IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 July 2024, accepted 28 July 2024, date of publication 1 August 2024, date of current version 13 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3437203

SURVEY

A Comprehensive Survey on SmartNICs: Architectures, Development Models, Applications, and Research Directions

ELIE F. KFOURY, (Member, IEEE), SAMIA CHOUEIRI, ALI MAZLOUM, (Graduate Student Member, IEEE), ALI ALSABEH, JOSE GOMEZ, AND JORGE CRICHIGNO, (Member, IEEE)
College of Engineering and Computing, University of South Carolina, Columbia, SC 29201, USA

Corresponding author: Elie F. Kfoury (ekfoury@email.sc.edu)

This work was supported by the National Science Foundation (NSF), Office of Advanced Cyberinfrastructure (OAC), under Grant 2118311, Grant 2403360, and Grant 2346726.

ABSTRACT The end of Moore's Law and Dennard Scaling has slowed processor improvements in the past decade. While multi-core processors have improved performance, they are limited by the application's level of parallelism, as prescribed by Amdahl's Law. This has led to the emergence of domain-specific processors that specialize in a narrow range of functions. Smart Network Interface Cards (SmartNICs) can be seen as a revolutionary technology that combines heterogeneous domain-specific processors and general-purpose cores to offload infrastructure tasks. Despite the impressive advantages of SmartNICs and their importance in modern networks, the literature has been missing a comprehensive survey. To this end, this paper provides a background encompassing an overview of the evolution of NICs from basic to SmartNICs, describing their architectures, development environments, and advantages over legacy NICs. The paper then presents a comprehensive taxonomy of applications offloaded to SmartNICs, covering network, security, storage, and compute functions. Challenges associated with SmartNIC development and deployment are discussed, along with current initiatives and open research issues.

INDEX TERMS SmartNIC, data processing unit (DPU), infrastructure processing unit (IPU), Moore's law, application offloading, P4, application specific integrated circuit (ASIC), field programmable gate array (FPGA).

I. INTRODUCTION

In 1965, Gordon Moore predicted that the number of transistors per chip would double every year [1], which was updated in 1975 to every two years [2]. In 1974, Robert Dennard noted that power density was constant for a given silicon area even as the number of transistors increased because of the smaller dimensions of each transistor. Transistors used less power and the performance of integrated circuits was enhanced by packing more transistors per chip [3]. The ability of the microprocessor, or simply processor, to exploit the advances in integrated circuits enabled impressive performance improvements, see Fig. 1 [4]. Unfortunately, in 2003, the limits of power due to the end of Dennard Scaling slowed processor performance to 23%. This observation forced the industry to use multiple processors per chip, referred to as cores. While multi-core processors helped improve performance, they have slowed down in the last decade, because of the natural limits prescribed by Amdahl's Law [5]: there is a maximum performance benefit from parallelism, as applications also have tasks that must be executed sequentially. Additionally, Moore's law has recently ended, resulting in the improvements of processors to slow down further.

The associate editor coordinating the review of this manuscript and approving it for publication was Rentao Gu.

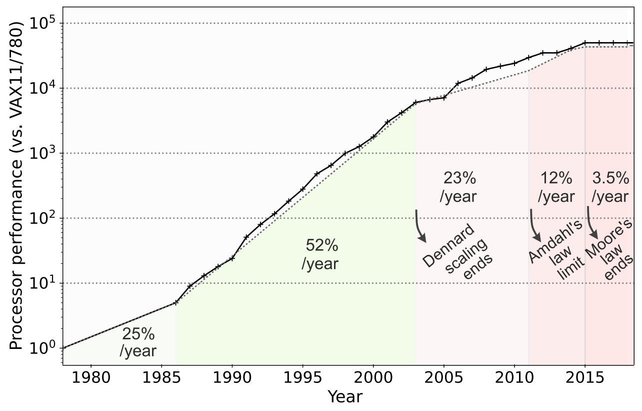


FIGURE 1. **Growth in processor performance over 40 years, relative to the VAX 11/780 as measured by the SPEC integer benchmarks. Reproduced from [4].**

In today's world, most data arrive at compute locations as packets from the networks. The traditional communication channel connecting networks and hosts is the Network Interface Card (NIC). In the past, NICs were simple hardware-based devices that received the packets from the network and placed them in memory at the host. Packets would then wait for processing time by the general-purpose processor at the host [6]. Although this model was successful for a long time, it has several challenges in current environments:

- As Moore's law and Denning scaling ended, simply adding more processing capacity to cope with the increasing amount of traffic is no longer an option.
- A large percentage of tasks executed by the processors relate to the infrastructure rather than to the user applications, e.g., TCP/IP tasks, encryption, compression, etc. Such operations use valuable processor cycles that may be used for application tasks instead.
- Historical software solutions for packet-related tasks are not efficient in terms of throughput, latency, and energy. While in the past inefficient software solutions were mitigated by the relentless progress of the (hardware) processors, today's solutions can no longer rely on future improvements in processor performance.
- The explosion of network traffic is accompanied by impressive improvement in the physical layer and bandwidth capacity. As network traffic arrives at servers at higher rates, processors are unable to process it on time, and the gap between processor performance and bandwidth is only increasing.

Since the Dennard Scaling ended and the energy budget is no longer increasing, many consider that the only path left to improve energy, performance, and cost is by using domain-specific processors rather than power-hungry general-purpose processors. SmartNICs can be seen as a revolutionary technology developed to address the challenges listed above by combining heterogeneous domain-specific processors that specialize in a narrow range of infrastructure tasks. These include compression/decompression processors, programmable pipelines, encryption/decryption processors, and others. SmartNICs also include general-purpose processors which are used for managing the system, aiding the domain-specific processors, and enabling users to run control-plane applications. In the context of SmartNICs, the terms accelerators and engines are also used to refer to domain-specific processors [7], [8]. Note that the development of domain-specific processors has been successfully used in several domains, including graphics in the 2000s – Graphics Processing Units (GPUs)–, machine learning in mid 2010s –Tensor Processor Units (TPUs)–, networking in the late 2010s -Network Processor Units (NPUs) that adhere to architecture models such as the Protocol Independent Switch Architecture (PISA)–, and genomics in 2018 [9], [10], [11].

The momentum of SmartNICs is reflected in the global Information Technology (IT) ecosystem. Hyperscalers such as Google, Amazon, and Microsoft are designing their own SmartNICs to run infrastructure functions and optimize revenue and performance [12], [13], [14]. Manufacturers such as Intel, NVIDIA, and AMD are emphasizing the development of SmartNICs for a broad market range, offering Systems on a Chip (SoCs) with programmable domain-specific processors for security, networks, storage, and telemetry [15]. Cloud systems such as the Monterey project are redefining cloud architectures by incorporating SmartNICs to run storage, networks, and security services, resulting in substantial improvement in performance while leaving more processor cycles for user applications [7], [8]. Research and education networks (RENs) such as the Energy Sciences Network (ESnet) -the high-performance network that carries traffic for the U.S. Department of Energy and research organizations- are upgrading their infrastructures with SmartNICs to enable data-intensive science [16]. Software vendors are also offloading their solutions to SmartNICs; VMware's ESXi, vCenter, and NSX -integral components for virtualizing High Performance Computing (HPC) environments- can now be effectively offloaded onto SmartNICs [17]. Palo Alto Networks, a leading Next-Generation Firewall (NGFW) vendor, introduced the "Intelligent Traffic Offload" service [18]; this service offloads firewall functions to SmartNICs. Juniper Networks' virtual router/firewall can also be offloaded to SmartNICs [19]. Telecommunication operators are increasingly migrating their core services to run on SmartNICs [20]. Serverless and edge computing workloads, including Machine Learning (ML) training and inference, can be accelerated using SmartNICs [21], [22]. Testbeds such as FABRIC [23] and GEANT [24], used worldwide for fundamental research, rely on SmartNICs and other programmable devices to allow experimenters to program the data path behavior and process network traffic in novel ways at line rate [25], [26].

A. PAPER CONTRIBUTIONS

Despite the increasing interest in SmartNICs, prior research has only partially covered this technology. As shown in

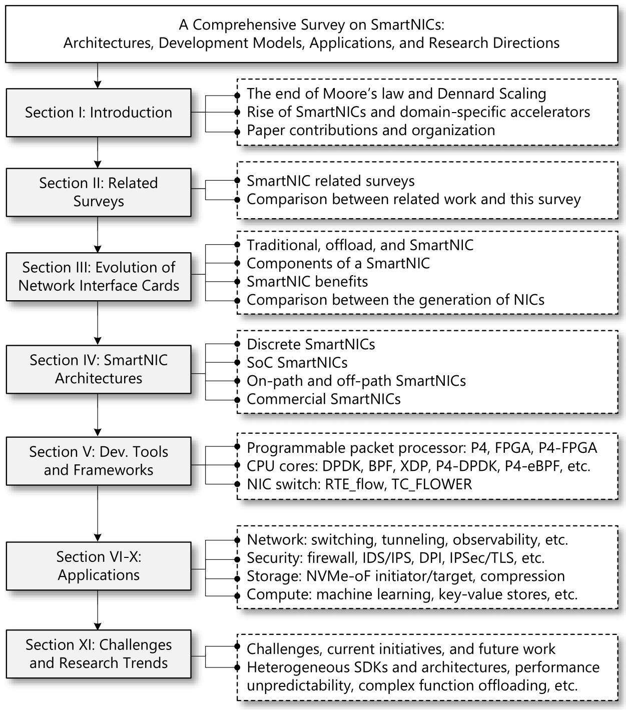

A Comprehensive Survey on SmartNICs:
Architectures, Development Models, Applications, and Research Directions
Section I: Introduction
The end of Moore's law and Dennard Scaling
Rise of SmartNICs and domain-specific accelerators
Paper contributions and organization
Section II: Related Surveys
SmartNIC related surveys
Comparison between related work and this survey
Section III: Evolution of Network Interface Cards
Traditional, offload, and SmartNIC
Components of a SmartNIC
SmartNIC benefits
Comparison between the generation of NICs
Section IV: SmartNIC Architectures
Discrete SmartNICs
SoC SmartNICs
On-path and off-path SmartNICs
Commercial SmartNICs
Section V: Dev. Tools and Frameworks
Programmable packet processor: P4, FPGA, P4-FPGA
CPU cores: DPDK, BPF, XDP, P4-DPDK, P4-eBPF, etc.
NIC switch: RTE_flow, TC_FLOWER
Section VI-X: Applications
Network: switching, tunneling, observability, etc.
Security: firewall, IDS/IPS, DPI, IPSec/TLS, etc.
Storage: NVMe-oF initiator/target, compression
Compute: machine learning, key-value stores, etc.
Section XI: Challenges and Research Trends
Challenges, current initiatives, and future work
Heterogeneous SDKs and architectures, performance unpredictability, complex function offloading, etc.

FIGURE 2. **Paper roadmap.**

Table 1, there is currently no updated and comprehensive material on SmartNICs. This paper addresses this gap by providing an overview of the evolution of NICs, starting from traditional basic NICs to SmartNICs. It describes the hardware architectures, technologies, and software development environments used with SmartNICs, as well as the advantages that SmartNICs offer over legacy NICs. The paper then proposes a taxonomy of the functions and applications being offloaded to SmartNICs, illustrating their advantages over the conventional method of executing such applications. Additionally, the paper discusses the challenges associated with SmartNICs and concludes by discussing future perspectives and open research issues.

B. PAPER ORGANIZATION

The road map of this survey is depicted in Fig. 2. Section II compares existing surveys on SmartNICs and related technologies and demonstrates the novelty of this work. Section III presents an overview of the evolution of NICs, from traditional basic NICs to SmartNICs. It describes the components of SmartNICs and their benefits compared to legacy NICs. Section IV describes the SmartNICs hardware architectures. Section V describes the tools, frameworks, and development environments for SmartNICs, both open-source and vendor-specific. Section VI provides a taxonomy of the applications and infrastructure workloads that are offloaded to SmartNICs. The subsequent sections (Sections VII-X) describe the security, network, storage, and compute functions. Section XI lists challenges associated with SmartNICs. It then discusses current initiatives that overcome the challenges and provides a reflection on open research issues. Section XII concludes the paper. The abbreviations used in this article are summarized in Table 12, at the end of the article.

II. RELATED SURVEYS

Despite the widespread interest from both industry and academia in SmartNICs, there is a noticeable absence of a comprehensive survey that adequately explores their potential and ongoing research endeavors. The existing surveys that are closest to this paper can be divided into 1) packet processing acceleration; and 2) programmable data planes.

A. SURVEYS ON PACKET PROCESSING ACCELERATION

The existing surveys in this category discuss the advantages of accelerating packet processing, particularly with software technologies. However, while SmartNICs are occasionally mentioned in these surveys, they fail to delve into crucial aspects such as their potential, architectures, applications, etc.

