tasks, such as replication and compression, required by distributed file systems to SNICs [39]. AlNiCo employs SNICs to schedule incoming requests in transaction processing [41]. Each of them explores one specific use case of SNICs and focuses on evaluating throughput and latency without analyzing the system-wide energy efficiency of servers deploying SNICs.

SNIC Performance and Energy Efficiency Analysis. Other studies analyze performance and energy efficiency of various SNICs. E3 [46] measured various runtime metrics, such as average/tail latency, throughput, and energy consumption of an SNIC running multiple microservices. However, they used a 10 Gbps SNIC (Cavium LiquidIO), which is a low-end one compared with BlueField-2 we evaluate in this paper. Liu *et al.* measured various performance characteristics of BlueField-2 but from a very different perspective. They focused on comparing only the throughput of the SNIC CPU with that of the host CPU, and evaluated only microbenchmarks (stress-ng) that do not run with a networking stack [45]. Xing *et al.* took a 25 Gbps SNIC (Broadcom Stringray) and a 100 Gbps SNIC (NVIDIA BlueField-2), and measured throughput and average latency of the SNIC CPUs running end-to-end applications, such as Memcached, Redis, and OpenLambda [79]. Wei *et al.* measured only throughput and average latency of the SNIC CPU running RDMA applications in the off-path mode [76]. In contrast, building on a preliminary study [23], our study holistically compares the throughput, p99 latency, and system-wide energy efficiency of both the SNIC CPU and the SNIC accelerators running diverse functions with those of a server CPU.

7. Conclusion

As the speed of network interfaces has rapidly increased, functions processing network packets have become responsible for a rapidly increasing portion of the datacenter tax. To tackle this problem, the industry has developed SNICs integrating conventional NICs with processors that can efficiently execute such functions and promised to reduce the TCO for datacenters. In this paper, we have analyzed energy efficiency of a server with an SNIC, especially under p99 latency constraints after evaluating popular TCP/UDP, DPDK, and RDMA-based functions using a state-of-the-art SNIC processor and the Intel Xeon CPU. Based on analyses of the measurements, we have made the following key observations. The SNIC accelerators do not always provide higher maximum throughput and lower p99 latency than the host CPU and they do not achieve the line rate. Besides, the SNIC processor improves energy efficiency for some functions, but not always and not considerably as the system-wide energy efficiency of servers is dominated by the throughput of processing the functions. Based on these key observations, we have suggested that future SNICs should provide mechanisms for balancing the load between the host CPU and the SNIC processor.

Acknowledgment

This work was supported in part by the IBM-Illinois Discovery Accelerator Institute and PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Nam Sung Kim has a financial interest in Samsung Electronics and NeuroRealityVision.

References

[1] "CDW," https://www.cdw.com/.

[2] "Compressionratings.com corpus," https://compressionratings.com/download.html.

[3] "fio - Flexible I/O tester," https://fio.readthedocs.io/en/latest/fio_doc.html.

[4] "ipmitool," https://linux.die.net/man/1/ipmitool.

[5] "lm-sensors," https://github.com/lm-sensors/lm-sensors.

[6] "OpenSSL," https://www.openssl.org/.

[7] "Snort Rules," https://www.snort.org/downloads/#rule-downloads.

[8] "strongSwan: Open-source, modular and portable IPsec-based VPN solution," https://www.strongswan.org/.

[9] "TurboBench," https://github.com/powturbo/TurboBench.

[10] Arch Linux, "CPU frequency scaling," https://wiki.archlinux.org/title/CPU_frequency_scaling.

[11] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the Killer Microseconds," *Communications of the ACM*, 2017.

[12] S. Belanger and A. Nadkarni, "Server Upgrade Cycles: Why Faster Is Better," https://www.ibm.com/downloads/cas/AV1ZWNZM.

[13] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," *SIGCOMM Comput. Commun. Rev.*, 2010.

[14] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A Cloud-Scale Acceleration Architecture," in *Proceedings of the 49th IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*, 2016.

[15] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*, 2010.

[16] D. Du, Q. Liu, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Serverless Computing on Heterogeneous Computers," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, 2022.

