HAL: Hardware-assisted Load Balancing for Energy-efficient SNIC-Host Cooperative Computing

Jinghan Huang*, Jiaqi Lou*, Srikar Vanavasam*, Xinhao Kong†, Houxiang Ji*,
Ipoom Jeong*, Danyang Zhuo†, Eun Kyung Lee‡, Nam Sung Kim*

*University of Illinois Urbana-Champaign, †Duke University, ‡IBM Research

***Abstract*—A typical SmartNIC (SNIC) integrates a processor comprising Arm CPU and accelerators with a conventional NIC. The processor is designed to energy-efficiently execute network functions frequently used by datacenter applications. With such a processor, the SNIC has promised to notably improve the system-wide energy efficiency of datacenter servers. Nevertheless, the latest trend of integrating accelerators into server CPUs for these functions sparks a question on the SNIC processor's superiority over a host processor (*i.e.*, server CPU with accelerators) in system-wide energy efficiency, especially under given tail latency constraints. Answering this question, we first take an Intel Xeon processor, integrated with various accelerators (*e.g.*, QuickAssist Technology), as a host processor, and then compare it to an NVIDIA BlueField-2 SNIC processor. This uncovers that (1) the host accelerator, coupled with a more powerful memory subsystem, can outperform the SNIC accelerator, and (2) the SNIC processor can improve system-wide energy efficiency only at low packet rates for most functions under tail latency constraints. To provide high system-wide energy efficiency without compromising tail latency at any packet rates, we propose HAL, consisting of a hardware-based load balancer and an intelligent load balancing policy implemented inside the SNIC. When HAL determines that the SNIC processor cannot efficiently process a given function beyond a specific packet rate, it limits the rate of packets to the SNIC processor and lets the host processor handle the excess. We implement a HAL-enabled SNIC with a commodity FPGA and a BlueField-2 SNIC, plug it into a commodity server, and run 10 popular network functions. Our evaluation shows that HAL can improve the system-wide energy efficiency and throughput of the server running these functions by 31% and 10%, respectively, without notably increasing the tail latency.**

I. Introduction

The single-thread performance of CPUs has remained stagnant due to the demise of Dennard scaling [17]. Meanwhile, as the speed of networks gets faster in datacenters, server CPUs are required to process more packets per second. For example, to fully use the bandwidth of 100Gbps Ethernet, a server CPU core operating at 2.5GHz must process 625-million packets per second when the size of the packets is 20 bytes. This gives the CPU core only 4 cycles to process one packet. Furthermore, the server CPUs often process the packets with various network functions such as compression, cryptography, and packet inspection, consuming a significant fraction of cycles and energy. This motivates hyperscalers to explore the use of a SmartNIC (SNIC), which integrates a low-cost/energy-efficient processor with a conventional NIC for executing these functions. For instance, an NVIDIA BlueField-2 SNIC (BF-2) processor comprises an Arm CPU and accelerators [51], while the AMD SN1000 SNIC processor integrates such a processor with an FPGA [75]. These SNICs have inspired a large body of research work (*e.g.*, [14], [21], [34], [47], [64], [73], [82]). Meanwhile, it is noteworthy that server CPUs have also begun to be integrated with accelerators, such as Intel® QuickAssist Technology (QAT) [31], to efficiently execute these functions. This trend sparks a question on whether the SNIC processor can truly offer higher *system-wide* energy efficiency than such server CPUs, especially when holistically considering other important metrics for datacenter applications, such as tail latency.

Answering this question, we first set up a commodity server with a 100Gbps BF-2 and a QAT-equipped Intel Xeon CPU (also referred to as the host processor hereafter). Next, we take 10 popular network functions based on Data Plane Development Kit (DPDK) [25], a widely adopted stack for intra-datacenter networking [19]. We then proceed to evaluate system-wide energy efficiency and 99th-percentile (p99) latency while running these functions on the SNIC processor and the host processor, respectively. From this evaluation, we make two observations. (1) The host accelerators, coupled with a more powerful memory subsystem, outperform the SNIC accelerators for the functions supported by both processors in throughput, system-wide energy efficiency, and p99 latency at high packet rates. (2) The SNIC processor can improve system-wide energy efficiency over the host processor only at low packet rates for most functions under p99 latency constraints. These observations remain unchanged even with the latest 200Gbps BlueField-3 (BF-3) SNIC, as the host processor also has become more powerful with more CPU cores and accelerators.

These suggest that we should use both processors cooperatively to accomplish both high energy efficiency and low p99 latency simultaneously. To efficiently support such cooperative computing, we propose HAL, a **H**ardware-**A**ssisted **L**oad balancer in this work. HAL aims to balance the load between two processors such that the SNIC processor receives only what it can efficiently process, while the host processor handles the excess. As the host processor handles high packet rates only intermittently [6], [81], and the SNIC processor undertakes low packet rates most of the time, HAL can achieve both high energy efficiency and low p99 latency at any packet rates. Specifically, we propose HAL consisting of (1) a hardware-based load balancer integrated with SNIC hardware and (2)

an intelligent load balancing policy running on an SNIC CPU core, described below.

Hardware-based Load Balancer. It constantly monitors the rate of received packets and compares it with a threshold set by the load balancing policy running on an SNIC CPU core. When it finds that the rate exceeds the threshold, it limits the rate of packets delivered to the SNIC processor to the threshold while replacing the IP/MAC address of the excess packets with that of the host processor. After receiving those packets with two different destination IP/MAC addresses from the load balancer, the embedded switch hardware (§II-A) forwards them to their respective destinations (*i.e.*, SNIC or host processor). Lastly, it takes packets sent from the host processor and modifies the source IP/MAC address of these packets to the IP/MAC address associated with the SNIC processor. This guarantees that the packets from the host processor are correctly delivered to the clients that expect to receive packets with the source IP/MAC address linked to the SNIC processor.

Intelligent Load Balancing Policy. It periodically checks the occupancy of DPDK Rx queues in the SNIC CPU cores and the throughput of the SNIC processor using DPDK APIs to assess whether the SNIC processor is efficiently processing the received packets. Subsequently, it determines an appropriate threshold based on the occupancy and the throughput. Lastly, it sets the determined threshold to the hardware load balancer.

With a PCIe-attached SNIC (PCIe-SNIC), HAL described above is efficient only for stateless functions. To address such a limitation, we propose integrating HAL with an emerging CXL-attached SNIC (CXL-SNIC).

CXL-attached SNIC for Stateful Functions. When the SNIC processor and the host processor cooperatively process packets with a given stateful function, both may access and change the states of the function. This demands shared memory and cache coherence between the SNIC processor and the host processor which PCIe-SNIC cannot efficiently provide. Nonetheless, HAL can efficiently handle stateful functions, exploiting the hardware-managed shared memory and cache coherence features of CXL-SNIC. Since no CXL-SNIC has been available yet, to demonstrate the efficacy of HAL for stateful functions, we take an emulation approach based on the insights below. CXL-SNIC is expected to implement the CXL.cache protocol [3]. As the CXL.cache protocol is derived from the Intel UPI [27] protocol used for cache-coherent NUMA servers, it shares many characteristics with the UPI protocol. Analyzing SNIC architectures, we observe that the SNIC processor is connected to the networking subsystem of the NIC through (on-chip) PCIe (§II-A). Therefore, we can emulate a server with a CXL-SNIC using a NUMA server. One node connected to a standard NIC through (off-chip) PCIe serves as a CXL-SNIC, while the other socket serves as the host.

We evaluate HAL using a commodity server, three datacenter network traffic workloads, and ten popular stateless and stateful functions. Our evaluation shows that HAL does not compromise p99 latency while improving system-wide energy efficiency and throughput by 31% and 10%, respectively, compared to the host processor processing every packet.

II. Background

A. SNIC Architecture

An SNIC integrates a conventional NIC with a processor comprising CPU, accelerators, and/or FPGA. For example, BF-2 is a CPU-based SNIC integrating NVIDIA ConnectX-6 Dx, a conventional 100Gbps NIC [52], with the following subsystems: (1) 8 Arm A72 CPU cores, 256KB L1 data and 384KB instruction caches per core, 1MB L2 cache per two cores, and shared 6MB L3 cache; (2) a cache-coherent on-chip interconnect (*e.g.*, Arm Coherent Hub Interface (CHI) [5]); (3) an embedded switch (eSwitch); (4) a PCIe switch; (5) a DDR4-3200 DRAM controller; (6) 16GB on-board DRAM; and (7) accelerators, as depicted in Fig. 1.

The BF-2 processor provides three accelerators for network functions commonly used by datacenter applications: (1) regular expression matching (REM), (2) cryptography, and (3) (de)compression. The REM accelerator checks whether packets received through DPDK APIs contain specific string patterns. The SNIC CPU programs the string patterns as a compiled ruleset to the accelerator through Data Center-On-a-Chip Architecture (DOCA) Software Development Kit (SDK) APIs [50]. The cryptography accelerator supports 24 public key acceleration (PKA) algorithms, such as Rivest, Shamir, and Adleman (RSA) [61], Advanced Encryption Standard (AES) [15], Diffie–Hellman (DH) [48], Digital Signature Algorithm (DSA) [56], and Secure Hashing Algorithm (SHA) [55]. The SNIC CPU programs a specific PKA algorithm to a designated SNIC memory region, and then it controls accelerator operations through a memory-mapped register. The (de)compression accelerator is based on Deflate [58], and (de)compresses a given file received through DPDK APIs.

In addition, BF-2 offers hardware-assisted acceleration of Open vSwitch (OvS) [60], a software implementation of a

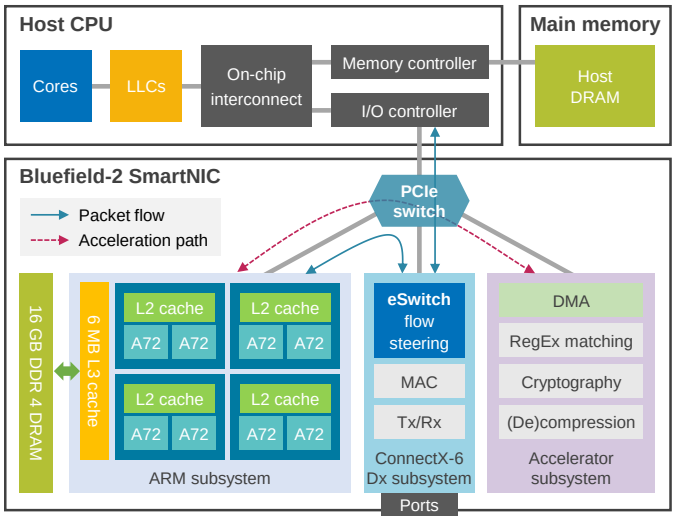


Fig. 1: NVIDIA BlueField-2 SNIC architecture.

virtual multilayer network switch. The eSwitch (Fig. 1) serves as the OvS data plane and forwards packets received at the Ethernet port to either the host CPU or the SNIC CPU based on forwarding rules set by the SNIC CPU acting as the OvS control plane. That is, an SNIC connected to the host CPU through the PCIe switch can operate as an independent system running its own OS.