Cerović et al. [27] discuss various software-based and hardware-based packet accelerators. The survey focuses on server-class networking. It first starts by explaining the problems associated with using the standard Linux kernel for packet processing in high-speed networks and then delves into exploring the different classes of packet accelerators. For the software-based packet accelerators, the survey mainly describes and analyzes Data Plane Development Kit (DPDK) [28], PF_RING [29], NetSlices [30], and Netmap [31]. For the hardware-based packet accelerators, it focuses mainly on leveraging GPUs and Field Programmable Gate Arrays (FPGAs) for optimized and efficient packet processing. The survey does not cover the latest generation of SmartNICs that include CPU cores and domain-specific accelerators. Also, the survey does not cover the applications or the infrastructure workloads that can be offloaded to SmartNICs.

Freitas et al. [32] describe multiple packet processing acceleration techniques. The survey also focuses on packet processing in Linux environments. It categorizes the packet processing acceleration into hardware, software, and virtualization-based. For each category, the survey offers background information and discusses a simple use case. The survey also provides discussions on the host resource usage efficiency, the high packet rate, the system security, and the flexibility/expandability. The survey briefly mentions programmable NICs (another term used for SmartNICs) and their role in accelerating packet processing. It does not cover their development environments, hardware architectures, and the applications/workloads that can be offloaded.

Linguaglossa et al. [33] focus on software and hardware technologies that accelerate Network Function Virtualization (NFV). It categorizes software acceleration technologies into pure software acceleration and hardware-supported functions in software. It also provides a brief overview of the software acceleration ecosystem which includes DPDK, XDP, Netmap, and PF_RING. For the hardware technologies, it discusses the offloading functions of traditional NICs (e.g.,

TABLE 1. Comparison with related surveys.

Paper	Evolution and definition	Architectures and models	Development environments	Applications and offloaded workloads taxonomy	Comparisons with regular NICs	Challenges and discussions	Research trends and directions
[27]	○	⦿	⦿	○	○	⦿	⦿
[32]	○	⦿	⦿	○	○	⦿	⦿
[33]	⦿	⦿	⦿	⦿	○	⦿	⦿
[34]	○	⦿	⦿	⦿	○	⦿	⦿
[35]	⦿	⦿	⦿	⦿	⦿	⦿	⦿
[36]	○	○	⦿	⦿	○	⦿	⦿
[37]	⦿	⦿	⦿	⦿	○	⦿	⦿
This survey	●	●	●	●	●	●	●

● Covered in this survey ○ Not covered in this survey ⦿ Partially covered in this survey

CRC calculation, checksum computation, and TCP Offload Engine (TOE)) and a subset of the hardware architectures of SmartNICs. Then, it provides a brief overview of the programming abstractions in SmartNICs. The survey has the following limitations: 1) it does not cover all the hardware architectures; 2) it does not cover the development tools and environments; and 3) it does not cover the applications and infrastructure workloads that can be offloaded to SmartNICs.

Fei et al. [34] also focus on NFV acceleration. The survey classifies NVF acceleration into three high-level categories: computation, communication, and traffic steering. Under the computation category, the survey discusses some hardware offloading architectures which include SmartNICs. The remaining of the survey focuses on software acceleration and how to tune the system to achieve better performance. The survey has the following limitations: 1) it does not cover the hardware architectures used by the latest generation of SmartNICs; 2) it does not cover the development tools and environments; 3) it does not cover the applications and workloads that can be offloaded to the SmartNIC.

Shantharama et al. [35] provide a comprehensive survey on softwarized NFs. The survey classifies the CPU, the memory, and the interconnects as the three main enabling technologies for NFVs. With low-level details, the survey explains how each class operates and how it can be optimized to provide better virtualization support. It also discusses the use of dedicated hardware accelerators (FPGAs, ASICs, etc.) to improve the performance of softwarized NFs. The survey briefly describes some of the applications offloaded to SmartNICS without providing a sufficient overview of the technology, the different available development environments, or the latest enhancement in the field of SmartNICs.

Vieira et al. [36] only focus on the extended Berkeley Packet Filter (eBPF) and the eXpress Data Path (XDP) software acceleration techniques. The survey illustrates the process of enhancing packet processing speed by running eBPF-based applications in the XDP layer of the Linux kernel network stack. It presents a tutorial that includes the compilation and verification processes, the program structure, the required tools, and walk-through example programs. Although the authors mentioned SmartNICs as a target platform for eBPF applications, it does not cover the available architectures of SmartNICs, the development environments, or the applications that can be offloaded to SmartNICs.

Rosa et al. [37] describe multiple software and hardware techniques to enhance packet processing speed in the cloud. While discussing software-based techniques, the survey focuses on zero-copy data transfers, minimal context switching, and asynchronous processing as the core techniques for network acceleration. After that, it shows how DPDK, XDP, and eBPF are used in the cloud to enable Network Acceleration as a Service (NAaaS). While discussing hardware-based techniques, the survey only focuses on RDMA. The authors only describe SmartNICs as an enabling technology for RDMA and virtualization without describing their different architectures, development environments, of their different capabilities for enhancing network acceleration.

B. SURVEYS ON PROGRAMMABLE DATA PLANES

Numerous surveys have covered the general aspects of programmable data planes in the past few years [38], [39], [40], [41], [42]. Some surveys focused on specific areas such as network security [43], [44], [45], ML training and inference [46], [47], TCP enhancements [48], virtualization and cloud computing [49], 5G and telecommunications [50], rerouting and fast recovery [51], [52]. All these surveys have discussed some applications developed on SmartNICs. However, their focus is on programmable switches (e.g., Intel's Tofino). Recent advances in SmartNICs are not covered in these surveys.

C. NOVELTY

Table 1 summarizes the topics and the features described in the related surveys. It also highlights how this paper differs from the existing surveys. To the best of the authors' knowledge, this work is the first to exhaustively explore the whole SmartNIC ecosystem. Unlike previous surveys, this survey provides in-depth discussions on the evolution and definition of SmartNICs, the common architectures used by various SmartNIC models in the market, and the development environments (both open source and proprietary). It then provides a detailed taxonomy covering the applications

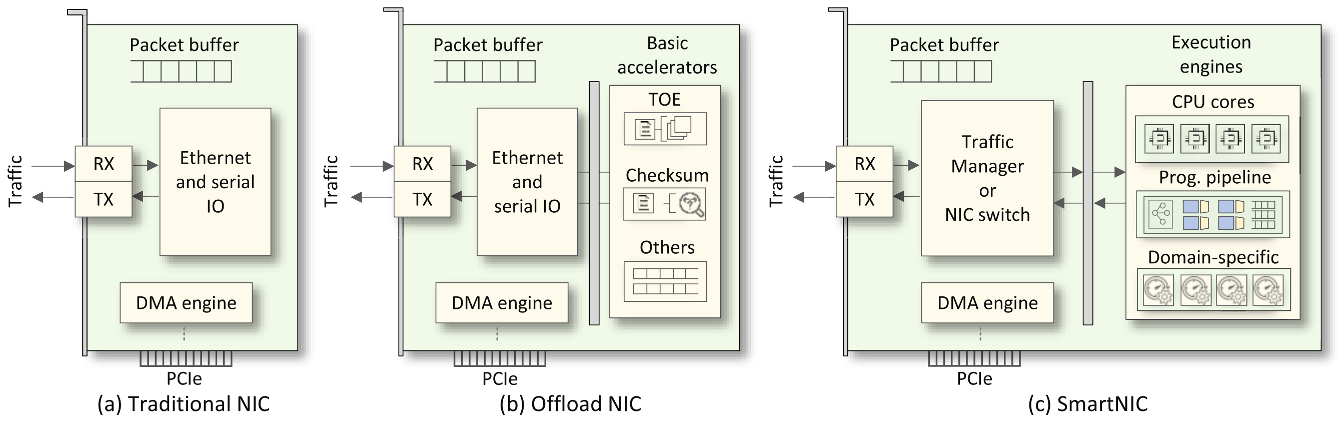


FIGURE 3. **Main components of (a) traditional NICs, (b) offload NICs, and (c) SmartNICs.**

that are offloaded to SmartNICs, while highlighting the performance gains compared to regular NICs. The survey also presents the challenges associated with programming and deploying SmartNICs, as well as the current and future research trends.

III. EVOLUTION OF NETWORK INTERFACE CARDS (NICs)

There are three main generations of NICs: traditional NICs, offload NICs, and SmartNICs. Fig. 3 shows a simplified diagram of the three NICs.

A. TRADITIONAL NICs

Traditional NICs (Fig. 3 (a)) are devices that implement basic physical and data-link layer services. These services include serializing/deserializing frames, managing link access, and providing error detection. Typically, these services are executed by a fixed-function component residing on a special-purpose chip within the NIC. On the sending side, the fixed-function component accepts a datagram created by the host, encapsulates it in a link-layer frame, and then transmits the frame into the communication link, following the link-access protocol. On the receiving side, the fixed-function component receives the frame and forwards it to the host via a Peripheral Component Interconnect Express (PCIe) card.

B. OFFLOAD NICs

Offload NICs (Fig. 3 (b)) incorporate hardware in the form of ASICs and/or FPGAs to execute basic "infrastructure" functions[1] that were previously handled by the host. The goal is to free up cycles in the main host's CPU for application (end-user) tasks rather than infrastructure tasks. Examples of such functions include:

- Basic packet processing: parsing and reassembling IP datagrams, computing IP checksum, encapsulating and de-encapsulating TCP segments.
- Managing TCP connections on the NIC: connection establishment, checksum and sequence number calculations, TOE, sliding window calculations for segment acknowledgment and congestion control, among others.
- Other functions that manipulate TCP/IP header fields to implement basic filtering and traffic classification.

Offload NICs allow end users to perform pre-programmed functions on the NIC. However, they do not support the creation and execution of custom applications directly on the NIC. Even with full transport layer offload, application protocols still need to be implemented on the host [9].

C. SmartNICs

The definition of a SmartNIC is not widely agreed upon. Traditionally, NICs that performed functions beyond basic packet processing were labeled as SmartNICs. Unless otherwise noted, the term SmartNIC in this survey refers to the latest generation of NICs, also known as SoC SmartNICs, Infrastructure Processing Units (IPUs),[2] Data Processing Units (DPUs), and Auxiliary Processing Units (xPUs).[3]

Fig. 3 (c) shows a simplified diagram of a SmartNIC. The SmartNIC includes a Traffic Manager (TM) or a NIC switch that performs Quality of Service (QoS) and steers traffic to the NIC execution engines. The NIC execution engines consist of a combination of processors used for custom packet processing and other domain-specific functions. Some SmartNICs (e.g., NIVDIA's BlueField-2 [53]) use a multi-core CPU processor for custom packet processing. Other SmartNICs (e.g., AMD Pensando DSC [54]) use embedded flow engines running a P4 programmable Application Specific Integrated Circuit (ASIC) pipeline. Other SmartNICs (e.g., AMD Xilinx SN1000 [55]) use an FPGA for custom packet processing. The domain-specific processors are optimized to provide high-performance and energy-efficient processing for a specific set of functions (e.g., cryptography). The execution

[1]In this context, infrastructure functions refer to tasks that facilitate data movement to the host and do not involve application data.

[2]IPU is the terminology used by Intel.

[3]xPU is used by the Storage Networking Industry Association (SNIA) community.

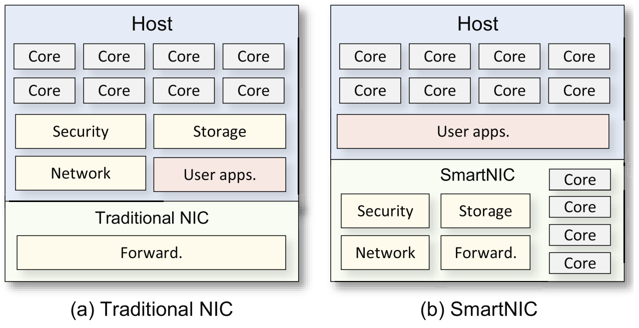


FIGURE 4. **In a deployment with a traditional NIC (a), the host CPU cores execute infrastructure functions and user applications. With SmartNICs (b), the host CPU cores solely execute user applications. The SmartNIC CPU cores assist other accelerators in executing infrastructure functions.**

engines have a memory hierarchy that typically consists of an L1 cache, scratchpad, L2 cache, and Dynamic RAM (DRAM).

The SmartNICs have general-purpose CPU cores for executing control plane functions. The CPU cores also enable SmartNICs to function autonomously and have their own Operating System (OS), such as Ubuntu Linux, which is independent of the host system in which they are running.

The programmable components of a SmartNIC allows it to execute infrastructure functions, without involving the CPU of the host. Consider Fig. 4. In a deployment with a traditional NIC (a), the host CPU cores execute infrastructure functions (typically classified as network, security, and storage) and user applications; with SmartNICs (b), the host CPU cores solely execute user applications. The SmartNIC CPU cores assist other domain-specific accelerators in executing infrastructure functions.

1) CUSTOM PACKET PROCESSING

SmartNICs enable the developers to devise custom packet processing on its execution engines. The packet processing logic can be implemented on CPU cores, FPGAs, or programmable ASIC pipelines. Regardless of the hardware architecture used by the SmartNIC, its packet processing engines include the following components: programmable parser, programmable match-action pipeline, and programmable deparser. These components closely resemble those of the PISA architecture [56], see Fig. 5. The *programmable parser* allows the developer to define headers based on custom or standard protocols and parse them. It is represented as a state machine. The *programmable match-action pipeline* carries out operations on packet headers and intermediate results. Each match-action stage comprises multiple memory blocks (such as tables and registers) and Arithmetic Logic Units (ALUs) that enable concurrent lookups and actions. To address data dependencies and ensure coherent processing, stages are organized sequentially. After processing the packet, the *programmable deparser* reconstructs packet headers and serializes them for transmission.