[17] J. Dugan, J. Estabrook, J. Ferbuson, A. Gallatin, M. Gates, K. Gibbs, S. Hemminger, N. Jones, F. Qin, G. Renker, A. Tirumala, and A. Warshavsky, "iPerf - The ultimate speed test tool for TCP, UDP and SCTP," https://iperf.fr/.

[18] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, "Advanced Encryption Standard (AES)," *Federal Inf. Process. Stds. (NIST FIPS)*, 2001.

[19] F5 NGINX, "Connect, Scale, and Secure Apps and APIs with F5 NGINX Management Suite," https://www.nginx.com/blog/connect-scale-secure-apps-apis-with-f5-nginx-management-suite/.

[20] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, L. Jack, L. Norman, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, M. Silva, Ganriel nd Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, 2018.

[21] C. Gough, I. Steiner, and W. Saunders, *Energy Efficient Servers: Blueprints for Data Center Optimization*. Apress, 2015.

[22] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'20)*, 2020.

[23] J. Huang, J. Lou, Y. Sun, T. Wang, E. K. Lee, and N. S. Kim, "Analyzing Energy Efficiency of a Server with a SmartNIC under SLO Constraints," in *Proceedings of the 24th IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'23)*, 2023.

[24] Intel, "Intel Digital Random Number Generator (DRNG) Software Implementation Guide," https://www.intel.com/content/www/us/en/developer/articles/guide/intel-digital-random-number-generator-drng-software-implementation-guide.html.

[25] Intel, "Intel Infrastructure Processing Unit (Intel® IPU) ASIC E2000," https://www.intel.com/content/www/us/en/products/details/network-io/ipu/e2000-asic.html.

[26] Intel, "Intel Intelligent Storage Acceleration Library," https://www.intel.com/content/www/us/en/developer/tools/isa-l/overview.html.

[27] Intel, "Intel Xeon Gold 6140 Processor," https://www.intel.com/content/www/us/en/products/sku/120485/intel-xeon-gold-6140-processor-24-75m-cache-2-30-ghz/specifications.html.

[28] Intel, "Intel® Advanced Vector Extensions 512," https://www.intel.com/content/www/us/en/architecture-and-technology/avx-512-solution-brief.html.

[29] Intel, "Intel® In-Memory Analytics Accelerator (Intel® IAA)," https://www.intel.com/content/www/us/en/content-details/780887/intel-in-memory-analytics-accelerator-intel-iaa.html.

[30] Intel, "Intel® QuickAssist Technology (Intel® QAT)," https://www.intel.com/content/www/us/en/architecture-and-technology/intel-quick-assist-technology-overview.html.

[31] Intel, "What Is Hyper-Threading?" https://www.intel.com/content/www/us/en/gaming/resources/hyper-threading.html.

[32] Intel, "What Is Intel® Turbo Boost Technology?" https://www.intel.com/content/www/us/en/gaming/resources/turbo-boost.html.

[33] H. Ji, Y. Sun, M. Mansi, Y. Yuan, J. Huang, R. Kuper, M. M. Swift, and N. S. Kim, "STYX: Exploiting SmartNIC Capability to Reduce Datacenter Memory Tax," in *Proceedings of USENIX Annual Technical Conference (USENIX ATC'23)*, 2023.

[34] JiakunYan, "dpu-pingpong," https://github.com/JiakunYan/dpu-pingpong.

[35] U. Y. Kakaiya, S. Kumar, R. M. Sankaran, and P. Sethi, "Scalable I/O Between Accelerators and Host Processors," https://www.intel.com/content/www/us/en/developer/articles/technical/scalable-io-between-accelerators-host-processors.html.

[36] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd IEEE/ACM International Symposium on Computer Architecture (ISCA'15)*, 2015.

[37] V. Kapoor, V. S. Abraham, and R. Singh, "Elliptic Curve Cryptography," *Ubiquity*, 2008.

[38] S. Karamati, C. Hughes, K. S. Hemmert, R. E. Grant, W. W. Schonbein, S. Levy, T. M. Conte, J. Young, and R. W. Vuduc, ""Smarter" NICs for faster molecular dynamics: a case study," in *Proceedings of the 36th IEEE International Parallel and Distributed Processing Symposium (IPDPS'22)*, 2022.

[39] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "LineFS: Efficient SmartNIC Offload of a Distributed File System with Pipeline Parallelism," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM (SOSP'21)*, 2021.