B. Compute Express Link (CXL)

CXL is built on the physical layer of PCIe, the industry-standard interconnect technology to connect a CPU with I/O devices [9]. It defines three protocols: CXL.io, CXL.cache, and CXL.mem. CXL.io uses protocol features of the standard PCIe to initialize the interface between a CPU and a device. CXL.cache and CXL.mem allow the device and the host CPU to access the host CPU memory and the device memory, respectively. Incorporating these three protocols, CXL supports three device types for different use cases. Among them, the CXL Type-2 device, which uses all three protocols, facilitates fine-grained cooperative computing between the host CPU and a CXL-based device.

III. Throughput, Latency, and Energy Efficiency of SNIC-equipped Servers

A body of past work has analyzed the performance and/or energy efficiency of various SNICs [23], [44], [47], [74], [78]. In this section, following a methodology from the latest SNIC characterization work [23], we take ten DPDK-based functions that have been used to evaluate various SNIC proposals in the past [21], [47], [80], [82] (§VI). For a given function, if there is a corresponding accelerator in the SNIC processor and/or host processor, we execute the function using the accelerator. Otherwise, we run it on the SNIC CPU and/or host CPU to measure maximum throughput, system-wide energy efficiency, and p99 latency. Compared to the past work, we make the following two unique contributions in this section.

First, we use a host processor providing both hardware accelerators and Instruction Set Architecture (ISA) extensions to accelerate functions. For example, the Intel Skylake Xeon processor and its successor, the Intel Sapphire Rapids Xeon processor, also accelerate many functions with advanced technologies, such as QAT [31], Data Streaming Accelerator (DSA) [35], and In-memory Analytics Accelerator (IAA) [28] integrated in its Platform Controller Hub (PCH) chipset or processor die. Besides, Intel Xeon processors have provided ISA extensions (*e.g.*, AES-NI, SHA, RDRAND, RDSEED, and AVX) to accelerate cryptography and compression functions. These ISA extensions are exploited by optimized libraries, such as Intel ISA-L [29], [30], [40] and OpenSSL [57]. Table I lists the functions that can be accelerated by both the BF-2 accelerators and the optimized libraries using the QAT and/or ISA extensions. Second, we measure throughput, system-wide energy efficiency, and p99 latency change as a function of the received packet rate. Lastly, we demonstrate that the SNIC processor can improve system-wide energy efficiency over the host processor only at low packet rates for most functions.

TABLE I: BlueField-2 functions also supported by Intel ISA extensions ('ISA') and/or QAT.

Function	ISA	QAT	Function	ISA	QAT	Function	ISA	QAT
SHA	✓	✓	RSA	✓	✓	EC-DH	✓	✓
AES	✓	✓	DSA	✓	✓	EC-DSA	✓	✓
Deflate	✓	✓	RAND	✓	✓	GHASH	✓	
HMAC	✓	✓	MD5	✓		DES-EDE3	✓	
Whirlpool	✓		RMD160	✓		DES-CBC	✓	
Camellia	✓		RC2-CBC	✓		RC4	✓	
Blowfish	✓		SEED-CBC	✓		CAST-CBC	✓	
EdDSA	✓		MD4	✓		MD5	✓	

A. Throughput and Latency

Fig. 2 plots the maximum throughput and p99 latency of the ten functions running on the host processor and the SNIC processor with the Maximum Transmission Unit (MTU) packet size (1500 bytes). We set the packet rate to achieve the maximum throughput for a given function to measure the p99 latency. The throughput and latency of the SNIC processor are normalized to those of the host processor. Refer to Section VI for detailed information on the system setup, measurement methodology, and benchmark functions.

Hardware-accelerated Functions. Except for REM, the host processor provides the accelerators for the same functions as the SNIC processor. For the cryptography functions (*i.e.*, RSA, DH, and DSA), the host accelerator gives 24–115× higher maximum throughput and 95–99% lower p99 latency than

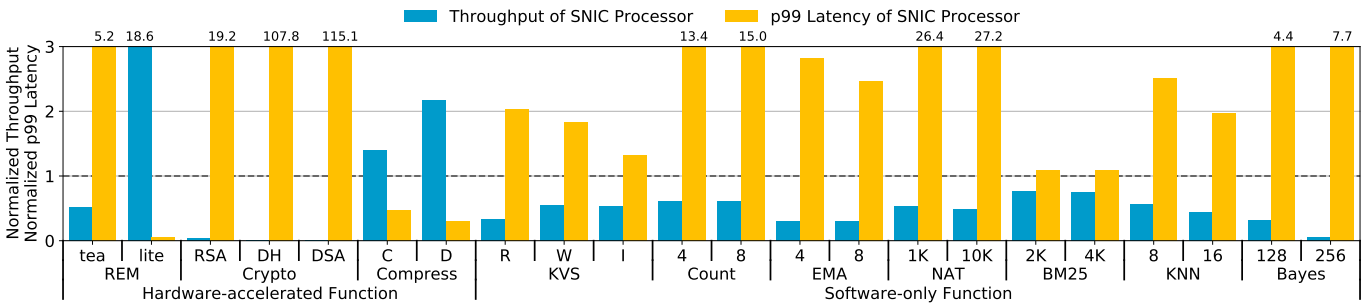


Fig. 2: Maximum throughput and p99 latency of the SNIC processor, normalized to those of the host processor.

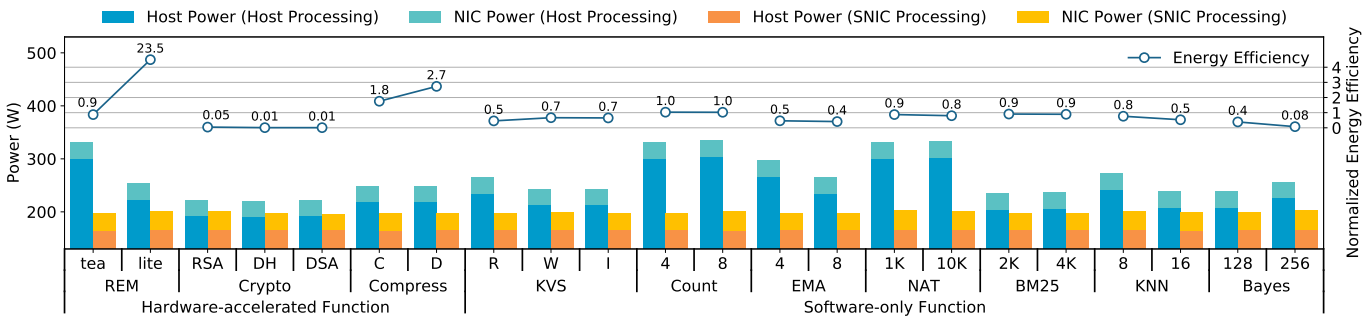


Fig. 3: Average power consumption and energy efficiency (*i.e.*, throughput/power) of a server using the BF-2 processor, normalized to those of a server using the host processor.

the SNIC accelerator. As these functions are also memory-intensive, the host accelerators, backed by a more powerful memory subsystem than the SNIC accelerators, perform better than the SNIC accelerators. For the (de)compression functions, which are more compute-intensive, the host accelerator offers 46–72% and 2.1–3.3× of the maximum throughput and p99 latency provided by the SNIC accelerator, respectively. However, the latest Intel Xeon processor, integrated with four accelerator instances, can offer higher overall throughput than the SNIC accelerator.

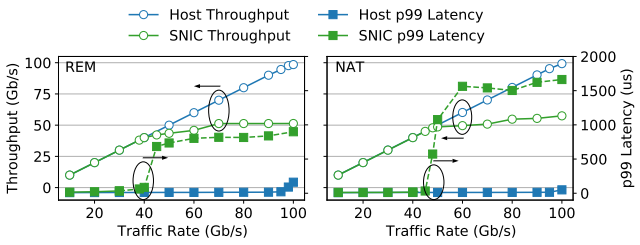
For the REM function, we compare the maximum throughput and p99 latency of the host CPU with those of the SNIC accelerator since the host processor does not have such an accelerator. For a simple ruleset (*e.g.*, `teakettle_2500` ruleset, `tea`), the host CPU can provide 93% higher maximum throughput and 81% lower p99 latency than the SNIC accelerator. This is because the host CPU can process multiple requests with multiple cores in parallel, almost linearly increasing the throughput with more cores. For a complex ruleset (*e.g.*, `snort_literals`, `lite`), however, the SNIC accelerator can outperform the host CPU, offering 19× higher maximum throughput and 94% lower p99 latency than the host CPU. Even so, the maximum throughput of the SNIC accelerator is limited to 50Gbps [53].

Lastly, the SNIC processor is physically closer to the eSwitch than the host processor, allowing it to potentially receive packets faster. Nonetheless, as both processors receive packets from the PCIe switch in the SNIC (Fig. 1), which largely dictates packet delivery latency, the SNIC processor does not get them notably faster. Our measurement shows that the latency difference between the two cases is only $\sim$0.3μs on average. Similarly, in a dual-socket server where only one socket is directly connected to a NIC or SNIC, it takes a longer time for the host processor in the other socket to receive the packets. This is because the packets need to go through an additional socket-to-socket cache-coherent interconnect (*e.g.*, Intel Ultra Path Interconnect (UPI)). Yet, it takes only $\sim$0.5 μs longer for the host processor in the remote socket to receive the packets than the host processor in the local socket. Such amounts of time increase the processing latency of complex applications like REM by less than 0.5%.

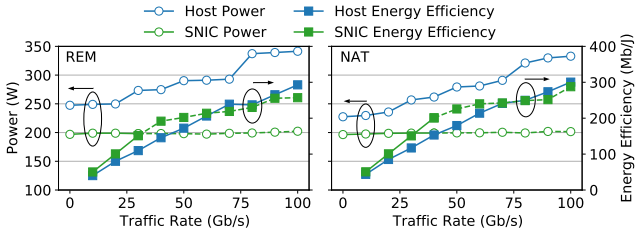
Software-only Functions. Although the SNIC CPU performs only the basic DPDK packet processing function, it cannot achieve the line rate (*e.g.*, 100Gbps for BF-2) for small-size packets. For example, although the SNIC CPU uses its all 8 cores for the DPDK packet processing function, it delivers throughput of only 40Gbps with 64-byte packets, which is slightly larger than a dominant packet size in datacenters (*i.e.*, 40 bytes [6]). With the MTU-size packets, which is another dominant packet size in datacenters [6], the SNIC CPU can accomplish the line rate but at 4.7× higher p99 latency than the host CPU. Finally, for the software-only functions, the SNIC CPU offers 24–69% lower maximum throughput and 1.1–27× higher p99 latency than the host CPU, because the SNIC CPU with wimpy cores and subsystems cannot compete with the host CPU.