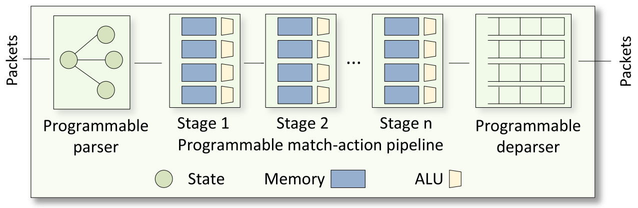


FIGURE 5. **Programmable Pipeline. The parser allows parsing custom packets. The match-action pipeline executes operations over the packet headers and intermediate results. The deparser assembles the packet headers back and serializes them for transmission.**

Although various vendors have their own models for programming the pipeline, there is a common goal across the industry to make them P4-programmable [57]. P4, originally designed as a domain-specific language for programmable data plane switches, has gained popularity in programming other packet data paths due to its simplicity and versatility.

2) DOMAIN-SPECIFIC PACKET PROCESSING

Infrastructure tasks can be broadly categorized into network functions, security functions, and storage functions. These tasks are integral to various networks, including data centers, cloud environments, enterprise networks, and campus networks. Given the specificity of these functions, some are optimized by being implemented directly in hardware to enhance their speed and efficiency. For instance, Transport Layer Security (TLS), a widely used protocol for encrypting application payloads and authenticating users, involves functions like encrypting and decrypting data. Recognizing the repetitive nature of these operations, it is practical to hardcode them into hardware. Hardware-based crypto processors, which have been utilized for some time, are examples of domain-specific processors incorporated into SmartNICs. In addition to improving the speed and efficiency of their respective functions, domain-specific processors free up CPU cores on the host for other computing tasks.

Other examples of domain-specific processors include regular expression (RegEx) used for tasks requiring Deep Packet Inspection (DPI), Non-Volatile Memory Host Controller over Fabrics (NVMe-oF) for remote storage, data compression, data deduplication, Remote Direct Memory Access (RDMA), etc. The application sections of this survey (Sections VII-X) will explore more specific use cases that leverage these domain-specific accelerators for various applications.

3) CONTROL PLANE AND MANAGEMENT

SmartNICs incorporate CPU cores for running control plane functions and for managing the SmartNIC. The CPU cores can also be used for implementing functions that do not fit in ASIC/FPGA execution engines. The CPU cores are typically ARM or MIPS-based. Some advantages of incorporating CPU cores within the SmartNIC are:

- Certain infrastructure functions (e.g., key distribution for TLS sessions) require execution in the CPU. The

TABLE 2. Features, Traditional NICs, Offload NICs, and SmartNICs.

Feature	Traditional NIC	Offload NIC	SmartNIC
Infrastructure functions separation	Low	Medium	High
Security Isolation	Low	Low	High
General-purpose CPU	No	No	Yes
Domain-specific processors	Low	Medium	High
Customization of data plane	No	Low	High
Flexibility to define new protocols	No	Low	High
Innovation	Low	Medium	High
Standardized models	Yes	Yes	No
Technology maturity	High	High	Medium

SmartNIC CPU cores can be utilized to perform these functions. This alleviates the burden on the host's CPU cores, allowing them to focus on executing user application functions.
- Infrastructure functions will run more efficiently on the CPU cores of the SmartNIC than on the CPU cores of the host since they will be separated from other compute-intensive workloads used by user applications.
- The security is improved because the infrastructure functions will be completely isolated from the host.
- The ASIC/FPGA execution engines have limitations on the complexity of operations to be performed on the packets. Such limitations stem from the fact that packets must be processed as quickly as possible to sustain line rate. Having CPU cores on the SmartNIC can help in executing such functions, at the cost of an increase in the latency.

D. SmartNICs BENEFITS

SmartNICs offer a wide range of features and benefits that solve modern network challenges.

- Infrastructure offloads: Data center infrastructure tasks currently consume up to 30% of processing capacity [58]. This phenomenon is commonly known as the *Data Center Tax*. By offloading these tasks to the SmartNIC, the freed-up 30% of processing capacity becomes available for user applications. This optimization can significantly increase revenue opportunities for cloud providers. This is the main reason why hyperscalers are among the early adopters of this technology.
- Application acceleration: By incorporating hardware-based accelerators, SmartNICs demonstrate superior performance per watt compared to host-based applications. This results in reduced latency, enhancing overall efficiency.
- Agility and reprogrammability: The process of developing new silicon is time-consuming, expensive, and requires thorough testing. By the time this cycle is completed, rapid technological advancements may have already rendered the hardware obsolete. SmartNICs address this challenge by offering programmable components, allowing for adaptability and timely updates in response to changing technological needs.
- Security isolation: SmartNICs enhance security by isolating the execution of infrastructure functions from the server execution environments.

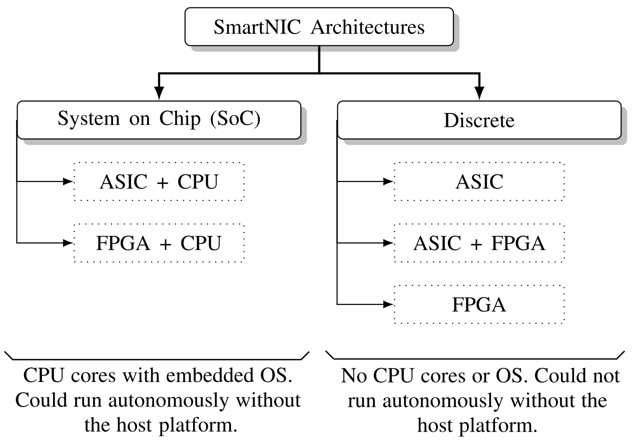

SmartNIC Architectures
System on Chip (SoC)
Discrete
ASIC + CPU
FPGA + CPU
ASIC
ASIC + FPGA
FPGA
CPU cores with embedded OS. Could run autonomously without the host platform.
No CPU cores or OS. Could not run autonomously without the host platform.

FIGURE 6. Taxonomy of SmartNIC architectures based on SoC and discrete categories. The taxonomy is reproduced from [59].

TABLE 3. Comparison between various SmartNIC architectures.

Architecture	Cost	Programming Complexity	Flexibility	Speed
ASIC	Low	Low	Low	High
FPGA	High	High	Medium	High
ASIC + FPGA	High	Medium	Medium	High
ASIC + CPU	Medium	Low	High	Medium
FPGA + CPU	High	High	High	Medium

E. COMPARISON OF TRADITIONAL, OFFLOAD, AND SmartNICs

Table 2 contrasts the main characteristics of traditional, offload, and SmartNICs. In the latter, the infrastructure functions are separated from the user applications; this isolation improves security by protecting the user applications on the host. The separation is possible due to the presence of CPU cores and domain-specific accelerators on the SmartNIC. Moreover, the data plane (i.e., packet processing) of the SmartNIC is customizable and is defined by the developer's code; this provides flexibility in defining and processing new protocols as well as innovating with new applications. The technology maturity and the standardized architectures for SmartNICs can still be considered low in contrast to traditional and offload NICs.

IV. SmartNICs ARCHITECTURES

The definition of a SmartNIC in Section III-C targeted SoC SmartNICs. SoC SmartNICs comprise computing units, which include a general-purpose ARM/MIPS multicore processor, packet processing engines, and domain-specific accelerators. It also includes a multi-level onboard memory hierarchy. The architectural components are connected by

TABLE 4. Commercial discrete SmartNICs from various vendors.

Architecture	Vendor	Model	PCIe generation	Bandwidth (Gbps)	Technical document
ASIC	Mellanox	ConnectX -5 / 6LX / 6DX / 7	3 / 4 / 5	50 / 100 / 200 / 400	[60]–[63]
FPGA	Achronix	VectorPath S7t	5	400	[64]
	AMD	Alveo U50 / U50 LV / U55C / U200 / U250 / U280	3 / 4	1x100 / 2x100	[65]–[68]
	Napatech	NT200A02	3	2x100	[69]
	Silicom	N501x / N5110A / FB2CDG1 / N6010/6011 / FB4XXVG TimeSync	3 / 4 / 5	4x25 / 2x100 / 4x100 / 2x400	[70]–[74]
ASIC+FPGA	NVIDIA	Innova -2 Flex	4	2x100	[75]

TABLE 5. Commercial SoC SmartNICs from various vendors.

Architecture	Vendor	Model	Core type	CPU cores	PCIe generation	Bandwidth (Gbps)	Technical document
ASIC+CPU	AMD Pensando	Giglio / DSC2-(25/100/200)	Arm A72	16	4	2x25 / 2x100 / 2x200	[54], [76], [77]
	Asterfusion	Helium EC2004Y / EC2002P	Arm V8	24	3 / 4	4x24 / 2x100	[78], [79]
	Broadcom	Stingray PS225-H16	Arm A72	8	3	2x25	[80]
	Intel / Google	E2000	Arm Neoverse N1	16	4	200	[81]
	Marvell	LiquidIO III	Arm A72	36	4	5x100	[82]
	Netronome	Agilio FX / CX	Arm A72	4	3	2x10 / 2x40	[83], [84]
	NVIDIA	BlueField 2 - 2X	Arm A72	8	4	200	[53]
	NVIDIA	BlueField 3 -3X	Arm A78	16	5	400	[85]
FPGA+CPU	AMD	Alveo U25N / U45N / SN1000	Arm A53/A42	4 / 16	3 / 4	2x25 / 1x100 / 2x100	[55], [86], [87]
	Intel	N6000-PL / N6001-PL / C6000X-PL / C5000X-PL	Arm A53 / Xeon D	4 / 8	3 / 4	2x25 / 2x100	[88]
	Napatech	NT400D1xSCC / F2070X IPU	Arm A53 / Xeon D	4 / 8	4	2x100	[89]

high-bandwidth coherent memory buses or high-performance interconnects [90].

Another category of SmartNICs, referred to as discrete SmartNICs, do not incorporate CPU cores and thus, cannot run autonomously without the host platform. Regardless of whether the SmartNIC is SoC or discrete, its packet processing logic may be implemented using ASIC or FPGA. Various SmartNICs available in the market may employ either of these hardware architectures or in some cases, a combination of both. The SmartNICs architecture taxonomy is shown in Fig. 6. Table 3 summarizes the differences between the various SmartNIC architectures, as described next.

A. DISCRETE SmartNICs

Hardware implementations come with tradeoffs in terms of cost, programming simplicity, and adaptability. Furthermore, the hardware needs to support a high degree of parallelism to achieve high-speed packet processing. The discrete architectures can be classified as:

- ASIC: An ASIC device offers cost-effectiveness and optimal price-performance, as it is designed for a specific application, providing high efficiency for predefined tasks. However, its flexibility is limited because it cannot be reprogrammed after manufacturing. ASIC-based SmartNICs feature a programmable pipeline that is relatively straightforward to configure, yet this programmability is constrained by predefined functions within the ASIC, leading to potential limitations in supporting certain workloads.
- FPGA: FPGA-based SmartNICs are exceptionally programmable. FPGAs consist of an array of programmable logic blocks and a hierarchy of reconfigurable interconnects allowing the blocks to be inter-wired. Common FPGA-based SmartNICs have hardened blocks for providing a range of functions (e.g., hardened blocks for the network interface, hardened blocks for floating point, etc.) [59]. Given sufficient time, effort, and expertise, FPGAs can efficiently accommodate nearly any functionality within the confines of their available gates. However, FPGAs are known for being challenging to program and can be costly. They require hardware description languages (HDLs) such as VHDL or Verilog, making the development process complex and time-consuming.
- ASIC + FPGA: Integrating both ASIC and FPGA within the SmartNIC presents a balanced solution. Common functions are efficiently executed on the ASIC, leveraging its ease of programmability compared to the FPGA. Functions that cannot be programmed on the ASIC will be implemented on the FPGA, providing flexibility, albeit with increased programming complexity. In other words, the FPGA acts as an assist to the ASIC. This design provides high packet processing speed but is costly due to the use of FPGA technology. The trade-off between flexibility, cost, and performance is critical

in choosing the appropriate hardware architecture for specific applications.

Table 4 shows some popular commercial discrete SmartNICs from various vendors and their specifications.

B. SoC SmartNICs

SoC SmartNICs use ASIC or FPGA for the majority of packet processing, with some packets being processed by general-purpose CPU cores. The two major SoC SmartNIC architectures are:

- ASIC + CPU: this architecture includes a set of programmable ASIC cores that handle most of the packet processing. The level of programmability is defined by what has been designed within the ASIC during the manufacturing process. The CPU cores, on the other hand, handle the packet processing functions that are not supported on the ASIC.
- FPGA + CPU: this architecture includes FPGA configurable logic for handling most of the packet processing. Unlike the ASIC-based SoC, FPGA enables more flexibility in programming. The level of programmability is limited by the resources available on the FPGA. The CPU cores in this architecture will typically implement functions that are difficult to be implemented on the FPGA.