[40] T. Kim, D. M. Ng, J. Gong, Y. Kwon, M. Yu, and K. Park, "Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*, 2023.

[41] J. Li, Y. Lu, Q. Wang, J. Lin, Z. Yang, and J. Shu, "AlNiCo: SmartNIC-accelerated Contention-aware Request Scheduling for Transaction Processing," in *Proceedings of USENIX Annual Technical Conference (USENIX ATC'22)*, 2022.

[42] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A Holistic Approach to Fast In-Memory Key-Value Storage," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*, 2014.

[43] J. Lin, A. Cardoza, T. Khan, Y. Ro, B. E. Stephens, H. Wassel, and A. Akella, "RingLeader: Efficiently Offloading Intra-Server Orchestration to NICs," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*, 2023.

[44] linux-rdma, "perftest," https://github.com/linux-rdma/perftest.

[45] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance Characteristics of the BlueField-2 SmartNIC," *CoRR abs/2105.06619*, 2021.

[46] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers," in *Proceedings of USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'19)*, 2019.

[47] Marvell, "Marvell LiquidIO III," https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf.

[48] Microsoft Azure, "Total Cost of Ownership (TCO) Calculator," https://azure.microsoft.com/en-us/pricing/tco/calculator/.

[49] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting Network Support for RDMA," in *Proceedings of Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'18)*, 2018.

[50] Y. Moon, S. Lee, M. A. Jamshed, and K. Park, "AccelTCP: Accelerating Network Applications with Stateful TCP Offloading," in *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation (NSDI'20)*, 2020.

[51] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)," https://csrc.nist.gov/publications/detail/fips/186/4/final, 2013.

[52] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)," https://csrc.nist.gov/publications/detail/fips/180/4/final, 2015.

[53] Network Working Group, "RFC 1631: The IP Network Address Translator (NAT)," https://www.rfc-editor.org/rfc/rfc1631.

[54] Network World, "Speed race: Just as 400Gb Ethernet gear rolls out, an 800GbE spec is revealed," https://www.networkworld.com/article/3538789/speed-race-just-as-400gb-ethernet-gear-starts-rolling-out-800gb-ethernet-gets-standardized.html, 2019.

[55] NVIDIA, "DOCA Document," https://docs.nvidia.com/doca/sdk/index.html.

[56] NVIDIA, "NVIDIA BlueField-2 DPU: Data Center Infrastructure on a Chip," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf.

[57] NVIDIA, "NVIDIA BlueField DPU BSP," https://docs.nvidia.com/networking/display/BlueFieldDPUOSLatest.

[58] NVIDIA, "NVIDIA CONNECTX-6 Dx," https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectX-6-dx-datasheet.pdf.

[59] NVIDIA, "NVIDIA DPU POWER EFFICIENCY White Paper," https://resources.nvidia.com/en-us-accelerated-networking-resource-library/nvidia-dpu-power-eff.

[60] Optocrypto, "Intel Sapphire Rapids with HBM2E, CXL 1.1, and PCIe 5.0 by end of 2022," https://optocrypto.com/intel-sapphire-rapids-with-hbm2e-cxl-1-1-and-pcie-5-0-by-end-of-2022/.

[61] S. Oswal, A. Singh, and K. Kumari, "DEFLATE COMPRESSION ALGORITHM," *International Journal of Engineering Research and General Science*, 2016.

[62] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*, 2015.

[63] Y. Qiu, J. Xing, K.-F. Hsu, Q. Kang, M. Liu, S. Narayana, and A. Chen, "Automated SmartNIC Offloading Insights for Network Functions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM (SOSP'21)*, 2021.

[64] Redis, "Introduction to Redis: Learn about the Redis open source project," https://redis.io/docs/about/.

[65] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Commun. ACM*, 1978.

[66] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," *Found. Trends Inf. Retr.*, 2009.

[67] M. Roesch, "Snort: Lightweight Intrusion Detection for Networks," in *Proceedings of the 13th Systems Administration Conference (LISA'99)*, 1999.

[68] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "Xenic: SmartNIC-Accelerated Distributed Transactions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM (SOSP'21)*, 2021.

[69] R. Shashidhara, T. Stamler, A. Kaufmann, and S. Peter, "FlexTOE: Flexible TCP Offload with Fine-Grained Parallelism," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*, 2022.