B. Energy Efficiency

Fig. 3 compares the average power consumption and energy efficiency of a server using the SNIC processor with those of a server using the host processor. We measure the power consumption when a given function runs at the maximum sustainable throughput point shown in Fig. 2. The energy efficiency is obtained by dividing throughput with the average power consumption of the server. The SNIC itself consumes only a fraction of power that the host CPU consumes, but it cannot be used as a standalone device. Meanwhile, other system components such as host CPU, DRAM, and SSD consume a considerable amount of (static) power. For instance, our server and SNIC used for evaluations dissipate 194W and 29W, respectively, when they are idle. When processing packets at the maximum throughput points (Fig. 2), they consume 219–336W and 30–37W (Fig. 3), respectively. The small difference between the maximum and idle power consumption of the SNIC suggests that the power consumption of the SNIC is dominated by the other SNIC subsystems. Our measurement shows that the SNIC contributes to only 0.5–2% of the system-wide power consumption. That is, the energy efficiency of a server is primarily determined by the throughput, whether a given function is processed by the host processor or SNIC processor. As the host CPU consistently provides higher throughput than the SNIC CPU for processing



(a) Throughput and p99 latency



(b) System-wide server power consumption and energy efficiency

Fig. 4: Throughput and p99 latency (top), and system-wide power consumption and energy efficiency (bottom) versus packet rate for running REM (left) and NAT (right) on the host processor and SNIC processor. The dotted lines represent data points that the SNIC processor notably increases p99 latency.

the software-only functions, it gives 73% higher system-wide energy efficiency on average. Note that the SNIC CPU offers system-wide energy efficiency comparable to the host CPU for 'Count' because its slightly lower throughput is compensated by its lower power consumption. This does not suggest that we should use the SNIC CPU for performing such functions as it incurs 13.4–15.0× higher p99 latency than the host CPU.

C. Packet Rate vs. Energy Efficiency

Our analysis above has shown that relying solely on an SNIC is neither sufficient nor efficient for executing both the hardware-accelerated and software-only functions, especially at high packet rates. This is also supported by the latest SNIC characterization work [23] and further reinforced by our analysis using the host accelerators. Nonetheless, this does not suggest that using an SNIC is not beneficial at all.

Fig. 4 plots (a) throughput and p99 latency, and (b) system-wide power consumption and energy efficiency, as the packet rate increases. This shows that the SNIC processor can offer 38% and 31% higher system-wide energy efficiency than the host processor without notably increasing p99 latency when the packet rate is below 30Gbps and 41Gbps for REM and NAT, respectively. Note that the SNIC accelerator processing REM shows relatively constant p99 latency above 50Gbps, as it drops packets beyond this rate and only the latency of the processed packets is measured.

Table II shows the maximum throughput of the SNIC processor without increasing p99 latency. We denote this throughput as the service level objective (SLO) throughput. It also presents the system-wide energy efficiency achieved by the SNIC processor, normalized to that delivered by the host processor at the SLO throughput point. Depending on functions, we observe that the SNIC processor can improve system-wide energy efficiency by 14–55% without increasing p99 latency. However, the SLO throughput is often significantly lower than the line rate. This necessitates a load balancer and a load balancing policy capable of determining the SLO throughput point for a given function, restricting the rate of packets sent to the SNIC processor only up to that point, and forwarding any excess to the host processor.

TABLE II: SLO throughput of the SNIC processor ('SLO TP'), and system-wide energy efficiency by the SNIC processor, normalized to that by the host processor at the SLO throughput point ('SNIC EE').

	SLO TP	SNIC EE		SLO TP	SNIC EE
KVS	3 Gbps	1.19	KNN	7 Gbps	1.17
Count	58 Gbps	1.41	Bayes	0.1 Gbps	1.14
EMA	6 Gbps	1.17	REM	30 Gbps	1.38
NAT	41 Gbps	1.31	Cryp.	28 Gbps	1.33
BM25	1 Gbps	1.18	Comp.	43 Gbps	1.55

IV. Software-based Load Balancing

Section III demonstrated that we need load balancing between the SNIC processor and the host processor to leverage high energy efficiency of the SNIC processor at low packet rates and low p99 latency of the host processor at high packet rates. A load balancer can be designed to determine whether packets received by the SNIC should be processed by the SNIC processor or forwarded to the host processor, depending on the rate of received packets ($\texttt{Rate}_{\texttt{Rx}}$) and processing capability of the SNIC processor for a given function. In this section, we implement a software-based load balancer (`SLB`) with BF-2 and then demonstrate its limitation to stress the need for a hardware-assisted load balancer (§V).

To implement `SLB`, we leverage DPDK APIs and eSwitch-assisted OvS in BF-2 (§II-A). First, we set up two IP/MAC addresses: one for the SNIC (CPU) and the other for the host CPU to receive and process packets. Then we configure the SNIC's OvS accordingly. The clients send packets only to the IP/MAC address linked to the SNIC CPU. That is, the eSwitch serving as the OvS data plane will send all the packets from the clients to the SNIC CPU first. Second, the SNIC CPU continuously invokes `rte_eth_rx_burst()`, a DPDK API, to receive the packets. The SNIC CPU can obtain the number of packets received from each invocation of `rte_eth_rx_burst()`, based on which `SLB` running on the SNIC CPU determines $\texttt{Rate}_{\texttt{Rx}}$. If `SLB` observes a packet rate over a threshold ($\texttt{Fwd}_{\texttt{Th}}$), it takes a certain number of packets corresponding to $\texttt{Rate}_{\texttt{Fwd}}$ ($= \texttt{Rate}_{\texttt{Rx}} - \texttt{Fwd}_{\texttt{Th}}$) and then invokes `rte_eth_tx_burst()` with the IP/MAC address to the host CPU. This makes the SNIC CPU forward those packets to the host CPU through the eSwitch. Nonetheless, `SLB` needs to continuously compare the obtained packet rate with $\texttt{Fwd}_{\texttt{Th}}$ and invoke `rte_eth_tx_burst()` to transfer the packets to the host CPU through the following path: eSwitch

→ SNIC memory hierarchy → SNIC CPU → SNIC memory hierarchy→ eSwitch. This significantly increases the consumption of SNIC resources (*i.e.*, CPU cycles, cache capacity, on-chip interconnect bandwidth, and off-chip memory bandwidth) and the latency to transfer the packets from the NIC to the host CPU, especially as the packet rate increases.

Fig. 5 plots the throughput and p99 latency of NAT after deploying SLB. When using only one SNIC CPU core, SLB cannot forward the packets to the host CPU fast enough, ending up dropping 58–61% of packets for the range of $\texttt{Fwd}_{\texttt{Th}}$ values. With four SNIC CPU cores, the overall throughput approaches to 80Gbps at $\texttt{Fwd}_{\texttt{Th}}$ = 20Gbps as the SNIC CPU processes packets only at 20Gbps and SLB keeps up with forwarding the remaining packets at 60Gbps. However, this leads to even higher p99 latency than simply making the SNIC CPU process every packet without deploying SLB, due to the long latency of forwarding the packets to the host CPU. As $\texttt{Fwd}_{\texttt{Th}}$ increases to 60Gbps, the overall throughput with four SNIC CPU cores used for SLB decreases from 77Gbps to 53Gbps. This decrease occurs because the SNIC CPU struggles to process packets at 60Gbps only with four SNIC CPU cores. This falsely reduces p99 latency shown in Fig. 5, as the number of packets that NAT needs to process decreases due to more dropped packets at higher $\texttt{Fwd}_{\texttt{Th}}$. Lastly, although all the SNIC CPU cores are exclusively used for DPDK packet processing and SLB, the SNIC CPU presents 7× higher p99 latency than the host CPU for DPDK processing.

Note that we may have the host CPU run SLB, forwarding all the packets to the SNIC CPU when the packet rate is below $\texttt{Fwd}_{\texttt{Th}}$, and processing only the packets at $\texttt{Rate}_{\texttt{Fwd}}$. This approach can overcome the limited capability of the SNIC CPU for both running SLB and processing packets at high packet rates, but it comes with two disadvantages. First, it makes the power-hungry host CPU always active and consume expensive cycles to count and forward the packets. For instance, when the host CPU counts MTU-size packets at the line rate, it gives 40% lower system-wide energy efficiency than the SNIC CPU. Second, it makes the packets go through an even longer communication path (*i.e.*, SNIC eSwitch → host CPU (DPDK packet processing) → SNIC eSwitch → SNIC CPU (DPDK pakcet processing), instead of SNIC eSwitch → SNIC CPU or host CPU where → denotes on/off-chip PCIe crossing) and 2× more DPDK packet processing. These result in 2.3× longer p99 latency than HAL for processing MTU-size DPDK packets.

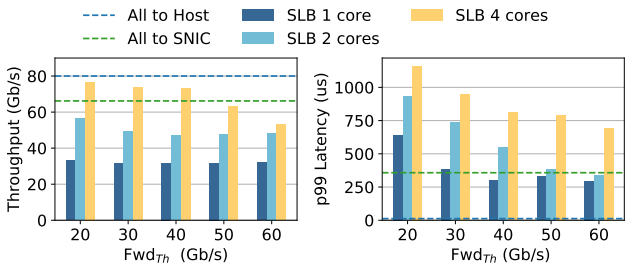


Fig. 5: Throughput and p99 latency of NAT with SLB. We vary the number of SNIC CPU cores processing DPDK packets and running SLB, and $\texttt{Fwd}_{\texttt{Th}}$ from 20Gbps to 60Gbps after making a client send packets to the SNIC at 80Gbps.

V. Hardware-assisted Load Balancing

To address the limitation of SLB (§IV), we propose HAL, consisting of a hardware-based load balancer (HLB) built in the SNIC and a lightweight intelligent load balancing policy (LBP) running on an SNIC CPU core. Specifically, HLB monitors $\texttt{Rate}_{\texttt{Rx}}$. Whenever HLB observes that $\texttt{Rate}_{\texttt{Rx}}$ exceeds $\texttt{Fwd}_{\texttt{Th}}$, it limits the rate of packets to the SNIC processor to $\texttt{Fwd}_{\texttt{Th}}$ set by LBP, and forwards the remaining packets directly to the host CPU. Compared to SLB, HLB eschews the high consumption of SNIC resources and the long latency associated with transferring packets through the lengthy forwarding path.