General-purpose CPU cores can be programmed using familiar high-level programming languages such as C, C++, and Python. The flexibility of the system is greatly enhanced, allowing for the implementation of a wide range of programs, including those with complex features like loops and multiplications. This versatility is particularly challenging to achieve on ASIC or FPGA, which are better suited for specific or predefined tasks. While the CPU cores allow additional features on the NIC, functions executed on the CPU cores might not achieve line-rate performance. Additionally, the latency can vary based on the program structure and the execution path that the packet follows. Adding more complexity to the packet processing will further increase the latency. The trade-off here involves balancing the benefits of programmability and flexibility against the potential performance impacts.

In addition to processing a small subset of the packets, integrating general-purpose CPU cores improves the management of the SmartNIC. They also enable the SmartNIC to be independent of the host, run its own OS, manage its own resources, and perform tasks autonomously without relying on the host system for every operation. This reduces the load on the host CPU and provides isolation.

Table 5 shows some popular commercial SoC SmartNICs from various vendors and their specifications.

C. ON-PATH AND OFF-PATH SmartNICs

Another way to categorize the architectures of SmartNICs is based on how their NIC cores interact with network traffic. There are two categories: *on-path* and *off-path* [90].

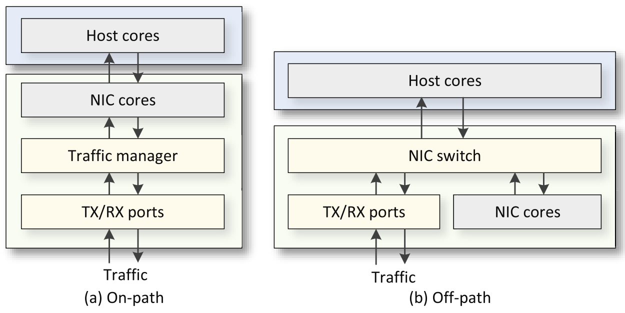


FIGURE 7. On-path and off-path SmartNICs. In on-path SmartNICs, all packets are processed by the NIC cores. In off-path SmartNICs, an embedded NIC switch directs the packets to be executed by the NIC cores or the host cores.

TABLE 6. Characteristics of on-path and off-path SmartNICs.

Characteristics	On-path	Off-path
NIC switch	×	✓
Operating system	×	✓
Full network stack	×	✓
Programming complexity	High	Low
Host performance impact	High	Low
Complex code offloading	Low	High

1) ON-PATH SmartNICs

With on-path SmartNICs (Fig. 7 (a)), the NIC cores actively manipulate each incoming and outgoing packet along the communication path. These SmartNICs provide low-level programmable interfaces, allowing for direct manipulation of raw packets. In this design, the offloaded code is closely situated to the network packets, increasing efficiency. However, the drawback is that the offloaded code competes for NIC cores with requests sent to the host. If too much computation is offloaded onto the SmartNIC, it can result in a significant performance degradation of regular networking requests sent to the host. Typical on-path SmartNICs are augmented with specialized hardware to enhance the packet processing on the cores. An example is *pre-fetching* in which the content of the packet is placed in a structure similar to the L1 cache [91].

Note that programming on-path NICs can be challenging due to the utilization of low-level APIs. Examples of on-path SmartNICs include Netronome Agilio [92], Marvell LiquidIO [82].

2) OFF-PATH SmartNICs

Off-path SmartNICs (Fig. 7 (b)) take a different approach by incorporating additional compute cores and memory in a separate SoC located next to the NIC cores. The offloaded code is strategically placed *off* the critical path of the network processing pipeline. The SoC is treated as a second full-fledged host with an exclusive network interface, connected to NIC cores and the host through an embedded switch (sometimes referred to as eSwitch or NIC switch). Based on forwarding rules installed on the embedded switch,

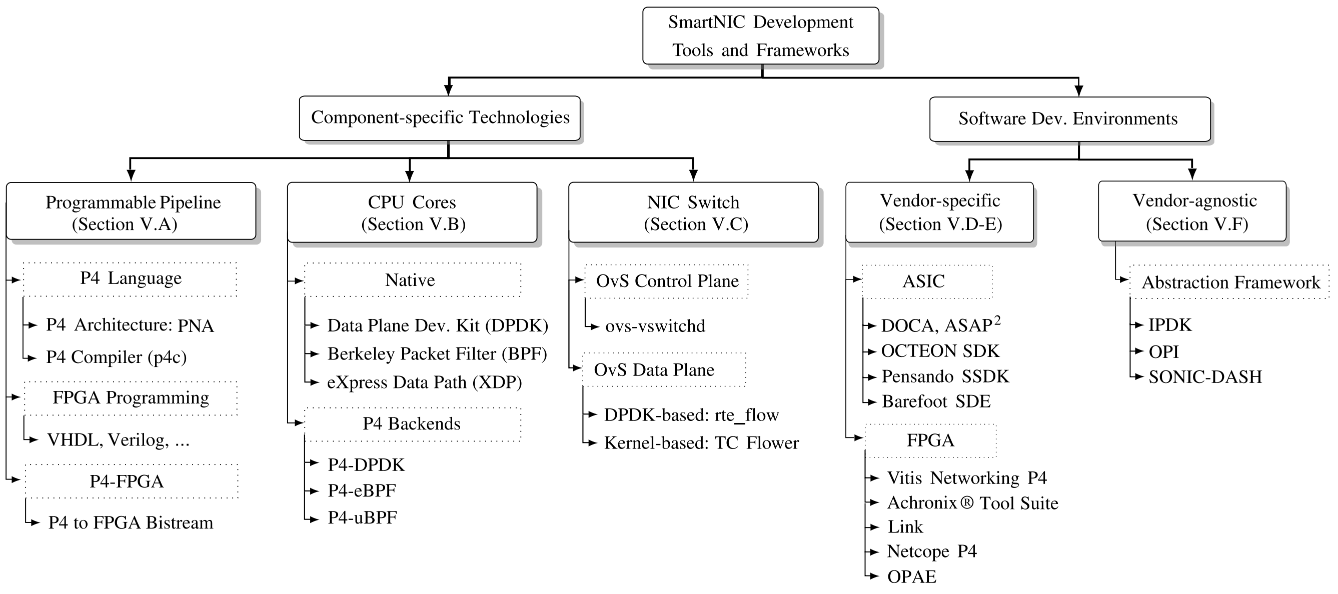


FIGURE 8. Taxonomy of SmartNIC development tools and frameworks, categorized by component-specific technologies and software development environments.

the traffic will be delivered to the host or the SmartNIC's CPU cores. Users can create a pipeline of match-action tables, serving as the fast path. The CPU cores on the other hand handle the complex logic and serve as the slow path. A common example is having the CPU cores process the first packet in a flow and generate rules for the match-action tables to process subsequent packets.

In contrast to on-path SmartNICs, the offloaded code in off-path SmartNICs does not impact the host's network performance. This clear separation enables the SoC to run a complete kernel (e.g., Linux) with a comprehensive network stack (RDMA), simplifying system development and allowing for the offloading of complex tasks. Examples of off-path SmartNICs include NVIDIA's BlueField-2 [53] and Broadcom Stingray [80].

Table 6 summarizes the differences between the on-path and off-path SmartNICs.

V. SmartNICs DEVELOPMENT TOOLS AND FRAMEWORKS

This section provides an overview of the development tools and frameworks employed for programming SmartNICs. The taxonomy, illustrated in Fig. 8, categorizes them based on the specific component within the SmartNIC being programmed.

A. PROGRAMMABLE PIPELINE

The packet processing logic is commonly built using ASICs or FPGAs. The development of offloaded applications depends on the hardware architecture and the vendor's Software Development Kits (SDKs).

1) P4 LANGUAGE

In 2016, the PISA architecture was introduced as a domain-specific processor for networking [56]. PISA is programmed using the Programming Protocol-independent Packet Processor (P4) language [93]. Although P4 was initially intended to program the data plane of PISA-based switches, it has demonstrated its versatility to program data planes for other packet processing devices. Despite the variety of programming models used by various vendors, there is a common goal which is to make their pipeline programmed in P4 [57].

P4 has a reduced instruction set and has the following goals:

- Reconfigurability: P4 enables the reconfiguration of the parser and the processing logic in the field.
- Protocol independence: P4 ensures that the device remains protocol-agnostic, allowing the programmer to define protocols, parsers, and operations for processing headers.
- Target independence: P4 hides the underlying hardware from the programmer, with the compiler considering the device's capabilities when transforming a target-independent P4 program into a target-dependent binary.

The initial specification of the P4 language, denoted as $P4_{14}$, was released in 2014 [94]. Subsequently, in 2016, a more refined version known as $P4_{16}$ was drafted [95]. $P4_{16}$ represents a matured language that extends the capabilities of P4 to a broader range of underlying targets, including ASICs, FPGAs, SmartNICs, and more.

Fig. 9 shows the workflow of developing a P4 program and deploying it into a target device. The P4 code is written by the user. The code must include a P4 architecture model, which is typically supplied by the device's manufacturer. The code is then compiled by a P4 compiler, which generates two artifacts: 1) the binary that will be deployed in the data plane of the target device; and 2) data plane APIs that will allow

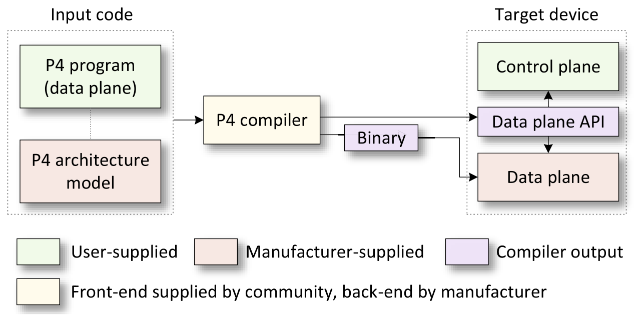


FIGURE 9. **P4 workflow. The P4 compiler accepts the user-supplied P4 code and the P4 architecture model. The output is a binary that will be loaded in the data plane, and APIs that the control plane can leverage to interact with the data plane.**

the control plane to interact with the data plane (e.g., for generating table entries, manipulate stateful memories, etc.).

2) P4 ARCHITECTURE

A P4 architecture is a programming model that defines the capabilities of a target's P4 processing pipeline. P4 programs are specifically designed for a particular P4 architecture, and these programs can be applied to any targets that adhere to the same P4 architecture.

Although the P4 architecture is provided by the manufacturer of the device, it often follows the specifications of open-source architectures. With the emergence of SmartNICs, the community has developed an open-source architecture tailored for programming these NICs. This architecture is the Portable NIC Architecture (PNA) [96].

a: PORTABLE NIC ARCHITECTURE (PNA)

PNA [96] is a P4 architecture that defines the structure and common capabilities for SmartNICs. PNA has four P4 programmable blocks (main parser, pre-control, main control, and main deparser), and several fixed-function blocks (e.g., network ports, packet queues, inline externs), as shown in Fig. 10. In contrast to the architectures used by programmable switches (e.g., PSA) which process packets only from network ports, PNA can process packets coming/going to

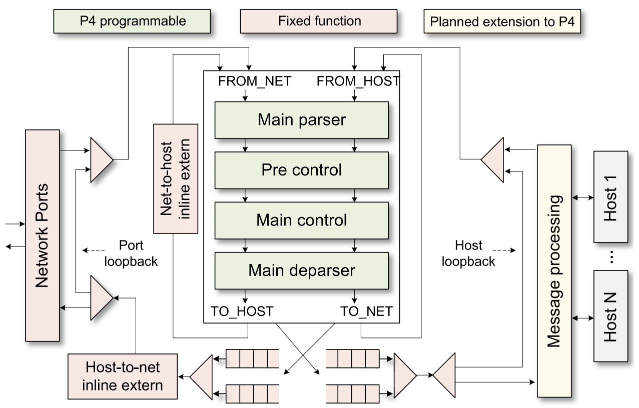


FIGURE 10. **Portable NIC Architecture (PNA).**

TABLE 7. **Comparison between P4 architectures: PSA and PNA.**

Feature	PSA	PNA
Main target devices	Switches	SmartNICs
Table entries modification	×	✓
Table accessibility	One stage	Multiple stages
Non-packet processing (message processing)	×	✓
Accelerator invocation	×	✓
Directionality (host-to-net, net-to-host)	×	✓
TCP connection tracking	×	✓
Stateful elements (counters, registers, meters, etc.)	✓	✓

network ports and to the host interfaces. Thus, there are four packet paths: net-to-host, host-to-net, net-to-net, and host-to-host.

Consider Fig. 10. The packets that arrive on a network port are processed by the MainParser. The MainParser parses network packets based on their headers. Once parsing is complete, the header fields are stored in data structures available throughout the entire packet processing pipeline. Next, the PreControl block decides whether packets need to be processed by the net-to-host extern block. One example is decrypting packet payloads using the IPsec protocol. If a packet has an IPsec header but fails to match any established security associations found through P4 table lookups, the PreControl code could drop the packet.