[70] SourceForge, "Linux layer 7 packet classifier," https://sourceforge.net/projects/l7-filter/.

[71] A. Sriraman and A. Dhanotia, "Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[72] Stratosphere Lab, "Stratosphere Datasets," https://www.stratosphereips.org/datasets-overview.

[73] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A SmartNIC-driven Accelerator-centric Architecture for Network Servers," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020.

[74] X. Wang, Y. Hong, H. Chang, K. Park, G. Langdale, J. Hu, and H. Zhu, "Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019.

[75] Z. Wang, H. Huang, J. Zhang, F. Wu, and G. Alonso, "FpgaNIC: An FPGA-based Versatile 100Gb SmartNIC for GPUs," in *Proceedings of USENIX Annual Technical Conference (USENIX ATC'22)*, 2022.

[76] X. Wei, R. Chen, Y. Yang, R. Chen, and H. Chen, "Characterizing Off-path SmartNIC for Accelerating Distributed Applications," in *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)*, 2023.

[77] K. Wiles, "The Pktgen Application," https://pktgen-dpdk.readthedocs.io/en/latest/contents.html.

[78] Xilinx, "The Composable SmartNIC: The Alveo SN1000 SmartNIC," https://www.xilinx.com/applications/data-center/network-acceleration/alveo-sn1000.html.

[79] T. Xing, H. Tajbakhsh, I. Haque, M. Honda, and A. Barbalace, "Towards Portable End-to-End Network Performance Characterization of SmartNICs," in *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'22)*, 2022.

[80] Yoctopuce, "Yocto-Watt," http://www.yoctopuce.com/EN/products/usb-electrical-sensors/yocto-watt.

[81] Y. Yuan, J. Huang, Y. Sun, T. Wang, J. Nelson, D. R. K. Ports, Y. Wang, R. Wang, C. Tai, and N. S. Kim, "RAMBDA: RDMA-driven Acceleration Framework for Memory-intensive µs-scale Datacenter Applications," in *Proceedings of the 29th IEEE International Symposium on High-Performance Computer Architecture (HPCA'23)*, 2023.

[82] Y. Yuan, Y. Wang, R. Wang, and J. Huang, "HALO: Accelerating Flow Classification for Scalable Packet Processing in NFV," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19)*, 2019.

[83] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-Resolution Measurement of Data Center Microbursts," in *Proceedings of Internet Measurement Conference*, 2017.

[84] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, and J. Sherry, "Achieving 100Gbps Intrusion Prevention on a Single Server," in *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, 2020.

Appendix

1. Abstract

We present how to run the three microbenchmarks and ten benchmarks on both the host CPU and the SNIC processor, including detailed information on our system setup, a comprehensive guide on the software installation, examples of running microbenchmarks and benchmarks.

2. Artifact check-list (meta-information)

- **Program:** `DPDK-Pktgen` [77], `Redis` [64], `YCSB` [15] and `Snort` [67].
- **Data set:** A PCAP trace from Stratosphere [72] and a registered rule set [7].
- **Run-time environment:** Ubuntu 18.04.6 LTS (Linux 5.4.0), Ubuntu 18.04.6 LTS (Linux 4.15.0), and Ubuntu 20.04.5 LTS (Linux 5.4.0-1049-bluefield) for the server, the client, and the SNIC, respectively. and the SNIC with .
- **Hardware:** Server: Intel Xeon Gold 6140 CPUs with 100 Gbps BlueField-2 for the server. Client: Intel Xeon E5-2640 CPU and 100 Gbps ConnectX-6 Dx. Power measurement setup: Yocto-Watt power sensor, a PCIe riser card, and a USB cable.
- **Execution:** Hyper Threading and Turbo Boost are disabled to reduce performance variations and interference.
- **Experiments:** The server host CPU or the SNIC processor runs microbenchmarks and benchmarks while receiving packets from the client.
- **Publicly available:** yes
- **Code licenses:** BSD 3-Clause License

3. Description

3.1. How to access.

Acquire the artifact from the following repository: https://zenodo.org/record/8313202. The detailed instructions and other relevant information can be found in the repository's README files.