A. Hardware-based Load Balancer

Fig. 6 depicts an HAL-enhanced SNIC (or simply HAL-SNIC) where the eSwitch is integrated with HLB comprising ❶ traffic monitor, ❷ traffic director, and ❸ traffic merger. We prototype HAL HLB using an FPGA connected to BF-2 to evaluate an HAL-SNIC with a commodity server. Below, we delve into each component in detail.

❶ **Traffic Monitor.** In a conventional SNIC, whenever a packet is received at the tranceiver, it is passed to the media access control (MAC) unit, and then to the eSwitch (Fig. 1). In a HAL-SNIC, however, the packet will be passed from the MAC unit to the traffic monitor first. Upon receiving the packet, the traffic monitor inspects it to determine the number of bytes occupied by both the header and the payload. It then increments a counter (ReceivedBytes) by the number of received bytes. Lastly, the traffic monitor periodically checks ReceivedBytes (*e.g.*, every 10 μs) to determine $\texttt{Rate}_{\texttt{Rx}}$, passes $\texttt{Rate}_{\texttt{Rx}}$ to the traffic director, and then resets ReceivedBytes.

❷ **Traffic Director.** When a server boots up, HAL sets up two IP/MAC addresses: one for the SNIC (CPU) exposed to the clients and the other for the host CPU, which is hidden from the clients. Then, it configures the SNIC's eSwitch-assisted OvS accordingly in the same way as SLB. This setup makes the eSwitch automatically route the packet directly to either

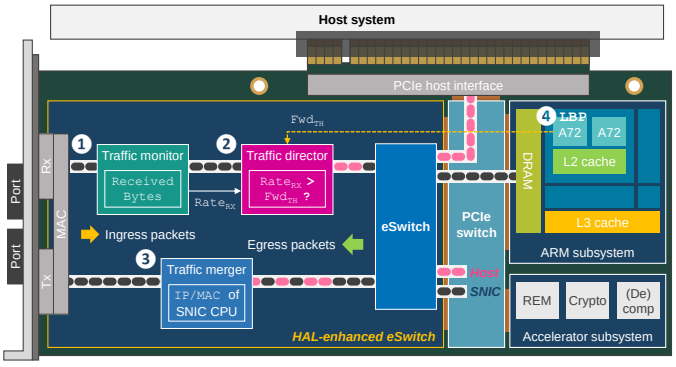


Fig. 6: HAL-enhanced SNIC overview.

the SNIC CPU or the host CPU, depending on the IP/MAC address in the destination field of the packet. Next, the traffic director periodically compares $\texttt{Rate}_{\texttt{Rx}}$ from the traffic monitor with $\texttt{Fwd}_{\texttt{Th}}$ from LBP. If $\texttt{Rate}_{\texttt{Rx}}$ is below $\texttt{Fwd}_{\texttt{Th}}$, the traffic director does not take any action. However, if $\texttt{Rate}_{\texttt{Rx}}$ exceeds $\texttt{Fwd}_{\texttt{Th}}$, the traffic director takes some of received packets in a round-robin fashion at $\texttt{Rate}_{\texttt{Fwd}}$. Subsequently, it modifies the IP/MAC address in the destination field of these packets and updates the checksum value of each modified packet. In both cases, every received packet is passed from the traffic director to the eSwitch. Lastly, the eSwitch automatically limits the rate of received packets to the SNIC CPU to $\texttt{Fwd}_{\texttt{Th}}$ while directing the remaining packets to the host CPU at $\texttt{Rate}_{\texttt{Fwd}}$.

❸ **Traffic Merger.** After the SNIC processor and the host processor complete processing a given function for the received packets, they send packets comprising responses to the clients. When they send the packets, only the IP/MAC address linked to the SNIC (CPU) is known to the clients. As such, to create the illusion of packets from the same physical source (*i.e.*, SNIC) to the clients, the traffic merger first intercepts the packets sent from the host CPU. Then, it replaces the IP/MAC address in the source field of the packets with the IP/MAC address associated with the SNIC, and updates the checksum values of the packets accordingly. This guarantees the packets from the host CPU are correctly delivered to and recognized by the clients.

Since evaluating HAL requires a hardware modification in an SNIC, we implement the traffic monitor, director, and merger blocks in an AMD Alveo U280 FPGA [76]. Subsequently, we integrate these blocks with the transceiver and MAC units provided by the FPGA. As the FPGA provides two 100Gbps Ethernet ports, one port is connected to the client and the other port is connected to the SNIC. As such, LBP communicates with the traffic director through Ethernet instead of the on-chip interconnect. However, considering the frequency and the amount of information (*i.e.*, $\texttt{Fwd}_{\texttt{Th}}$) sent through the Ethernet connection from the SNIC to the FPGA, we find that the network bandwidth consumption and communication latency between the SNIC and the FPGA are not notable (§VII). We may implement HAL on an FPGA-based NIC design such as Corundum [18], but we uncover that none has been able to support DPDK yet. Besides, the currently available P4-enabled SNIC [49] provides much lower networking and processing capabilities than BF-2.

B. Load Balancing Policy

Fig. 6 also shows ❹ LBP which runs on one of the SNIC CPU cores. It is designed to adaptively set $\texttt{Fwd}_{\texttt{Th}}$ based on performance characteristics of a given function running on the SNIC processor. When running a single function on an SNIC, we may profile the performance characteristics of the function to determine $\texttt{Fwd}_{\texttt{Th}}$ in advance. Alternatively, we may monitor the performance characteristics at run-time and dynamically determine $\texttt{Fwd}_{\texttt{Th}}$. Such a dynamic policy is beneficial when the SNIC runs more than one function (*e.g.*, the output of one function on the SNIC CPU is input to another function on the SNIC accelerator) and the performance characteristics of a given function vary for different inputs (§III).

Algorithm 1: LBP algorithm

```
1  Function set_forward_rate(SNIC_TP)
2      if Fwd_Th < SNIC_TP + Delta_TP then
3          RxQ_Occ = 0
4          for i ← 1 to num_queue do
5              if RxQ_Occ < rte_eth_rx_queue_count(i)
                then
6                  RxQ_Occ =
                     rte_eth_rx_queue_count(i)
7          if RxQ_Occ < WM_Low then
8              Fwd_Th += Step_Th
9          if RxQ_Occ > WM_High then
10             Fwd_Th -= Step_Th
```

Algorithm 1 describes LBP based on a simple greedy algorithm. It periodically monitors the throughput of SNIC processing ($\texttt{SNIC}_{\texttt{TP}}$) and the occupancy of DPDK Rx queues ($\texttt{RxQ}_{\texttt{Occ}}$). Prior work also exploits the Rx queue occupancy for DPDK load balancing [59], but it balances loads among the host CPU cores whereas HAL does so between the host processor and the SNIC processor. $\texttt{SNIC}_{\texttt{TP}}$ is estimated by accumulating the returned values from `rte_eth_rx_burst()`, which is continuously invoked to get packets from DPDK Rx queues, for a given period. $\texttt{RxQ}_{\texttt{Occ}}$ is determined by invoking `rte_eth_rx_queue_count()` `num_queue` times (one queue per SNIC CPU core) and taking the maximum value at the end of each period. If it observes that $\texttt{SNIC}_{\texttt{TP}}$ gets close to $\texttt{Fwd}_{\texttt{Th}}$ within a certain range ($\texttt{Delta}_{\texttt{TP}}$), it checks $\texttt{RxQ}_{\texttt{Occ}}$. If $\texttt{RxQ}_{\texttt{Occ}}$ is below the low watermark ($\texttt{WM}_{\texttt{Low}}$) or above the high watermark ($\texttt{WM}_{\texttt{High}}$), it implies that the SNIC processor is underutilized or overutilized, thereby increasing or decreasing $\texttt{Fwd}_{\texttt{Th}}$ by $\texttt{Step}_{\texttt{Th}}$. If $\texttt{RxQ}_{\texttt{Occ}}$ is between $\texttt{WM}_{\texttt{Low}}$ and $\texttt{WM}_{\texttt{High}}$, $\texttt{Fwd}_{\texttt{Th}}$ remains the same. Finally, we can further optimize Algorithm 1 to search the best $\texttt{Fwd}_{\texttt{Th}}$ faster by adaptively changing $\texttt{Step}_{\texttt{Th}}$ and $\texttt{Delta}_{\texttt{TP}}$ based on $\texttt{SNIC}_{\texttt{TP}}$ and $\texttt{RxQ}_{\texttt{Occ}}$.

HAL does not forward any packets to the host CPU at low packet rates (below $\texttt{Fwd}_{\texttt{Th}}$), which makes the host CPU cores simply busy-wait. To minimize the power consumption of the host CPU simply busy-waiting at the low packet rates, HAL exploits a DPDK power management API [12] that puts the host CPU cores into a sleep state. As soon as the API detects received packets in the DPDK Rx queues associated with the host CPU cores, it wakes up the sleeping host CPU cores. Although the API can improve energy efficiency, it also increases the p99 latency of host CPU processing due to the wake-up penalty. Our evaluation will capture the increase in not only energy efficiency but also the p99 latency.

C. Supporting Stateful Functions with CXL-SNIC

HAL facilitates cooperative computing using both the SNIC processor and the host processor. It seamlessly works for stateless functions based on DPDK, where a given packet can be processed by either the SNIC processor or the host

processor. However, for a stateful function, the state may change over time (*e.g.*, Key-Value Store updating its database), which can affect the output of the function for the same input. As such, if the SNIC processor and the host processor are to cooperatively process a given stateful function correctly, they need to share the memory space storing the state in a cache-coherent manner. Unfortunately, the PCIe interconnect used to connect an SNIC to the host CPU is not cache-coherent, and therefore HAL with a PCIe-SNIC cannot efficiently support stateful functions. To make HAL efficiently work for stateful functions, we propose integrating HAL with a CXL-SNIC where the SNIC and the host CPU are connected over the CXL interconnect (§II-B). There has been an announcement for a planned CXL Type-2 SNIC product from the industry [10].

Since no commodity CXL-SNIC has been available yet, we devise an emulation approach. First, revisiting the SNIC architecture, we find that an SNIC is a complete system with a CPU, cache, memory, and the (on-chip) PCIe interconnect that connects the SNIC CPU and its cache with the SNIC subsystems (§II-A). Second, a CXL-SNIC is supposed to be exposed as a NUMA node to the host CPU (§II-B). As such, we propose using a conventional dual-socket NUMA server as a server connected to a CXL-SNIC. One socket serves as a server, while the other socket with a standard NIC acts as a CXL-SNIC (Fig. 7). In fact, the CXL interconnect shared many characteristics with the Intel UPI [27], which connects two or more NUMA nodes in a server, as the CXL has evolved from the UPI [2], [7], [20]. To emulate asymmetric compute capability between the SNIC processor and the host processor, with two symmetric CPUs, we set the frequency of the CPU on the node emulating the CXL-SNIC (`node 1`) to the lowest value (800MHz). We find that the performance of the SNIC CPU at 2GHz is comparable to that of the host CPU at 800MHz when running SPEC-2017 `mcf` (1391s vs. 1424s), for instance. Besides, we use only 8 cores in the CPU on `node 1` and disable the rest, using the CPU hotplug feature in the Linux kernel [66]. A similar approach is used by the latest work for a cache-coherent NIC [63], [67].