The MainControl block is where packet processing code is typically written. This code modifies headers, updates stateful elements like counters, meters, and registers, and optionally attaches additional user-defined metadata to the packet. The MainDeparser then serializes the headers back into a packet for further transmission. After this, the packet may either proceed to the NIC's message processing section and then typically to the host system or loop back towards the network ports. This design enables on-NIC processing of port-to-port packets without involving the host system.

The host-to-net and net-to-host externs allow executing functions on the domain-specific accelerators such as encrypting or decrypting IPsec payload. The message processing is responsible for converting between large messages in host memory and network size packets on the network and for dealing with one or more host operating systems, drivers, and/or message descriptor formats in host memory.

b: PNA FEATURES

PNA has features that are not traditionally supported by other similar P4 architectures including:

1) Table entries modification: Other P4 architectures only allow modifying table entries from the control plane. PNA allows modifying the entries in a table directly from the data plane. This is possible through the *add_on_miss* table property. When this property is set, the *add_entry* extern function can be invoked in the default action. This function will add a new entry in

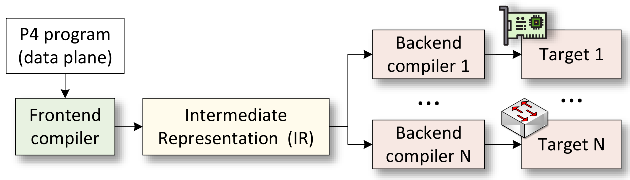


FIGURE 11. P4 compilation process.

the table with the same key that was used for matching and action data that are specified in the data plane. Note that the control plane can still modify the table entries, even if they were added by the data plane.

2) Table entry timeout: if the *pna_idle_timeout* property is set to a table, a notification will be sent once the configured duration has elapsed since an entry was last matched.
3) Table accessibility: traditional P4 architectures allow only one operation on a table per stage. With PNA, tables can be accessible by multiple stages, even in different pipelines.
4) Non-packet processing: PNA facilitates message processing, enabling operations on larger blocks of data to be transferred to and from host memory.
5) Accelerator invocation: PNA supports invoking accelerators (e.g., crypto accelerator).

Table 7 compares and contrasts PNA and the Portable Switch Architecture (PSA), an architecture mainly used by switches.

3) P4 COMPILER

After writing a P4 program, the programmer invokes the compiler to generate a binary that will be deployed on the target device (e.g., the programmable pipeline of the SmartNIC). Consider Fig. 11. The P4 compiler (p4c) has a frontend and a backend. The frontend is universal across all targets and handles the parsing, syntactic analysis, and target-independent semantic analysis of the program. The frontend generates an Intermediate Representation (IR) which is then compiled by the backend compiler for a specific target. The backend is provided by the manufacturer of the device.

4) FPGA PROGRAMMING

FPGAs consist of an array of configurable logic blocks and programmable interconnects, allowing users to define the functionality of the chip based on their application requirements. FPGA-based SmartNICs follow the same programming workflows as other FPGAs provided by the vendors. This means that the development tools, methodologies, and languages used for programming traditional FPGAs can be applied to SmartNICs as well. FPGA vendors provide software tools that facilitate the programming process. These tools include Integrated Development Environments (IDEs) and compilers that translate Hardware Description Languages (HDLs) such as VHDL and Verilog into configuration files for the FPGA.

5) P4-FPGA

Programming FPGAs with languages such as VHDL or Verilog can be challenging and time-consuming, especially for newcomers. To address this issue, frameworks have been developed to translate P4 code into FPGA bitstream. P4, being a high-level and user-friendly language ideal for programming datapaths, offers a faster and more efficient alternative for FPGA programming. This approach streamlines the programming process, making it particularly accessible for users without extensive FPGA programming expertise, ultimately enhancing both accessibility and efficiency. However, there are challenges in designing a compiler that translates P4 code to VHDL or Verilog. First, FPGAs are typically programmed using low-level libraries that are not portable across devices. Second, generating an efficient implementation from a source P4 program is difficult since programs vary widely and architectures make different tradeoffs.

The community has been actively working on developing P4 FPGA compilers. The vendors (e.g., Xilinx [97], Intel [98]) are providing their workflows to generate bitstreams from P4 on their targets. P4-FPGA tools can significantly reduce the engineering effort required to develop packet-processing systems based on devices while maintaining high performance per Lookup Table (LUT) or Random Access Memory (RAM).

B. CPU CORES

User applications run on CPU cores, whether on the cores on the SmartNIC, or the cores in the host. The steps for an application to process a packet coming from the NIC are shown in Fig. 12 (a). When a packet is received, the NIC triggers an interrupt that informs the OS about the packet's location in memory. The OS subsequently transfers the packet to the network stack which then initiates system calls from the OS kernel to deliver the packet to its intended user-level application. These steps induce overheads that dramatically degrade the bandwidth throughput. Today's NICs have already reached more than 200Gbps [18]. As NICs become faster, the available time for processing individual packets becomes increasingly limited. For instance, with 200Gbps, the time between consecutive 1500-byte packets is as low as 60 nanoseconds (ns). The standard network stack is inadequate to keep up with the high traffic rates.

1) DATA PLANE DEVELOPMENT KIT (DPDK)

DPDK comprises a collection of libraries and drivers designed to enhance packet processing efficiency by bypassing the kernel space and handling packets within user space (see Fig. 12 (b)). With DPDK, the ports of the NIC are disassociated from the kernel driver and associated with a DPDK-compatible driver. In contrast to the conventional

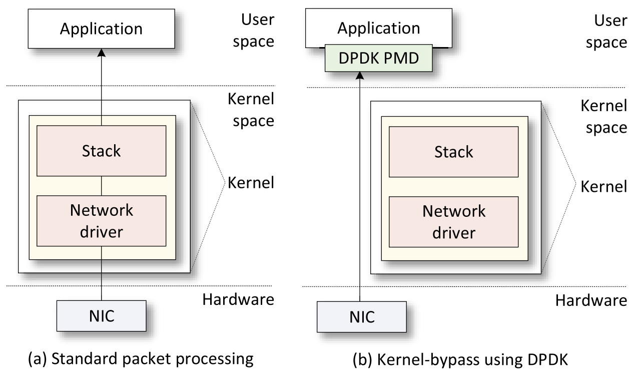


FIGURE 12. Software packet processing. (a) standard packet processing (interrupt-based), (b) kernel-bypass packet processing (polling mode).

method of packet processing within the kernel stack using interrupts, the DPDK driver operates as a Poll Mode Driver (PMD). It consistently polls for incoming packets. The utilization of a PMD, combined with the kernel bypass, yields superior packet processing performance. DPDK's APIs can be used in C programs.

DPDK started as a project by Intel and then became open source. Its community has been growing, and DPDK now supports all major CPU and NICs architectures from various vendors. A list of supported NICs can be found at [99].

2) eXpress DATA PATH (XDP) AND EXTENDED BERKELEY PACKET FILTER (eBPF)

When utilizing DPDK, the kernel is bypassed to achieve enhanced performance. However, this comes at the cost of losing access to networking functionalities provided by the kernel. User space applications are then required to re-implement these functionalities. XDP presents a solution to this issue. XDP operates as an eBPF program within the kernel's network code. It introduces an early hook in the RX (receive) path of the kernel, specifically within the NIC driver after interrupt processing. This early hook

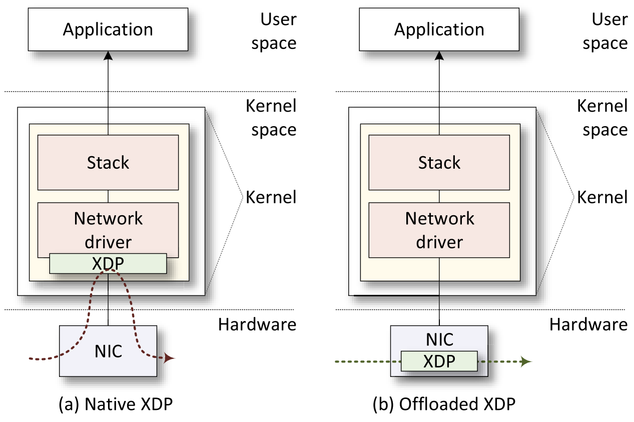


FIGURE 13. XDP packet processing. (a) Native XDP, slower, (b) Offloaded XDP, faster.

TABLE 8. Comparison between the P4 backends.

Features	P4-DPDK	P4-eBPF/XDP	P4-uBPF
Userspace	✓	×	✓
NIC support	✓	✓	✓
P4 Architectures	PNA, PSA	[ebpf,xdp] _model.p4	ubpf_model.p4
Compilation	P4⟶spec ⟶C⟶so	P4⟶C⟶ eBPF bytecode	P4⟶C⟶ uBPF bytecode
Supported Features	High	Low	Medium

allows the execution of a user-supplied eBPF program, enabling decisions to be made before the Linux networking stack code is executed. Decisions include dropping packets, passing packets to the normal network stack, and redirecting packets to other ports on the NIC. XDP reduces the kernel overhead and avoids process context switches, network layer processing, interrupts, etc.

XDP programs have callbacks that will be invoked when a packet is received on the NIC. There are three models for deploying an XDP program:

- Generic XDP: XDP programs are incorporated into the kernel within the regular network path. While this method does not deliver optimal performance advantages, it serves as a convenient means to experiment with XDP programs or deploy them on standard hardware that lacks dedicated support for XDP.
- Native XDP: The NIC driver loads the XDP program during its initial receive path, see Fig. 13 (a). Support from the NIC hardware is required for this mode.
- Offloaded XDP: The XDP program is loaded directly by the NIC hardware, bypassing the CPU as a whole, see Fig. 13 (b). This requires support from the NIC hardware.

3) P4 BACKENDS

Creating P4 programs is generally considered more straightforward compared to writing DPDK or BPF/XDP code. Consequently, there have been efforts to translate P4 into these codes. The P4 compiler (p4c) is equipped with backends specifically designed for generating DPDK, BPF/XDP, and Userspace BPF (uBPF) codes. Table 8 compares the P4 backends.

a: P4-DPDK

The p4c-dpdk backend translates $P4_{16}$ programs into DPDK Application Programming Interface (API), allowing the configuration of the DPDK software switch (SWX) pipeline [100]. The P4 programs can be written for the PNA

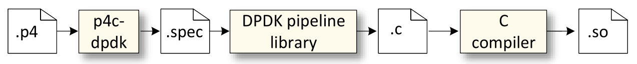


FIGURE 14. P4 DPDK pipeline.

or PSA architectures.[4] The backend transforms a given P4 program into a representation (.spec) that aligns with the DPDK SWX pipeline (see Fig. 14). The subsequent step involves the generation of a C code from the.spec file. This code includes C functions corresponding to each action and control block. A C compiler then generates a shared object (.so) from the C code. It is important to note that P4-DPDK is not a P4 simulator (e.g., BMv2); it achieves high performance.

b: P4-eBPF

The expressive powers of P4 and eBPF programming languages differ, yet there is a significant overlap, particularly in network packet processing. The P4 to eBPF compiler translates P4 programs into a restricted subset of C code that is compatible with eBPF. The P4 program only defines the data plane. The control plane is separately implemented; BPF Compiler Collection (BCC) tools simplify this by generating C and Python APIs for the interaction between the data plane and the control plane. The P4 to eBPF compiler also facilitates the integration of custom C extern functions, enabling developers to extend the P4 program's functionality by incorporating eBPF-compatible C functions. This capability empowers the P4 program with features not natively supported by the P4 language. Upon compilation, the P4 compiler generates a C file and its corresponding header. A subsequent C compiler then generates an eBPF program, loadable into the kernel using the Traffic Control (TC). Once loaded, manipulating tables in the eBPF program can be achieved with the bpftool provided by the kernel.

c: P4-uBPF

uBPF adapts the eBPF processor to run in userspace. The utilization of uBPF is advantageous due to its compatibility with any solution implementing kernel bypass, such as DPDK apps.

The p4c-ubpf compiler translates P4 programs into uBPF programs. The backend for uBPF predominantly relies on the P4-eBPF compiler, but generates C code compatible with user space BPF implementation. The uBPF backend offers a broader scope compared to the eBPF backend. Beyond simple packet filtering, the P4-uBPF compiler supports P4 registers and programmable actions, encompassing packet modifications and tunneling. The generated C programs are compiled by the clang compiler, which generates uBPF bytecode. The bytecode is then loaded into the uBPF VM.

C. NIC SWITCH

The NIC switch performs QoS traffic control and steers traffic to the NIC execution engines. SmartNICs typically implement the NIC switch following the specifications of the open-source Open vSwitch (OvS). OvS is a software switch originally designed to enable communication among Virtual

[4]Preliminary experiments show that more features are implemented for PNA over PSA for the P4-DPDK target.

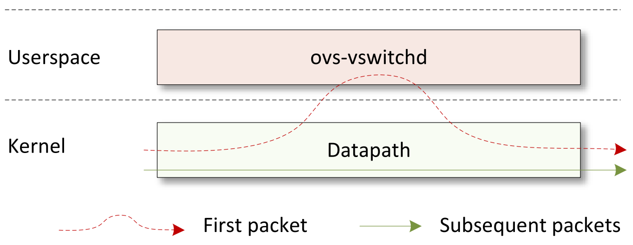


FIGURE 15. Open vSwitch (OvS) components.