3.2. Hardware dependencies.

The experiments require that the client should be directly connected to the server through a 100 Gbps network cable. The client needs a DPDK-compatible and RDMA-supported 100 Gbps NIC. The server is equipped with an SNIC (BlueField-2). To support measurements of the system-wide power consumption, the server needs to support the Intelligent Platform Management Interface (IPMI). A PCIe riser card needs to be connected to two Yocto-Watt power sensors to measure the power consumption of the NIC and the SNIC.

3.3. Software dependencies.

The software packages for TCP/UDP, DPDK and RDMA should be installed on the server, the client and the SNIC. For BlueField-2, the DOCA (*i.e.*BlueField-2 SDK) [55] package should be installed on the SNIC. Although Ubuntu 18.04.6 LTS, DOCA 1.5.0, DPDK 22.07, and MLNX_OFED 5.8-1.0.1.1-LTS are installed for our evaluations, other versions should work, too.

3.4. Data sets.

The PCAP trace and the registered rule set can be obtained from https://www.stratosphereips.org/datasets-overview and https://www.snort.org/downloads/#rule-downloads, respectively.

4. Installation

On a server running Ubuntu 18.04.6 LTS, the software packages for DOCA (*i.e.*, BlueField-2 SDK), RDMA and DPDK should be installed. Refer to the repository for comprehensive infomrmation on all the benchmarks and related software dependencies.

For the power measurement setup, the power pin-outs from the PCIe riser card are connected to the Yocto-Watt power sensors, following the PCIe connector pin-out definition available at https://en.wikipedia.org/wiki/PCI_Express. A picture of this setup is shown in Fig. 3. The PCIe riser card is connected between the server and the SNIC. the Yocto-Watt power sensors send traces of measured power values to the server through a USB cable.

5. Evaluation and expected results

We illustrate the evaluation process using one benchmark for each networking stack as a representative example. These benchmarks can be used to run experiments shown in Fig. 4 and Fig. 6. For a complete set of benchmarks, refer to the associated repository.

5.1. Power measurement.

The system-wide power consumption of the server can be obtained by using `ipmitool`, while the power consumption of the SNIC is measured by the Yocto-Watt power sensors:

```
$ cd power_measure_tool
$ sudo python3 get_power.py <duration> <power data file name>
```

The duration of a power measurement should be set based on the application's running time. Alternatively, a long duration can be set, and then the measurement can be manually terminated by `Ctrl+C`. This also ensures that the measured power values are sent to the server.

5.2. TCP/UDP-based benchmark.

We choose `NAT` as an example. Network packets are sent from the client to the server CPU or the SNIC CPU.

```
$ cd software_offload_function/tcp_udp/udp_app
$ make
$ (Client) bash udp_client_nat.sh
$ (Server/BF-2) bash udp_server_nat.sh
```

Latency values will appear on the client side, and throughput values will appear on the server or SNIC side after Ctrl+C.

5.3. RDMA-based benchmark.

we choose `fio` as an example. Before the benchmark runs, both `HOST_MLX_IP` in `request_setup.sh` and `MLX_IP`

in target_setup.sh should be modified to match the IP addresses of the network interface of the client.

```
$ (Client) sudo bash target_setup.sh
$ (Server/BF-2) sudo bash request_setup.sh
$ (Server/BF-2) sudo bash fio_randread.sh
$ (Server/BF-2) sudo bash fio_randwrite.sh
```

Both the server CPU and the SNIC CPU can run fio to access a remote storage device in the client serving as a storage server.

5.4. DPDK-based benchmark.

We choose REM as an example and run DPDK-Pktgen on the client to send network packets to the server at specific <traffic_rate>. The server or SNIC will use given <rule_set> and NIC's <pci_address>.

```
$ (Client) sudo bash DPDK-Pktgen-setup.sh
$ (Client) sudo bash DPDK-Pktgen-exe.sh
$ (Client) Pktgen: set 0 rate <traffic_rate>
$ (Client) Pktgen: start 0
```

```
$ (BF-2) sudo bash rxp-bf.sh <rule_set>
$ (Server) sudo bash rxp-server.sh <rule_set> <pci_addr>
```

We can also configure DPDK-Pktgen with specific packet rates and sizes.

6. Experiment customization

Experiments can be conducted using various network packet traces and rule sets. Additionally, several parameters for each benchmark, such as batch sizes and packet sizes, can be changed.

7. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html