VI. Evaluation Methodology

Systems. We use two systems: one for the server (processing network packets) and the other for a client (generating packets) for our evaluations. Table III summarizes the configurations

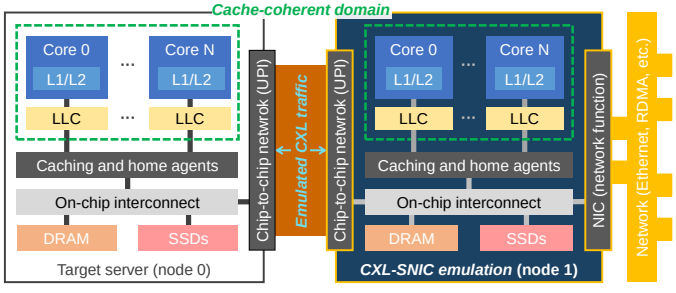


Fig. 7: CXL-SNIC emulation system leveraging NUMA.

TABLE III: Evaluation system configurations.

	Server	Client
OS	Ubuntu 20.04.5 LTS (Linux 5.15.0)	Ubuntu 22.04.1 LTS (Linux 5.15.0)
Processor Model	Intel Xeon Gold 6140 (Skylake)	Intel Xeon E5-2660 v4 (Broadwell)
# Cores	36 (2 sockets)	14 (1 socket)
LLC	100 MB (12 ways)	35 MB (20 ways)
System Memory	256 GB DDR4-2666 16 DIMMs, 12 channels	64 GB DDR4-2400 1 DIMM, 1 channel
NIC	BlueField-2	ConnectX-6 Dx
Driver Version	MLNX_OFED 5.8-1.0.1.1-LTS	
FPGA	Alveo U280 Data Center Accelerator Card	

of these two systems. The server is equipped with an SNIC (BF-2) while the client is equipped with a 100 Gbps NIC (ConnectX-6 Dx). In particular, for performance evaluation of the host CPU processing functions, we use the `userspace` governor [4] to set the frequency to 2.2GHz (*i.e.*, the maximum frequency under the TDP constraint) and disable Hyper-Threading and Turbo Boost to reduce performance variations and interference [1], [33]. To measure power consumption of a server and the end-to-end latency of each function, we use the same measurement mechanisms and methodologies as the latest BF-2 characterization work [23]. For example, we use the same sampling rate and resolution for power measurement. Finally, we connect the U280 FPGA, which implements the traffic monitor, director, and merger block, and the SNIC with an Ethernet cable. We emulate CXL-SNIC, replacing the SNIC with a standard NIC and treating one socket directly connected to the NIC as as a CXL-SNIC (Fig. 7).

Power measurement. We use Data Center Manageability Interface (DCMI) [11] to measure the system-wide power consumption of the server every 1s with $\pm$ 1W precision using the power sensor on the server motherboard and controlled by the Baseboard Management Controller (BMC) [69]. Since the DCMI-based power measurement cannot isolate the (S)NIC power consumption from the system-wide power consumption, we use another setup comprising a PCIe riser card and two Yocto-Watt [79] power sensors. It can measure the power consumption of a PCIe-attached device at 10Hz with $\pm$ 2mW precision.

Benchmark. In this work, we take ten network functions that have been used by datacenter applications, and run them on the following execution platforms: host CPU, BF-2 CPU, and/or BF-2 accelerator. These functions have been also used for evaluating various SNIC proposals in prior work [21], [47], [80], [82]. When setting up and running the functions, we follow the same methodology as the latest BF-2 characterization work [23], including selection of datasets, configurations, and operation types. As the BF-2 CPU has 8 cores, we also run these functions on 8 host CPU cores in our experiments if not specified otherwise. Table IV lists the name, description,

TABLE IV: Benchmark functions; (S): stateful function.

Function	Description & Configuration
KVS [42] (S)	Key-value store w/ `read`, `write`, & `insert`
Count [37] (S)	Frequency counting w/ 4 & 8 batch sizes
EMA [38] (S)	Exponential moving average w/ 4 & 8 batch sizes
NAT [22]	Network address translation w/ 1K & 10K entries
BM25 [62]	Search ranking w/ 2K & 4K terms
KNN [8]	K-nearest neighbours w/ 8 & 16 set sizes
Bayes [45]	Naive Bayes classifier w/ 128 & 256 features
REM [72]	RegEx. matching w/ `lite` & `tea` rulesets [32]
Cryptography [57]	Public key encryption w/ `RSA`, `DH`, & `DSA`
Compression [58]	Deflate w/ `Silesia-mozilla` file [36]

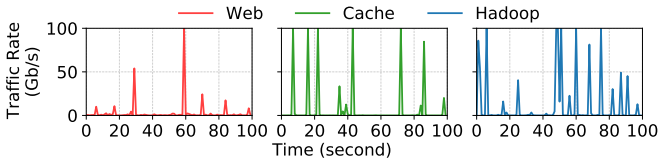


Fig. 8: Snapshots of packet rates of web, cache, and Hadoop for a 100-second duration with μ/σ equal to -1.37/1.97, -9/7.55, and -4.18/6.56 respectively, for the corresponding log-normal distributions. The average packet rates of these traces are 1.6Gbps, 5.2Gbps, and 10.9Gbps, respectively.

configurations, and source of each function.

Datacenter Network Traffic Workloads. To capture the effect of bursty changes in the rate of received packets at datacenter servers on the performance of HAL, we first obtain the cumulative distribution function (CDF) of network link utilization by three workloads: web, cache, and Hadoop from Meta [81]. Then, for each workload, we have the client synthetically generate packets at rates following a log-normal distribution [6] whose μ and σ are determined to follow the corresponding CDF. Fig. 8 shows snapshots of packet rates of web, cache, and Hadoop over time for a 100-second period with the statistical parameters of the three log-normal distributions we use to generate the traces. In our evaluation, the client generates packets for 10 minutes for each trace.

VII. Evaluation

In this section, we first evaluate HAL for two representative software-only and hardware-accelerated functions while sweeping the packet rates (§VII-A). Second, we evaluate HAL not only for individual stateless and stateful functions but also for various compositions of two pipelined functions while feeding three different datacenter network traffic workloads (§VII-B). Lastly, we analyze hardware, latency, power, and bandwidth costs of HAL (§VII-C).

A. Sweeping Packet Rates

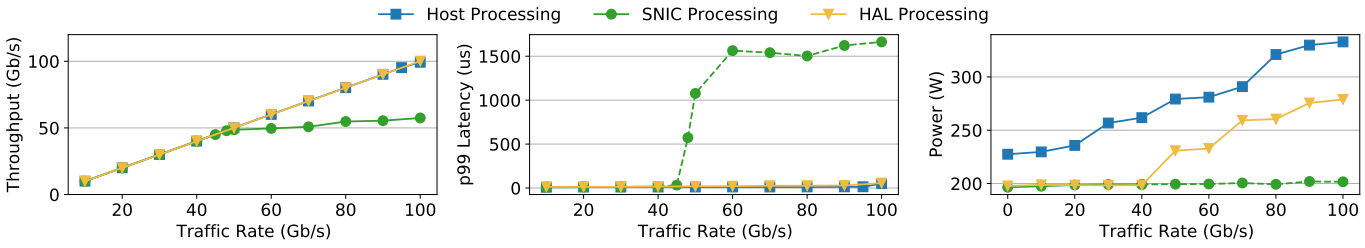
Fig. 9 plots the throughput, p99 latency and power consumption of the server using only the host CPU, the SNIC processor, and HAL for NAT and REM, respectively, as we sweep the packet rate. The left plots show that the SNIC processor processing NAT and REM begins to drop packets from 41Gbps and 30Gbps, respectively. Meanwhile, HAL leveraging the processing capability of the host CPU can provide linearly increasing throughput with the packet rates. The middle plots demonstrate that the SNIC processor begins to rapidly increase p99 latency at the rates exceeding 41Gbps and 30Gbps, giving 120× and 56× higher p99 latency than the host processor at 80Gbps for NAT and REM, respectively. In contrast, below the rates at which the SNIC processor begins to drop the packet, HAL gives almost the same latency as the SNIC processor within a ∼3% difference, which is due to the overhead of `HLB` (§VII-C) and measurement noises. We assume that the p99 latency constraint is determined by the SNIC processor processing a given function at low rates.

In the first-order approximation, the throughput and p99 latency of HAL are determined by the maximum values between those of the SNIC processor and those of the host processor, pls the overhead of `HLB` for p99 latency. However, HAL actually gives higher maximum throughput and/or lower p99 latency at the maximum throughput point since the host processor with HAL processes packets at lower rates than it should without HAL. For example, at 100Gbps, the host CPU processing REM gives 3.1× higher p99 latency than the host CPU with HAL, as the SNIC processor processes packets at the 30Gbps rate and the host CPU comfortably handles them only at the 70Gbps rate.

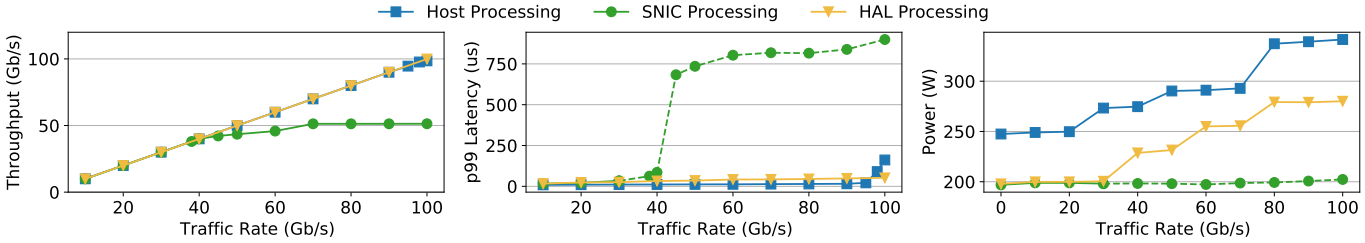
At the system level, the host CPU processing NAT and REM consumes 32–131W and 50–139W. That is, the host CPU consumes 25–69% and 53–99% higher power than the SNIC processor, respectively, to offer higher throughput and lower latency. Meanwhile, HAL gives the power consumption of the SNIC processor processing NAT and REM up to the rates at which the SNIC processor can efficiently process NAT (41Gbps) and REM (30Gbps). Even over those rates that the SNIC processor comfortably process NAT and REM, HAL consumes 31–63W and 36–61W, offering 11–24% and 12–27% lower power than the host CPU, because it uses the SNIC processor to process considerable percentages of packets, demonstrating the benefits of HAL for all the functions.