Machines (VMs). OvS has two major components, the control plane (ovs-switchd) and the data plane, also known as the datapath.

1) OvS CONTROL PLANE

Fig. 15 demonstrates how the OvS components interact to forward packets. After receiving the first packet of a flow, the datapath forwards it to the ovs-switchd. The ovs-switchd then determines how the packet should be handled, and then sends the packet back to the data plane with the desired handling method. It also instructs the data plane to cache the action for handling similar packets. Subsequent packets are then matched and their actions are executed, all in the data plane. Actions may include packet modification, packet sampling, packet dropping, etc.

The OvS control plane is traditionally executed on the host in the userspace. With SmartNICs, the OvS control plane is executed on the CPU cores of the SmartNICs.

2) OvS DATA PLANE

The standard OvS switch's datapath is situated in the kernel (see Fig. 15), which strains CPU resources and degrades the performance. This performance degradation becomes more pronounced with an increasing number of flows and more complex policy rules, consuming multiple CPU cores for datapath operations and ultimately resulting in the lowest server utilization. To address these issues, many SmartNICs offer support for offloading OvS into their NIC switch. When this feature is utilized, the OvS datapath is moved to the hardware, resulting in superior performance compared to the software-based versions.

a: OvS-DPDK AND RTE_FLOW

OvS-DPDK enhances OvS by incorporating a DPDK-based datapath in the userspace, surpassing the performance of

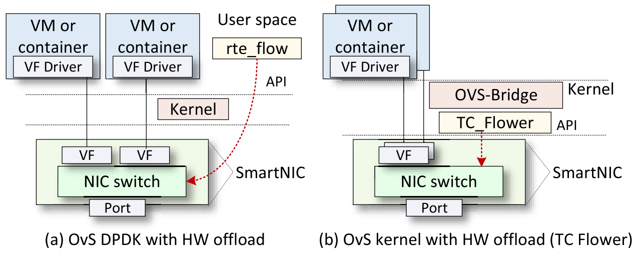


FIGURE 16. OvS hardware offload. (a) OvS DPDK with hardware offload using rte_flow; (b) OvS kernel with hardware offload using tc_flower.

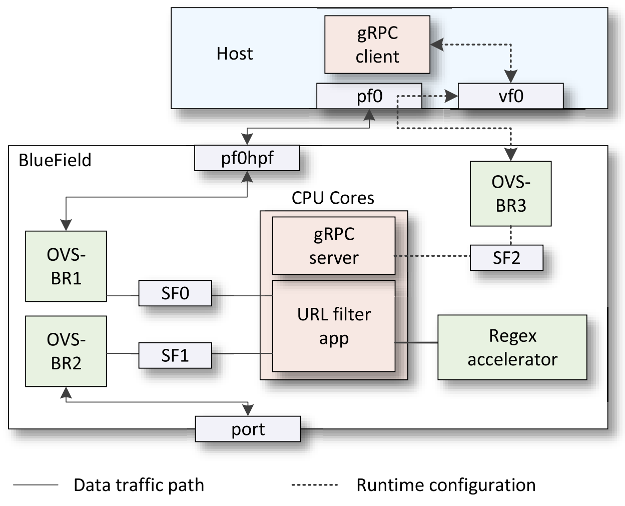


FIGURE 17. **DOCA URL filter reference application.**

the standard kernel OvS datapath with reduced latency. OvS-DPDK leverages hardware offload capabilities through rte_flow [101], a DPDK-based API, see Fig. 16 (a). This API facilitates the installation of rules into the hardware switch within the SmartNIC (NIC switch). The rte_flow API enables users to define rules for matching specific traffic, altering the packets, querying related counters, etc. Matching within this context can be based on various criteria such as packet data (including protocol headers and payload) and properties like associated physical port or virtual device function ID. Operations supported by the rte_flow API include dropping traffic, diverting traffic to specific queues, directing traffic to virtual or physical device functions or ports, performing tunnel offloads, and applying marks, among others.

b: OvS-KERNEL AND TC FLOWER

The OvS-kernel can use the TC Flower [102] to configure rules on the hardware switch integrated into the SmartNIC, see Fig. 16 (b). Within the Linux kernel, the TC flower classifier, which is a component of the TC subsystem, offers a means to specify packet matches using a defined flow key. This flow key encompasses fields extracted from packet fields and, if desired, tunnel metadata. TC actions enable the execution of diverse operations on packets, such as drop, modify, output, and various other functionalities.

D. VENDOR-SPECIFIC SDKs - ASIC

The following SDKs are proprietary and target ASIC-based SmartNICs.

1) NVIDIA's DOCA

The Data Center-on-a-Chip Architecture (DOCA), is a software development framework developed by NVIDIA for the BlueField SmartNICs [53]. This framework encompasses various components, including libraries, service agents, and reference applications. Applications developed using DOCA

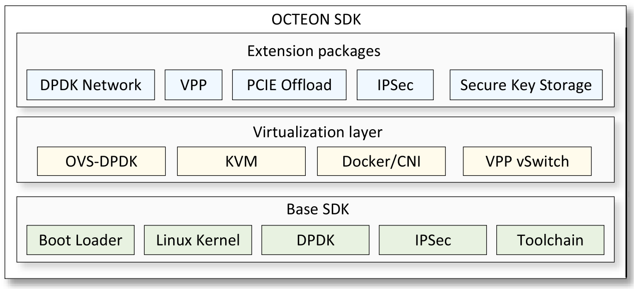


FIGURE 18. **OCTEON SDK layers and modules.**

are written in the C programming language and incorporate support for DPDK. This integration ensures that developers have access to all DPDK APIs for efficient packet processing. Additionally, DOCA comes equipped with its own set of libraries designed to streamline interactions with the components on the SmartNIC. For instance, to implement IPsec or perform encryption and decryption, DOCA offers dedicated APIs that developers can easily invoke, simplifying the integration of these functionalities into their applications.

One noteworthy library within DOCA is the DOCA Flow [103]. This library allows programmers to customize packet processing by defining matching criteria and actions. These match-action units are defined in pipes, which can be chained. Given DOCA's reliance on DPDK, it leverages rte_flow to transmit rules to the embedded switch (NIC switch). NVIDIA employs its proprietary ASAP^2 technology [104] for implementing the embedded switch and for efficient traffic offloading to the hardware.

Consider Fig. 17 which shows an example of a DOCA application for Uniform Resource Locator (URL) filtering. The developer must create OvS bridges and connect scalable functions (SF)[5] to them. Note that the OvS bridge is hardware offloaded. In this specific example, one bridge is used to connect the physical port to the application (OvS-BR2). Another bridge is used to connect the application to the host (OvS-BR1). The incoming packets on the physical port will be forwarded to the application, which runs on the CPU cores. URL filtering involves parsing the application layer because the URL to be visited is located at the HTTP header. The SmartNIC will invoke the regular expression (RegEx) hardware accelerator to scan for the URL, which is significantly faster than scanning using the CPU. A third bridge can be created to enable the user to manage the application (e.g., specifying the URLs to be blocked). BlueField provides gRPC interfaces for the runtime configuration.

It is possible to develop DOCA applications without the hardware; however, testing the compiled software must be done on top of a BlueField [105].

[5]An SF is a lightweight function that has dedicated queues for sending and receiving packets; it is analogous to the virtual function (VF) used in SR-IOV.

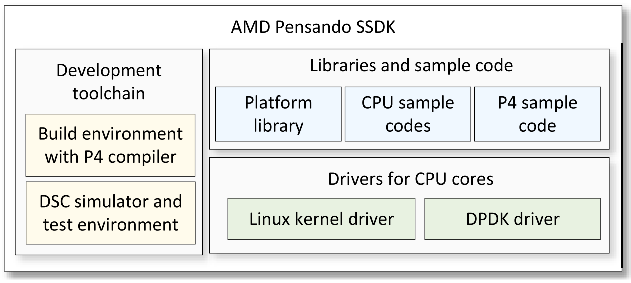


FIGURE 19. **AMD Pensando SSDK.**

2) OCTEON SDK

The OCTEON SDK is a comprehensive suite that integrates a development environment and optimized software modules for building applications on OCTEON family processors. The suite consists of a base SDK, a virtualization layer, and a collection of SDK extension packages designed for specific application functions. The Base SDK relies on a standard Linux environment and user-space DPDK (see Fig. 18). It facilitates the seamless compilation of DPDK, Linux, or control plane applications on top of it with minimal adjustments. Programmers write C code and invoke libraries for accelerating functions, including compression/decompression, regex matching, encryption/decryption, and more.

In addition to the Base SDK, the suite includes SDK extensions that help users enable complex applications. These extensions consist of pre-optimized, application-specific modules bundled into packages that run on the Base SDK. Notable extensions include OvS-DPDK, Vector Packet Processor (VPP), secure key storage, trusted execution environment, etc. Furthermore, the OCTEON SDK provides a cycle-accurate simulator. This simulator enables the developers to test the behavior of their programs with precision and accuracy in software.

3) AMD PENSANDO SSDK

The AMD Pensando SDK facilitates software development for the AMD Pensando SmartNIC. This comprehensive SDK includes a P4$_{16}$ compiler, debugging tools, a DPDK driver, example codes, and thorough documentation (see Fig. 19). Specifically, P4$_{16}$ can be used to write code for execution in the programmable pipeline. C and C++ are used to write code for the CPU core complex. Additionally, the SDK allows invoking the SmartNIC's built-in domain-specific accelerators.

Similar to DOCA, developers have the flexibility to compile applications without the SmartNIC hardware. However, unlike DOCA, Pensando SDK provides a simulator, allowing developers to test their ideas before uploading the image to the hardware. This validation capability becomes particularly advantageous when integrating the SDK and simulator into CI/CD-based development and workflows. The simulator boasts machine-register accuracy, ensuring that any code developed for it can be cross-compiled to run seamlessly on the real hardware. The simulator serves as a valuable tool for validation, speeding up development, and simplifying debugging processes within a virtualized environment.

The reference applications included with the AMD Pensando SDK include a basic skeleton hello world, Software Defined Network (SDN) policy offload with Longest Prefix Matching (LPM), Access Control List (ACL), flow aging, IPsec gateway, and other classic host offload such as TCP Segmentation Offload (TSO), checksum calculation, and Receive Side Scaling (RSS).

4) BAREFOOT SDE/INTEL P4 STUDIO

The compiler used for programming the pipeline on the Intel IPU has similarities with that used for programming the Tofino switches [81]. The compiler was originally developed by Barefoot Networks, which was acquired by Intel in 2021. This compiler was formerly known as Barefoot SDE, and now it has been rebranded as Intel P4 Studio. It is well-established and has undergone extensive revisions and optimizations. Additionally, the compiler is equipped with a Graphical User Interface (GUI) tool (P4 Insight [106]) that offers comprehensive insights into resource utilization. This includes details such as the location of specific match-action tables, the utilization of hash bits, and the usage of SRAM/TCAM. The public documentation does not provide clear specifics on how the compiler differs between Tofino switches and SmartNICs.

5) SDKs FOR ASIC SmartNICs COMPARISON

Table 9 compares the four SDKs. The characteristics compared include the supported SmartNIC models, P4 language support, development feasibility with or without dedicated hardware, availability of simulators or emulators for testing, and the necessity for special licensing. The AMD Pensando and Intel SmartNICs are P4 programmable and thus, their SDKs provide a P4 compiler. The NVIDIA BlueField and the Octeon SDK only support P4 for their CPU cores (e.g., through P4-DPDK). Furthermore, all SDKs except Intel/Barefoot SDE offer development without dedicated hardware, and Pensando SSDK and Octeon SDK provide simulators or emulators for testing purposes. The Pensando SSDK and the Intel SDE require the customer to sign a Non-disclose Agreement (NDA) to get the license for the SDKs.

E. VENDOR-SPECIFIC SDKS - FPGAs

The following SDKs are proprietary and target FPGA-based SmartNICs.

1) VITIS NETWORKING P4

Vitis Networking P4 [107], developed by AMD Xilinx, is the development environment for their FPGA SmartNICs. This high-level design environment greatly simplifies the creation of packet-processing data planes through P4 programs (see

TABLE 9. Comparison between the vendor-specific SDKs for ASIC SmartNICs.

Characteristic	NVIDIA DOCA	Octeon SDK	Pensando SSDK	Intel/Barefoot SDE
Supported SmartNICs	BlueField 2/3/X	Marvel LiquidIO	Pensando DSC-200	Intel IPU E2000
P4 support	×*	×*	✓	✓
Development wo/ hardware	✓	✓	✓	×
Simulator/ emulator	×	✓	✓	×
Special licensing	×	×	✓	✓

*While P4 is not the main language used for programming the packet processing engine, it can be used for programming the CPU cores (e.g., with P4 DPDK).

Section V-A5). The tool's primary function is to translate the P4 design intent into a comprehensive AMD FPGA design solution. The compiler maps the control flow with a custom data plane architecture composed of various engines. This process involves selecting suitable engine types and tailoring each one according to the specified P4 processing requirements. The architecture definition file for Vitis Networking P4 is named xsa.p4. This architecture follows the open-source P4 PNA architecture (see Section V-A2a).