B. Datacenter Network Traffic Workload

Table V reports the maximum/average throughput, p99 latency, and average system-wide power consumption of the SNIC processor-only, the host processor-only, and HAL after we run the three datacenter network traffic workloads (§VI) for 6 functions: KNN, NAT, Count, EMA, cryptography, and REM. We do not evaluate Bayes, BM25, and KVS because the SNIC CPU presents very low maximum throughput for them (0.1–4Gbps). Such functions also show limited throughput on various other CPU-based SNICs as well [47], [68], [71]. We also do not evaluate the compression function because it is a stateful function processing a given file and the SNIC accelerator is not designed to cooperatively work with the host processor even under the assumption that they are connected to a cache-coherent interconnect.



(a) NAT



(b) REM

Fig. 9: Throughput, p99 latency, and power consumption across different network packet rates.

Single Stateless Function. We analyze the throughput, p99 latency, and average system-wide power after running the 4 stateless functions (KNN, NAT, REM and cryptography). HAL gives 12.6% (2.2%), 12.8% (2.2%), and 11.4% (4.3%) higher maximum (average) throughput than the host processor for web, cache, and Hadoop workloads, respectively, on average (geometric-mean). As discussed earlier, HAL also uses the SNIC processor together with the host processor, and thus it can give higher throughput than the host processor alone. On average, exploiting the processing capability of the host processor, HAL offers 64%, 80%, and 81% lower p99 latency than the SNIC processor for web, cache, and Hadoop workloads, respectively. Note that HAL provides generally higher p99 latency than the host processor. Nonetheless, as discussed earlier, these values are determined by the SNIC processor running at low rates and considered as the target p99 latency for each function in this work. Benefiting from higher energy efficiency of the SNIC processor processing these functions at low rates, HAL provides 28%, 30%, and 29% higher energy efficiency than the host processor for web, cache, and Hadoop workloads, respectively, on average.

Single Stateful Function. To examine the throughput, p99 latency, and energy efficiency of the 2 stateful functions (Count and EMA), we take the following methodology. We first make each function run on two NUMA nodes after modifying the function code to share some memory space for state variables through the cache-coherent UPI interconnect. Next, we run the function like a stateless one without sharing the memory space (ignoring the functional correctness) and compare the maximum throughput and p99 latency at the maximum throughput point with using cache coherence with those without using cache coherence. Lastly, we run the function on the SNIC processor and the host processor like stateless functions for our measured throughput, p99 latency, and power reported in Table V. We take such a methodology because the performance and power consumption of the emulated SNIC processor are different from the true SNIC processor while the throughput and latency difference between the two setups are negligible (*e.g.*, only 0.3%–0.4% lower maximum throughput and 1.7%–3.4% higher p99 latency for Hadoop). Our evaluation shows that HAL is effective for the stateful functions as well, giving 7.9% (1.9%), 10.4% (1.6%), and 5.1% (6.3%) higher maximum (average) throughput than the host processor; 68%, 94%, and 88% lower p99 latency than the SNIC processor; and 24%, 28%, and 28% higher energy efficiency than the host processor for web, cache, and Hadoop, respectively.

Two Pipelined Functions. To evaluate the efficacy of HAL in a practical SNIC usage scenario, we pick two functions and make the first function take packets from DPDK packet processing and feed its output to the input of the second function (*e.g.*, NAT after REM). We create four such combinations and run them using the SNIC processor, the host processor, and HAL. Our evaluation shows that HAL provides 8.3% (0.6%), 7.4% (1.4%), and 8.3% (2.5%) higher maximum (average) throughput than the host processor; 77%, 75%, and 89% lower p99 latency than the SNIC processor; and 32%, 35%, and 33% higher energy efficiency than the host processor for web, cache, and Hadoop, respectively.

C. Hardware, Latency, Power, and Bandwidth Costs

After the implementation of HAL, our analysis based on a report from AMD Vivado shows that HAL consumes 13,861 LUTs (*i.e.*, 1.1% of the U280 FPGA LUT resources). Compared with an FPGA-based NIC implementation [18], HAL consumes only 16.7% of LUTs, and it does not affect the operating frequency of an FPGA-based NIC implementation [18]. We also measure the network packet Tx/Rx latency

TABLE V: Throughput, p99 latency, and system-wide power consumption of the SNIC processor-only (SNIC), the host processor-only (Host), and HAL.

Workload	Function	Max. (*Avg.*) Throughput (Gbps)			p99 Latency (μs)			Average Power (W)		
		SNIC	Host	HAL	SNIC	Host	HAL	SNIC	Host	HAL
Web	KNN	15.0 (*1.3*)	30.3 (*1.5*)	42.3 (*1.6*)	446.8	44.9	49.4	199.5	243.8	207.3
	NAT	40.2 (*1.6*)	89.2 (*1.6*)	95.3 (*1.6*)	30.9	14.1	16.4	200.3	268.0	201.0
	Count	58.4 (*1.6*)	95.3 (*1.6*)	96.1 (*1.6*)	22.1	12.3	17.4	200.2	257.0	200.5
	EMA	11.6 (*1.4*)	56.9 (*1.6*)	64.7 (*1.6*)	245.4	30.6	31.2	200.0	245.5	202.3
	REM	42.5 (*1.6*)	93.2 (*1.6*)	95.5 (*1.6*)	34.2	16.3	20.1	199.9	248.5	203.7
	Crypto	39.2 (*1.5*)	83.1 (*1.5*)	87.4 (*1.6*)	44.1	15.2	18.5	199.6	260.3	202.8
	NAT + REM	31.2 (*1.4*)	83.3 (*1.6*)	93.2 (*1.6*)	915.8	25.5	32.2	199.6	248.9	201.0
	NAT + Crypto	42.3 (*1.4*)	81.3 (*1.6*)	86.4 (*1.6*)	33.9	15.5	24.5	199.4	280.1	205.8
	Count + REM	30.9 (*1.5*)	84.2 (*1.6*)	92.6 (*1.6*)	865.6	27.6	35.0	199.9	249.8	200.4
	Count + Crypto	44.9 (*1.5*)	82.8 (*1.6*)	87.3 (*1.6*)	44.9	14.8	26.5	200.3	294.4	208.5
Cache	KNN	15.6 (*2.5*)	31.2 (*4.5*)	43.9 (*5.0*)	2752.7	213.1	336.5	200.0	242.6	210.1
	NAT	41.4 (*4.5*)	89.9 (*5.2*)	97.3 (*5.2*)	318.4	16.8	35.7	200.1	279.5	202.6
	Count	58.5 (*4.9*)	95.3 (*5.2*)	97.2 (*5.2*)	243.6	19.3	44.3	199.7	257.1	200.7
	EMA	12.7 (*2.2*)	55.2 (*4.9*)	65.9 (*5.0*)	1238.4	93.9	96.1	200.1	264.9	207.7
	REM	43.7 (*4.1*)	93.3 (*5.2*)	95.3 (*5.2*)	790.8	38.9	53.2	199.9	255.6	204.9
	Crypto	46.0 (*4.8*)	89.0 (*5.1*)	92.3 (*5.1*)	327.5	166.8	222.8	199.8	269.6	205.9
	NAT + REM	30.4 (*3.4*)	84.0 (*5.0*)	92.3 (*5.1*)	964.0	68.9	100.6	199.3	256.1	207.7
	NAT + Crypto	42.1 (*2.9*)	87.0 (*5.1*)	91.3 (*5.1*)	310.0	131.2	153.1	200.5	295.7	211.6
	Count + REM	31.2 (*3.7*)	85.2 (*5.0*)	92.8 (*5.1*)	950.0	85.5	135.4	199.6	255.0	207.5
	Count + Crypto	51.3 (*3.3*)	85.2 (*5.1*)	90.2 (*5.1*)	437.2	160.0	199.4	200.5	319.4	214.6
Hadoop	KNN	19.6 (*2.6*)	31.5 (*8.4*)	44.2 (*9.5*)	2758.8	367.5	483.7	199.9	244.6	219.7
	NAT	45.5 (*9.2*)	91.6 (*10.6*)	97.9 (*10.9*)	343.4	48.8	61.3	199.6	280.4	205.8
	Count	58.0 (*10.1*)	99.3 (*10.6*)	99.6 (*10.9*)	213.0	40.1	65.0	199.4	261.8	203.4
	EMA	11.8 (*5.6*)	62.0 (*9.3*)	67.3 (*10.5*)	1217.2	183.1	188.8	200.1	265.8	215.7
	REM	43.5 (*8.1*)	93.6 (*10.8*)	95.2 (*10.9*)	811.8	77.1	94.5	200.1	257.7	217.9
	Crypto	58.1 (*9.4*)	93.5 (*10.6*)	94.3 (*10.7*)	452.3	170.0	231.7	199.5	285.8	216.6
	NAT + REM	33.6 (*5.6*)	84.1 (*10.1*)	93.6 (*10.5*)	1005.2	81.9	141.6	199.8	258.9	216.3
	NAT + Crypto	42.8 (*5.4*)	90.1 (*10.7*)	94.3 (*10.8*)	979.5	165.4	191.9	200.0	305.2	220.7
	Count + REM	34.5 (*5.7*)	85.2 (*10.2*)	93.9 (*10.6*)	998.2	96.7	136.3	200.3	257.6	217.3
	Count + Crypto	52.9 (*7.1*)	86.1 (*10.5*)	92.6 (*10.7*)	1601.3	166.0	193.0	199.9	328.3	225.7

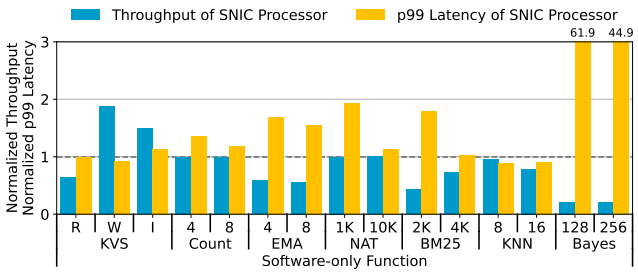
increase by HAL in an SNIC implementation. This shows that HAL increases the round trip latency of DPDK packets by only 8.3% (800ns), 45% of which (365ns [77]) is from the transceiver and MAC units. This latency increase by HAL can be practically eliminated in an ASIC implementation. Lastly, to estimate the power consumption of HAL implemented in the FPGA, we use AMD Vivado, which reports that HAL consumes less than 0.1W. Note that an ASIC implementation consumes 14$\times$ lower power than an FPGA implementation for the same function with the same technology [39].

VIII. Discussion

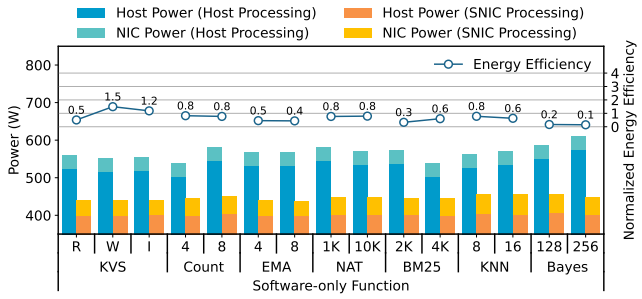
NVIDIA BF-3 vs. Intel Sapphire Rapids. The latest BF-3 provides 2$\times$ more CPU cores and 3.5$\times$ higher memory bandwidth than BF-2. However, our preliminary evaluation shows that BF-3 still faces the same challenges as BF-2, because of two reasons. (1) Its networking speed is doubled (*i.e.*, 200Gbps), and (2) the newest Intel Xeon Scalable Processor (Sapphire Rapids (SPR)) has also scaled the number of CPU cores and memory bandwidth in a similar degree, with more accelerators. In this work we first run the software-only functions on the BF-3 CPU, and then compare the maximum throughput, p99 latency, and system-wide energy efficiency of the BF-3 CPU with those of the SPR CPU. Fig. 10 shows that the BF-3 CPU provides up to 80% lower throughput and 61$\times$ higher p99 latency than the SPR CPU. Note that the throughput of the BF-3 CPU is similar to that of the SPR CPU for lightweight functions such as Count and NAT. However, this can be misleading because our setup is limited to 100Gbps for the clients, causing network bandwidth to saturate before the compute capability of the BF-3 CPU and the SPR CPU becomes the bottleneck for such functions. The SPR CPU also offers up to $\sim$80% higher system-wide energy efficiency, which is dominated by throughput (§III-B). Given this, we expect HAL to still be applicable to the latest generations of the SNIC processors and the host processors as a significant performance gap persists.

Lastly, more performant SNIC processors can be built, but they need to meet the cost and power/thermal constraints imposed by the SNIC market and market-driven form factor. An excellent example of a more performant processor/accelerator is the GPU. It is far more performant than the host processor (for certain applications such as ML/AI), but it is also far more expensive and power-hungry (*e.g.*, NVIDIA H100 GPU [54]).

Impact of SNIC processor's DVFS on the effectiveness of `LBP`. `LBP` is designed to dynamically assess the application-dependent processing capability of the SNIC processor at runtime by monitoring the Rx queue occupancy ($\texttt{RxQ}_{\texttt{Occ}}$). The voltage/frequency (V/F) state-dependent processing capability affects $\texttt{RxQ}_{\texttt{Occ}}$, similar to the application-dependent processing capability. Therefore, `LPB` should be as effective for the SNIC processor deploying dynamic V/F scaling (DVFS). Nonetheless, we find that the benefit of the SNIC processor with DVFS is limited because the power consumption of the SNIC



(a) Maximum throughput and p99 latency



(b) Average power consumption and energy efficiency

Fig. 10: Comparison of maximum throughput, p99 latency, average power consumption and energy efficiency between the BF-3 CPU and the Sapphire Rapids CPU.

processor contributes only a small fraction to the system-wide power consumption (§III-B). For instance, assuming the idle power consumption for SNIC processing at any rate in Table V, we estimate that deploying DVFS will reduce the system-wide power consumption by only 2% at most.

Executing complementary functions between the SNIC processor and the host processor. We may have the SNIC processor execute one function while making the host processor process other complementary functions. This may eliminate the need for load-balancing between the SNIC processor and the host processor. However, even the SNIC accelerators are unable to process the packets at the NIC line rate and provide unacceptable p99 latency at high packet rates. For example, we have shown that the REM accelerator began to drop packets and gave unacceptably high p99 latency when the packet rate exceeds ∼40Gbps (Fig. 9). This is because the BF-2's line rate is 100Gbps, but the maximum throughput of the BF-2 accelerators is 50Gbps at most (§III-A). This still demands load balancing between the host processor and the SNIC processor to handle all packets at any packet rate without increasing the p99 latency.

Comparison with FPGA-based SNIC. Although FPGA-based SNICs may be able to accelerate more diverse functions, users are required to design hardware for a given application at the register transfer level (RTL) or with a high-level synthesis (HLS) flow. The hardware design process, *i.e.*, coding at the RTL, synthesizing the RTL code, and verifying the synthesized design, can be time-consuming for most users. The HLS flow can partially reduce design complexity for users, but it often produces suboptimal hardware designs [16], [41].

IX. Related Work

Offloading Load Balancer to Programmable Network Devices. Turbo [65] and RingLeader [43] have focused on offloading load balancing among host CPU cores using an FPGA-based NIC and an SNIC, respectively. iPipe [46] has provided a software-based load balancer running on the SNIC CPU to selectively offload compute agents from the host CPU to the SNIC CPU, based on measured tail latency. However, it has used a 10Gbps/25Gbps SNIC for applications demanding ms-scale latency. P4Mite [70] has proposed load balancing between the host CPU and different PCIe-attached devices (*e.g.*, GPU and FPGA) using a programmable switch. In contrast, HAL aims for load balancing between the SNIC processor and the host processor for applications requiring μs-scale latency using the latest 100Gbps SNIC.

SNIC Characterization. The increasing significance of deploying SNIC in datacenters has led to extensive research, some of which aims to better understand the performance and energy efficiency of SNICs in datacenters [23], [44], [47], [74], [78]. They have focused on different performance metrics, energy analyses, and strategies for efficient utilization of SNICs. E3 [47] has measured throughput, latency, and energy consumption across various microservices on a 10Gbps SNIC. Wei *et al.* have characterized the performance of the host-SNIC and intra-SNIC communication paths [74]. Huang *et al.* have compared the maximum throughput, p99 latency, and system-wide energy efficiency of the SNIC processor with those of the host CPU [23]. In contrast, our work focuses on understanding the performance of a host-SNIC system, including accelerators, and energy efficiency at the system level. Furthermore, it considers the host accelerators and demonstrates that the SNIC processor can still improve system-wide energy efficiency under p99 latency constraints when the packet rate is low.

X. Conclusion

In this work, first, we have comprehensively compared the capability of the SNIC processor with that of the host processor. Then we have demonstrated that SNIC processor can improve system-wide energy efficiency under latency constraints only when the packet rate is low. To address such a limitation, we have proposed HAL to support seamless SNIC-host cooperative computing with a hardware-based load balancer and load balancing policy inside an SNIC. Lastly, we have presented that HAL improves system-wide energy efficiency and throughput by 31% and 10%, respectively, without compromising p99 latency.

Acknowledgment

This work was supported in part by grants from the IBM-Illinois Discovery Accelerator Institute (IIDAI), NSF (CNS-2238665), and AMD-Xilinx XACC.

APPENDIX

A. Abstract

In this section, we provide instructions for reproducing key experimental results presented in the paper. There are three key sets of experiments: (1) experiments that evaluate the throughput, p99 latency, power consumption, and energy efficiency of a server using the host processor and the SNIC processor, respectively, for processing ten functions (Fig. 2, Fig. 3, and Fig. 4); (2) experiments that evaluate the impact of software-based load balancing on throughput and p99 latency (Fig. 5); (3) experiments that show benefits realized by using hardware-assisted load balancing (HAL) (Fig. 9).

B. Artifact Check-list (Meta-information)

- **Program:** OpenSSL [57], QAT_Engine [26], QATzip [24], dpdk-test-compress-perf [13], RXPBench [53], and ipmitool [11].
- **Compilation:** gcc 9.4, and Xilinx Vivado 2023.1.
- **Data set:**
  − REM: `snort_literals` and `teakettle_2500` rulesets [32].
  − Compression: `Silesia-mozilla` file [36].
- **Run-time environment:** Ubuntu 20.04.5 LTS with DOCA 1.5.0.
- **Hardware:** A server with Intel Xeon Gold 6140 CPU and 100Gbps NVIDIA BlueFiled-2 SmartNIC, a client with Intel Xeon E5-2660 v4 CPU and 100Gbps ConnectX-6 Dx NIC, a Xilinx Alveo U280 FPGA, an Intel c62x QAT adapter, and custom power measurement setup: Yocto-Watt power sensor and a PCIe riser card.
- **Metrics:** Throughput, p99 latency, power consumption, and energy efficiency.
- **Output:** `.txt` files store experimental results for each figure.
- **Experiments:** The host and/or SNIC processors run functions while receiving incoming network packets from a client.
- **How much disk space required (approximately)?:** 200GB.
- **How much time is needed to complete experiments (approximately)?:** 4 hours.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT license
- **Archived (provide DOI)?:** 10.5281/zenodo.11040381

C. Description

1) How to access: The source code, scripts, and instructions can be accessed on GitHub (https://github.com/ece-fast-lab/ISCA-2024-HAL). The other open-source benchmarks we evaluated are publicly available on:

- **OpenSSL:** https://github.com/openssl/openssl
- **QAT_Engine:** https://github.com/intel/QAT_Engine
- **QATzip:** https://github.com/intel/QATzip
- **dpdk-test-compress-perf:** https://github.com/DPDK/dpdk
- **RXPBench:** https://developer.nvidia.com/networking/doca
- **ipmitool:** https://github.com/ipmitool/ipmitool

2) Hardware dependencies: The experimental setup requires a 100Gbps network connection between a client equipped with a DPDK-compatible 100Gbps NIC and a server equipped with an NVIDIA BlueField-2 SNIC and an Intel c62x QAT adapter. The server should be compatible with the Intelligent Platform Management Interface (IPMI). Power consumption is monitored through IPMI and a PCIe riser card with Yocto-Watt power sensor for isolated SNIC power measurement. For hardware-assisted load balancing experiments, a Xilinx Alveo U280 FPGA is used, with one network port connected to the client's NIC and the other to the server's SNIC.

3) Software dependencies: DOCA 1.5.0 is required for SNIC. Xilinx Vivado 2023.1 is utilized for FPGA compilation and programming. The Intel QAT driver HW Version 1.X is used for the host QAT accelerator.