Fig. 20 illustrates the AMD Vivado^{TM} hardware tool flows designed for AMD Vitis^{TM} Networking P4 implementations. There is a flow for the software, which is used for testing the behavior of the P4 program. The other flow is for the hardware. In the software flow, the P4C-vitisnet compiler accepts a P4 file as input and generates an output.json file. This json file is provided to the P4 Behavioral Model. Note that this behavioral model provided by Xilinx closely resembles the behavioral model created by the community, known as BMv2. The software flow also has a Command Line Interface (CLI) which is a control plane application used to interact with the data plane at runtime. Packets and metadata information can be fed into the behavioral model when testing.

For the hardware flow, the P4C-vitisnet compiler generates an.sv file. The compiler uses the Vitis Networking P4 IP. The.sv file can be used for launching Register Transfer Level (RTL) simulation on the RTL simulator or for running synthesis/implementation on the hardware. The hardware flow also accepts metadata and packets as inputs.

Xilinx also provides the Xilinx Runtime library (XRT) [108], which is a user-friendly, open-source software stack designed to facilitate communication between the application code and the FPGA device. It offers APIs for Python and C/C++ applications.

2) INTEL P4 SUITE FOR FPGA

The Intel P4 Suite for FPGA is a high-level design toolkit that produces packet processing IP for Intel-based FPGAs from P4 codes. This toolkit comprises a compiler responsible for converting P4 into RTL and a software framework equipped with APIs that facilitate the interaction between control plane applications and the data plane. The toolkit's workflow is shown in Fig. 21. Intel's P4 compiler for FPGA accepts as input the P4 program and a custom architecture, and generates RTL for the data plane and APIs for the control plane. The resulting RTL, combined with the architecture's RTL and the shell RTL are then transformed into an FPGA binary using conventional FPGA development environments (e.g., Intel Quartus). Finally, this binary is deployed to the hardware. The control plane APIs are pushed to the Intel P4 FPGA for Software Framework which enables user-defined control plane applications to interact with the hardware.

3) ACHRONIX TOOL SUITE

The Achronix Tool Suite (ACE) is used to design Achronix's FPGA SmartNICs. The tool includes place and route functions, timing analysis and bitstream generation and download, synthesis analysis, and in-system debugging using snapshots. The Achronix design flow facilitates ease for FPGA designers

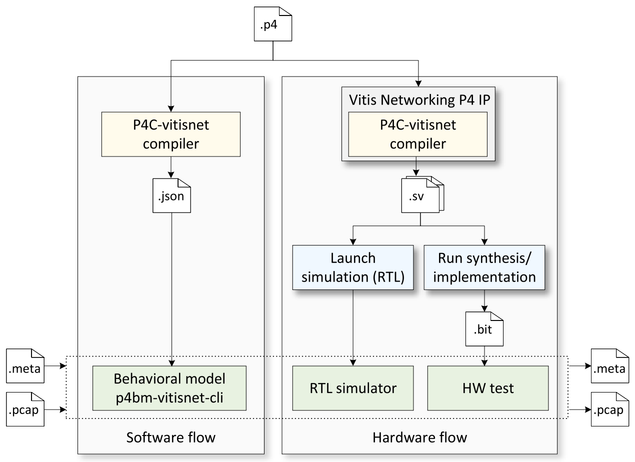


FIGURE 20. AMD Xilinx's Vitis Networking P4 software and hardware flows.

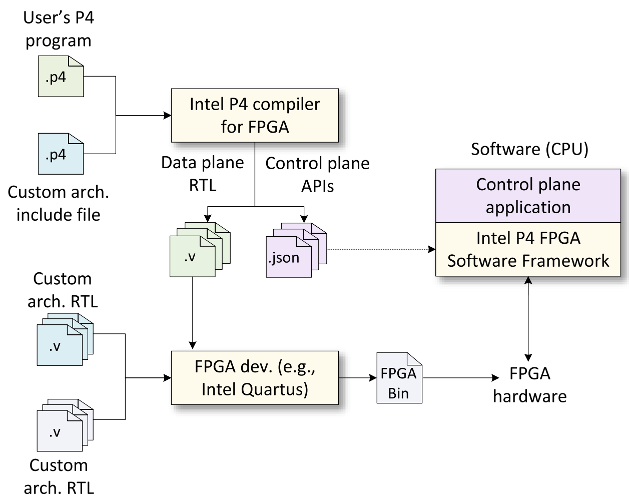

User's P4 program
.p4
.p4
Custom arch. include file
Intel P4 compiler for FPGA
Data plane RTL
Control plane APIs
.v
.json
Software (CPU)
Control plane application
Intel P4 FPGA Software Framework
Custom arch. RTL
.v
.v
Custom arch. RTL
FPGA dev. (e.g., Intel Quartus)
FPGA Bin
FPGA hardware

FIGURE 21. Intel P4 Suite for FPGA workflow.

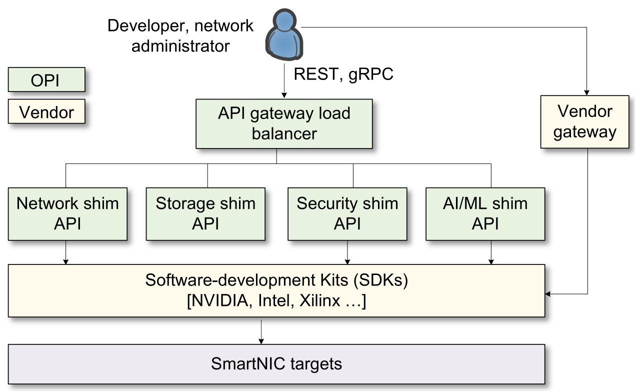

Developer, network administrator
REST, gRPC
OPI
Vendor
API gateway load balancer
Vendor gateway
Network shim API
Storage shim API
Security shim API
AI/ML shim API
Software-development Kits (SDKs) [NVIDIA, Intel, Xilinx ...]
SmartNIC targets

FIGURE 22. Open Programmable Infrastructure (OPI) architecture. The vendor-specific SDKs and hardware are abstracted via a common API developed by OPI.

by supporting standard RTL (VHDL and Verilog) input and employing industry-standard simulation techniques.

4) NAPATECH LINK TOOLKIT

The Link toolkit, developed by Napatech, is a collection of software providing plug-and-play features for various SmartNICs from Napatech. The software includes: Link-capture, which facilitates packet capture with nanosecond timestamping and replay with precise inter-frame gap control. Link-Inline accelerates various network and security applications, Link-Virtualization offloads a virtual switch to the SmartNIC, and Link-Storage accelerates virtualized storage.

5) OFS AND OPAE

The Open FPGA Stack (OFS) is an open-source solution that offers a hardware and software framework for creating shell design and workload [109]. OFS includes reference shell designs for various Intel FPGA devices, drivers, and software tools. Using OFS, Application-Specific FPGA Interface Managers (FIMs) can be developed, making use of the Open Programmable Acceleration Engine (OPAE) SDK. OPAE, which is a subset of OFS, is a software layer comprising various API libraries. These libraries are used when programming host applications that interact with the FPGA accelerator.

F. VENDOR-AGNOSTIC

1) OPEN PROGRAMMABLE INFRASTRUCTURE (OPI)

The OPI is a community-driven initiative focused on creating common APIs to configure and manage different SmartNIC targets [110]. Instead of relying on vendor-specific SDKs, developers can use OPI's standardized APIs to activate services, effectively abstracting the complexities associated with vendor-specific SDKs. Consider Fig. 22. The developer uses gRPC and REST APIs to initial calls to the API gateway. The gateway acts as a load balancer between four shim APIs: network, storage, security, and AI/ML. These shim APIs then translate the calls to the hardware accelerators through the vendor-specific SDKs. With such a design, portability can be ensured across various targets. Note that the developers can still execute functions provided by the vendor if they are not available through the OPI APIs.

2) INFRASTRUCTURE PROGRAMMER DEVELOPMENT KIT (IPDK)

The IPDK is an open-source, vendor-agnostic framework comprising drivers and APIs tailored for infrastructure offload and management tasks. It is versatile and capable of running on a range of hardware platforms including SmartNICs, CPUs, or switches. Operating within the Linux environment, IPDK leverages established tools like Storage Performance Development Kit (SPDK), DPDK, and P4 to facilitate network and storage virtualization, workload provisioning, root-of-trust establishment, and various offload capabilities inherent to the platform. IPDK is a sub-project of OPI.

IPDK already supports multiple targets including P4 DPDK, OCTEON SmartNICs, Intel IPU, Intel FPGA, and Tofino-based programmable switch [111].

IPDK has two main interfaces: 1) Infrastructure Application Interface; and 2) Target Abstraction Interface. The Infrastructure Application Interface serves as the northbound interface of the SmartNIC, encapsulating the diverse range of Remote Procedure Calls (RPCs) supported within IPDK. The Target Abstraction Interface represents an abstraction provided by an infrastructure device (e.g., SmartNIC) that runs infrastructure applications for connected compute instances. These instances could include attached hosts and/or VMs, which may or may not be containerized.

3) SONIC-DASH

SONiC, an open-source operating system for network devices, has experienced significant growth [112], [113]. The SONiC community has introduced a new open-source project called DASH (Disaggregated APIs for SONiC Hosts) aiming at being an abstraction framework for SmartNICs and other network devices. It consists of a set of APIs and object models which cover network services for the cloud. The initial objective of DASH is to enhance the performance and connection scale of SDN operations, aiming to achieve a speed increase of 10 to 100 times compared to software-based solutions in today's clouds and enterprise. DASH's ecosystem includes a community of cloud providers, hardware suppliers, and system solution providers.

VI. OFFLOADED APPLICATIONS TAXONOMY

This section describes the systematic methodology that was adopted to generate the proposed taxonomy. The results of this literature survey represent derived findings by thoroughly exploring the SmartNIC-related research works published in the last five years.

Fig. 23 shows the proposed taxonomy. The taxonomy was meticulously designed to cover the most significant works related to SmartNICs. The aim is to categorize the surveyed

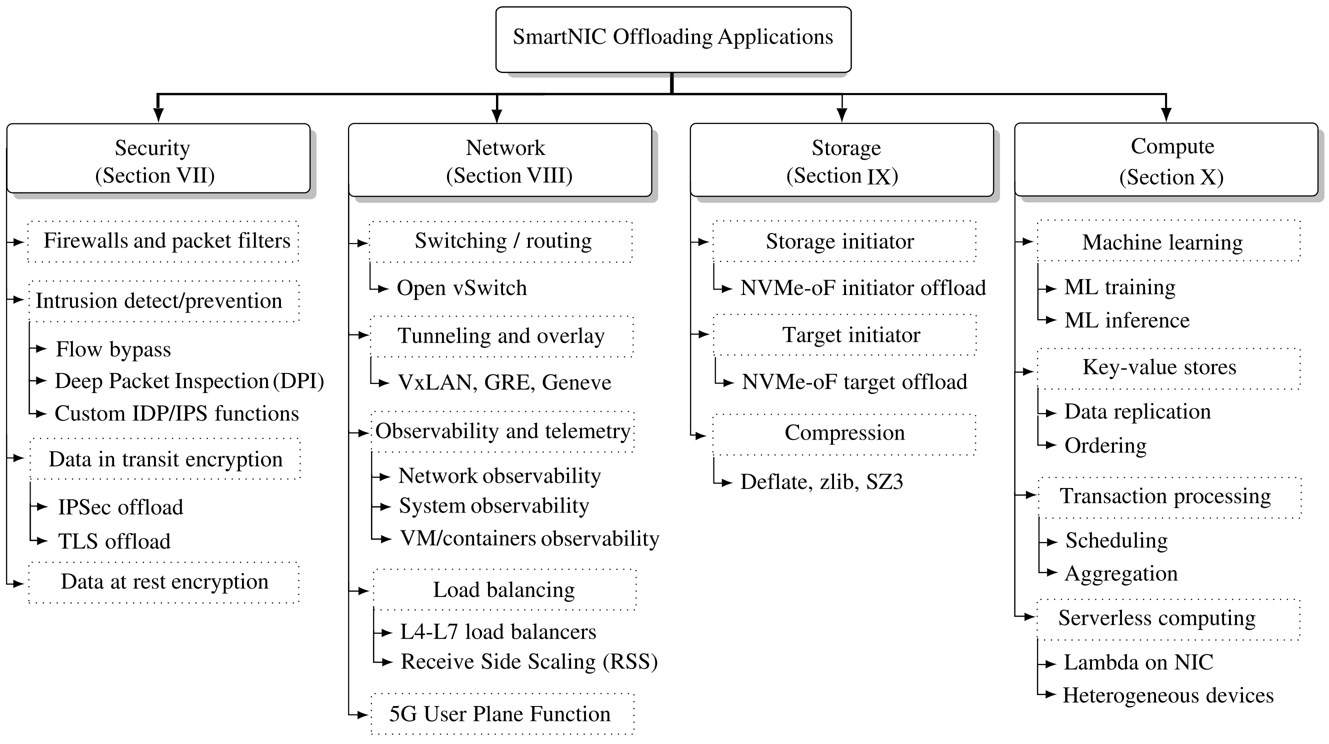


FIGURE 23. **Taxonomy of SmartNIC offloaded applications.**

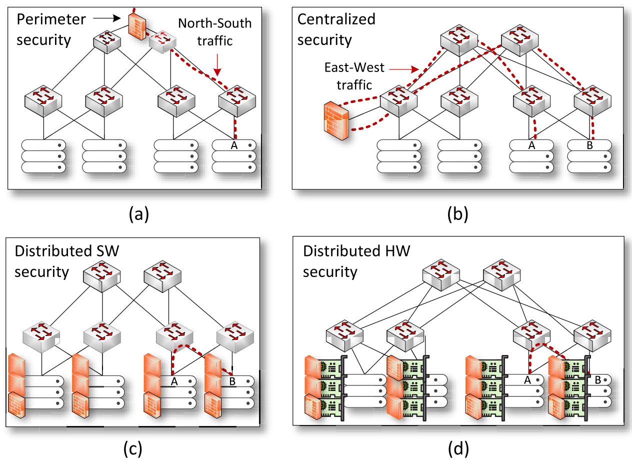


FIGURE 24. **(a) Perimeter-based security. The appliance only inspects NS traffic; (b) Centralized security. The appliance can inspect EW traffic, but the bandwidth overhead is high; (c) Distributed SW firewall. Software-based appliances are attached to the servers and can inspect EW traffic, but the performance is not high. (d) Distributed HW firewall. Appliances are offloaded to SmartNICs on the servers, enabling EW inspection with high performance.**

works based on various high-level disciplines. The taxonomy provides a clear separation of categories so that a reader interested in a specific discipline can only read the works pertaining to that discipline.