D. Installation

First, clone the artifact repository as follows:

```
$ git clone
    https://github.com/ece-fast-lab/ISCA-2024-HAL
```

Then, follow the instructions to install the software dependencies in `software_install.md`, and compile and program the FPGA by referring to `1_fpga_src/README.md`. Finally, in `4_script/common/env_setup.sh`, modify user account names and IP addresses according to the client, server, and SNIC information. Set up the environment and compile the code as follows:

```
$ source 4_script/common/env_setup.sh
$ bash 4_script/common/compile.sh
```

E. Experiment workflow

Follow `4_script/quick/README.md` to run experiments and draw figures for all the figures together with custom `<run_name>`, and experiment results are stored in `4_script/fig*/result<run_name>`:

```
$ cd 4_script/quick_run
$ bash run_all_fig.sh <run_name>
$ bash process_results_all_fig.sh <run_name>
$ bash draw_all_fig.sh <run_name>
```

Afterwards, view generated figures:

```
$ feh ../fig*/fig*.png
```

To make sure stable results, the experiments should be repeated multiple times, and the median values from these runs should be used to create the figures.

F. Evaluation and expected results

Our experiments compare the host processor, SNIC processor and SNIC-host cooperative computing. The host processor can outperform the SNIC processor for most functions at high packet rates. The SNIC processor can improve system-wide energy efficiency over the host processor only at low packet rates for most functions under p99 latency constraints. The experiments also reveal that software-based load balancing incurs penalties in throughput and p99 latency. For NAT and REM (Fig. 9), the hardware-assisted load balancing (HAL) can offer 11%–27% lower power than the host processor while meeting throughput and p99 latency requirements.

REFERENCES

[1] B. Acun, P. Miller, and L. V. Kale, "Variation Among Processors Under Turbo Boost in HPC Systems," in *Proceedings of the International Conference on Supercomputing (ICS)*, 2016.

[2] M. Ahn, A. Chang, D. Lee, J. Gim, J. Kim, J. Jung, O. Rebholz, V. Pham, K. Malladi, and Y. S. Ki, "Enabling CXL Memory Expansion for In-Memory Database Management Systems," in *Proceedings of the International Workshop on Data Management on New Hardware (DaMoN)*, 2022.

[3] Andy Rudoff, "Compute Express Link™ (CXL™) 3.0: Expanded capabilities for increasing scale and optimizing resource utilization," https://www.snia.org/sites/default/files/cmss/2023/SNIA-CMSS23-Rudoff-CXL-Expanded-Capabilities.pdf.

[4] Archlinux, "CPU Frequency Scaling," https://wiki.archlinux.org/title/CPU_frequency_scaling.

[5] Arm Corporation, "Introducing the AMBA Coherent Hub Interface," https://developer.arm.com/documentation/102407/0100/Introducing-the-AMBA-Coherent-Hub-Interface.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 40, no. 1, pp. 92–99, 2010.

[7] Chips and Cheese, "Hot Chips 34 – Intel's Meteor Lake Chiplets, Compared to AMD's," https://chipsandcheese.com/2022/09/10/hot-chips-34-intels-meteor-lake-chiplets-compared-to-amds/.

[8] T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[9] CXL Consortium, "Compute Express Link (CXL)," https://www.computeexpresslink.org.

[10] J. Dastidar, D. Riddoch, J. Moore, S. Pope, and J. Wesselkamper, "AMD 400G Adaptive SmartNIC SoC: Technology Preview," in *Proceedings of the IEEE Hot Chips 34 Symposium (HCS)*, 2022.

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

[12] DPDK, "DPDK Programmer's Guide: Power Management," https://doc.dpdk.org/guides/prog_guide/power_man.html.

[13] DPDK, "dpdk-test-compress-perf Tool," https://doc.dpdk.org/guides/tools/comp_perf.html.

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

[15] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, "Advanced Encryption Standard (AES)," https://doi.org/10.6028/NIST.FIPS.197.

[16] H. Eran, L. Zeno, Z. István, and M. Silberstein, "Design Patterns for Code Reuse in HLS Packet Processing Pipelines," in *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[17] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2011.

[18] A. Forencich, A. C. Snoeren, G. Porter, and G. Papen, "Corundum: An Open-Source 100-Gbps NIC," in *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020.

[19] H. Golestani, A. Mirhosseini, and T. F. Wenisch, "Software Data Planes: You Can't Always Spin to Win," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2019.

[20] W. Gomes, S. Morgan, B. Phelps, T. Wilson, and E. Hallnor, "Meteor Lake and Arrow Lake Intel Next-Gen 3D Client Architecture Platform with Foveros," in *Proceedings of the IEEE Hot Chips 34 Symposium (HCS)*, 2022.

[21] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020.

[22] P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 1998.

[23] J. Huang, J. Lou, Y. Sun, T. Wang, E. K. Lee, and N. S. Kim, "Making Sense of Using a SmartNIC to Reduce Datacenter Tax from SLO and TCO Perspectives," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2023.

[24] Intel, "Compression Library accelerated by Intel® QuickAssist Technology," https://github.com/intel/QATzip.

[25] Intel, "Data Plane Development Kit (DPDK)," https://www.dpdk.org.

[26] Intel, "Intel QuickAssist Technology (QAT) OpenSSL Engine," https://github.com/intel/QAT_Engine.

[27] Intel, "Intel Xeon Processor Scalable Family Technical Overview," https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html.

[28] 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.

[29] Intel, "Intel® Intelligent Storage Acceleration Library," https://github.com/intel/isa-l.

[30] Intel, "Intel® Intelligent Storage Acceleration Library Crypto Version," https://github.com/intel/isa-l_crypto.

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

[32] Intel, "Introduction to Hyperscan," https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-hyperscan.html.

[33] Intel, "Optimizing Computer Applications for Latency: Part 1: Configuring the Hardware," https://www.intel.com/content/www/us/en/developer/articles/technical/optimizing-computer-applications-for-latency-part-1-configuring-the-hardware.html.

[34] H. Ji, M. Mansi, Y. Sun, 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 the USENIX Annual Technical Conference (ATC)*, 2023.

[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. Karandikar, A. N. Udipi, J. Choi, J. Whangbo, J. Zhao, S. Kanev, E. Lim, J. Alakuijala, V. Madduri, Y. S. Shao, B. Nikolic, K. Asanovic, and P. Ranganathan, "CDPU: Co-Designing Compression and Decompression Processing Units for Hyperscale Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2023.

[37] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire Jr, "Metron: NFV Service Chains at the True Speed of the Underlying Hardware," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[38] F. Klinker, "Exponential Moving Average Versus Moving Exponential Average," *Mathematische Semesterberichte*, 2011.

[39] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[40] Q.-T. V. Le, "Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) Performance Under a Virtual Machine," https://www.intel.com/content/www/us/en/developer/articles/technical/intel-intelligent-storage-acceleration-library-intel-isa-l-performance-under-a-virtual.html.

[41] J. Li, Z. Sun, J. Yan, X. Yang, Y. Jiang, and W. Quan, "DrawerPipe: A Reconfigurable Pipeline for Network Processing on FPGA-Based SmartNIC," *Electronics*, vol. 9, no. 1, 2020.

[42] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky, "SILT: A Memory-efficient, High-performance Key-Value Store," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[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 USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

[44] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance Characteristics of the BlueField-2 SmartNIC," *arXiv preprint arXiv:2105.06619*, 2021.

[45] J. Liu, Z. Tian, P. Liu, J. Jiang, and Z. Li, "An Approach of Semantic Web Service Classification based on Naive Bayes," in *Proceedings of the IEEE International Conference on Services Computing (SCC)*, 2016.

[46] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading Distributed Applications onto SmartNICs Using IPipe," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019.

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

[48] U. M. Maurer and S. Wolf, "The Diffie–Hellman Protocol," *Designs, Codes and Cryptography*, vol. 19, no. 2-3, pp. 147–171, 2000.

[49] Netronome, "Agilio CX SmartNICs - Netronome," https://www.netronome.com/products/agilio-cx/.

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

[51] 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.

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

[53] NVIDIA, "NVIDIA DOCA RXPBench User Guide," https://docs.nvidia.com/doca/archive/doca-v1.5.0/rxpbench/index.html.

[54] NVIDIA, "NVIDIA H100 Tensor Core GPU," https://www.nvidia.com/en-us/data-center/h100/.

[55] N. I. of Standards and T. (NIST), "Secure Hash Standard (SHS)," https://csrc.nist.gov/publications/detail/fips/180/4/final.

[56] N. I. of Standards and T. (NIST), "Digital Signature Standard (DSS)," https://csrc.nist.gov/publications/detail/fips/186/4/final, 2013.

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

[58] S. Oswal, A. Singh, and K. Kumari, "Deflate Compression Algorithm," *International Journal of Engineering Research and General Science*, vol. 4, no. 1, 2016.

[59] A. Ousterhout, J. Fried, J. Behrens, A. Belay, and H. Balakrishnan, "Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[60] 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 USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[61] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Commun. ACM*, vol. 21, no. 2, 1978.

[62] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," vol. 3, no. 4, 2009.

[63] H. N. Schuh, A. Krishnamurthy, D. Culler, L. Levy, Henry M.and Rizzo, S. Khan, and B. E. Stephens, "CC-NIC: a Cache-Coherent Interface to the NIC," in *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

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

[65] H. Seyedroudbari, S. Vanavasam, and A. Daglis, "Turbo: SmartNIC-enabled Dynamic Load Balancing of µs-scale RPCs," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[66] S. A. Siewior, R. Russell, S. Vaddagiri, A. Raj, J. Schopp, and T. Gleixner, "CPU hotplug in the Kernel — The Linux Kernel documentation," https://docs.kernel.org/core-api/cpu_hotplug.html.

[67] C. Song, M. J. Kim, T. Wang, H. Ji, J. Huang, I. Jeong, J. Park, H. Nam, M. Wi, J. H. Ahn, and N. S. Kim, "TAROT: A CXL SmartNIC-Based Defense Against Multi-bit Errors by Row-Hammer Attacks," in *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[68] S. Sun, R. Zhang, M. Yan, and J. Wu, "SKV: A SmartNIC-Offloaded Distributed Key-Value Store," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, 2022.

[69] Supermicro, "Supermicro Intelligent Management: On-board BMC Specification," https://www.supermicro.com/en/solutions/management-software/bmc-resources.

[70] H. Tajbakhsh, R. Parizotto, M. Neves, A. Schaeffer-Filho, and I. Haque, "Accelerator-Aware In-Network Load Balancing for Improved Application Performance," in *Proceedings of the International Federation for Information Processing Networking Conference (IFIP Networking)*, 2022.

[71] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A SmartNIC-Driven Accelerator-Centric Architecture for Network Servers," in *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[72] 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 USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

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

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

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

[76] Xilinx, "Alveo U280 Data Center Accelerator Card," https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.

[77] Xilinx Customer Community, "100G Ethernet Subsystem Latency," https://support.xilinx.com/s/question/0D52E00006ihQZfSAM/100g-ethernet-subsystem-latency.

[78] 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 ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*, 2022.

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

[80] Y. Yuan, Y. Wang, R. Wang, and J. Huang, "HALO: Accelerating flow classification for scalable packet processing in NFV," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2019.

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

[82] 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 USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.