SmartNICs accelerate various infrastructure applications, categorized primarily into security, networking, and storage functions. It also accelerates various computing workloads including AI/ML inference and training, caching (key-value stores), transaction processing, serverless functions, and others. Each high-level category in the taxonomy is further divided into sub-categories. For instance, various transaction processing works belong to the sub-category "Transaction processing" under the high-level category "Compute". Additionally, the survey offers performance comparisons between applications running on the host and those offloaded to the SmartNICs.

The subsequent subsections delve into the ongoing developments within each of the aforementioned category, offering insights into the lessons learned from these advancements.

VII. SECURITY

The landscape of data center traffic has undergone a significant transformation with the rise of *cloud-hosted applications* and *microservices* [114]. Traditionally, traffic patterns were characterized by North-South (NS) flows (between internal and external devices) that are protected by dedicated security appliances (e.g., firewall) at the perimeter, see Fig. 24 (a). However, in the past decade, the dynamics have been shifting towards East-West (EW) flows (between internal data center devices), accounting for up to 80% of total data center traffic [115], [116]. Unlike North-South traffic, East-West traffic was relatively unprotected. A workaround for this was to use a centralized security appliance and forward EW traffic to it for inspection,[6] see Fig. 24 (b). This results in traffic traversing the intermediary devices (i.e., switches) twice, leading to duplication of both network load and the latency experienced by the two hosts.

This has led to the emergence of Zero Trust and microsegmentation architectures [117], whose main idea is to decentralize security functionality and move it closer to the resources that require protection. Data centers and cloud providers have shifted to using software-based security functions to protect East-West traffic [118], see Fig. 24 (c). While this shift is advantageous in terms of ease of deployment and cost-effectiveness, it has some drawbacks:

- Performance: Packets traverse the regular network stack to be processed by a security function on the general-purpose CPUs. This increases latency and decreases the throughput.
- Scalability: The CPU cores often struggle to inspect traffic at high rates, particularly in the absence of software accelerators (e.g., DPDK). This can lead to high packet drop rates.
- Isolation: all traffic, including malicious traffic, is sent to the host. This lack of isolation can pose security risks.
- CPU usage: security functions consume a substantial portion of the CPU processing power, particularly during periods of high traffic volume. This can result in performance bottlenecks and service degradation for end-user applications.

To mitigate these issues, SmartNICs have been used to offload the security functions from general-purpose CPUs, see Fig. 24 (d). Specifically, SmartNICs have been used to offload firewall functionalities, IDS/IPS, DPI, and data-at-motion and data-at-rest encryption.

A. FIREWALL

A firewall monitors incoming and outgoing network traffic and allows or blocks packets based on a set of pre-configured rules. Firewalls typically operate up to layer-4 to perform basic ACL operations. This means that the traffic can be matched against network layer information (e.g., source/destination IP addresses) and transport layer information (e.g., source/destination port numbers).

Software-based firewalls are widely being used, especially in cloud environments [118]. They are typically implemented in conjunction with a virtual switch (e.g., OvS). With software-based firewalls, traffic is inspected using the CPU cores of the host where the firewall is running. This degrades the performance and consumes the compute capacity of the CPU.

Recall that SmartNICs are equipped with a programmable pipeline or an embedded switch, where match-action rules can be defined. This makes it possible to implement firewalls with basic ACLs directly on the hardware at line rate. The functionality of the firewall can be implemented from scratch by a developer. However, it requires implementing many

[6]Sometimes referred to as traffic tromboning.

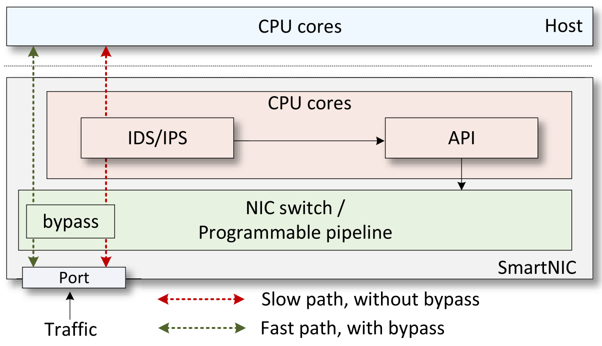


FIGURE 25. **IPS/IDS bypass offload to the SmartNIC.**

functions such as connection tracking if stateful inspection[7] is needed, flow caching and aging, etc. As an alternative, the hardware offloaded switch on the SmartNICs is being used to implement the firewall functionalities [119], [120]. The switch rules can be transparently offloaded to the hardware. The developer only needs to specify the rules that allow/block traffic. The connection tracking feature of the switch can be leveraged to enable stateful inspection. As an example, VMware allows offloading the firewall functionalities of its NSX distributed switch to the SmartNIC [17], specifically, the L2-L4 inspection and firewalling.

B. INTRUSION DETECTION/PREVENTION SYSTEM

Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) are cybersecurity technologies designed to safeguard networks and hosts from unauthorized access, malicious activities, and security threats. An IDS monitors and analyzes network or system events to identify suspicious patterns or anomalies. It provides real-time alerts or logs for further investigation. On the other hand, an IPS goes a step further by actively preventing or blocking unauthorized activities in real-time. Examples of open-source IDS/IPS include Zeek (formerly known as Bro) [122], Suricata [123], and Snort [124].

IDS and IPS are generally deployed on the general-purpose CPUs of the host. SmartNICs have been offloading IDS/IPS functions to accelerate data processing:

1) OFFLOADING IDS/IPS BYPASS FUNCTION

The IDS/IPS does not need to inspect every packet. Typically, the initial packets within a specific flow contain essential information, making continuous inspection unnecessary. There are situations, such as when dealing with an elephant flow resulting from a large data transfer, where it is imperative not to inspect the packets throughout the flow's lifetime. Additionally, encrypted traffic may also be exempt from inspection.

Current IDS/IPS systems incorporate bypass mechanisms within the software through the kernel datapath [132].

[7]Stateful inspection is a firewall technology that monitors and evaluates the state of active network connections, making decisions based on the context of the entire communication rather than individual packets.

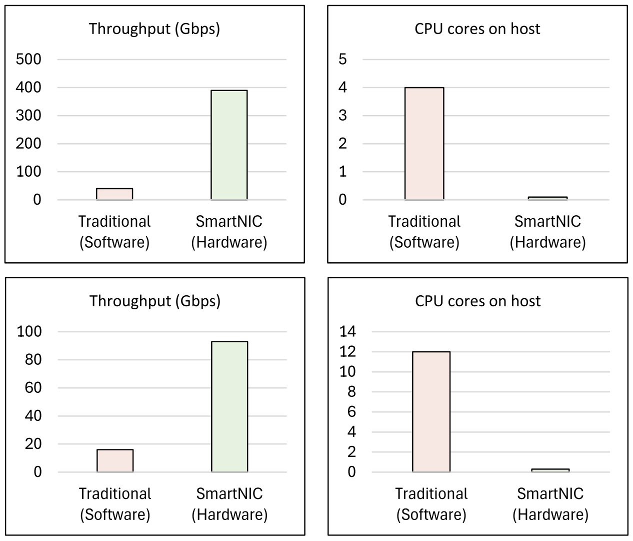


FIGURE 26. Performance of IPS/IDS with hardware-based bypass (SmartNIC offload) versus software-based bypass. The top row shows the results of offloading the bypass of Suricata IDS while the bottom row shows the result of offloading the bypass of Palo Alto's NGFW. Both were offloaded to NVIDIA's BlueField SmartNIC. Reproduced from [121].

While this enhances throughput, the process still depends on software that utilizes CPU cycles to efficiently route packets directly to the user space. Consider Fig. 25. With an offloaded IDS/IPS, the bypass function is implemented in the hardware without requiring additional intervention from the IDS/IPS or the host CPU [18], [121], [133]. The bypass is carried out on the programmable pipeline or the embedded switch within the SmartNIC. This significantly improves the performance.

Fig. 26 shows the throughput (left) and the CPU cores used on the host (right) of software-based IDS/IPS bypass and hardware-based (SmartNIC). The top row shows the results of offloading the bypass of Suricata IDS while the bottom row shows the result of offloading the bypass of Palo Alto's Next Generation Firewall (NGFW). Both were offloaded to NVIDIA's BlueField SmartNIC. The results show that hardware offloading (SmartNIC) can attain near line-rate throughput (∼1200% better than the software in the case of Suricata and ∼430% in the case of Palo Alto NGFW). Moreover, the CPU cores on the host are almost idle all the time.

2) OFFLOADING DEEP PACKET INSPECTION (DPI)

IDS and IPS often require inspecting the payload within packets to identify potential malicious patterns. One example of this is URL filtering, where the URL in the HTTP header of the packet is matched against a database. This process is used for access control and blocking known harmful websites and phishing pages. The software-driven nature of IDS/IPS in performing URL filtering has notable implications on the performance. This is because, for every packet, the IDS/IPS must parse deep into the packet contents (DPI). In addition to URL filtering, IDS/IPS systems apply DPI for various tasks such as application recognition (sometimes referred to as App-ID), signature matching for malware, etc.

SmartNICs are now integrating hardware-based RegEx engines. These engines perform pattern matching directly within the hardware, offering improved efficiency compared to traditional software-based approaches. Applications leveraging RegEx matching load a pre-compiled rule set into the engines at runtime. This hardware-driven approach helps alleviate the performance concerns associated with DPI in IDS/IPS, making network security more robust and responsive.

DPI has also been implemented on the hardware from scratch (e.g., using an FPGA). Ceska et al. [134] proposed an FPGA architecture for regular expression matching that can process network traffic beyond 100Gbps. The system compiles approximate Non-deterministic Finite Automata (NFAs) into a multi-stage architecture. The system uses reduction techniques to optimize the NFAs so that they can fit in the FPGA resources. The system was implemented on Xilinx FPGA. Other works [135], [136], [137], [138] have also explored optimizing NFAs for FPGAs.

3) OFFLOADING CUSTOM IPS/IDS FUNCTIONS

Zhao et al. [125] proposed Pigasus, an IDS that uses an FPGA to perform the majority of the IDS functions, and a CPU to perform the secondary functions. Pigasus achieves 100Gbps with 100K+ concurrent connections and 10K+ matching rules, on a single server. It requires on average five CPU cores and a single FPGA-based SmartNIC. The system was tested using Intel Stratix SmartNIC. Another FPGA-based solution proposed by Chen et al. [126] is Fidas, which offloads rule pattern matching and traffic flow rate classification. Fidas achieves lower latency and higher throughput than Pigasus. It was implemented on a Xilinx FPGA. Zhao et al. [127] implemented an FPGA design to analyze Internet of Things (IoT) traffic and summarize it in real time. The CPU then uses a flow entropy algorithm to detect the threats.

Panda et al. [128] proposed SmartWatch, a system that combines P4 switches and SmartNICs to perform IDS/IPS functions. The P4 switches perform coarse-grained traffic analysis while the SmartNIC conducts the finer-grained analysis. The SmartNIC used is Netronome Agilio. Wu et al. [129] implemented an anomaly detection-based IDS on the CPU cores of BlueField SmartNIC. The system uses the Analysis of Variance (ANOVA) statistical method for detecting anomalies. Tasdemir et al. [130] implemented an SQL attack detection system on the BlueField SmartNIC. The system uses NLP and ML classifiers to analyze and classify SQL queries. Miano et al. [131] implemented a DDoS mitigation system by combining hardware-based packet filtering on the SmartNIC and software-based packet filtering using XDP/eBPF.

Table 10 summarises and compares the aforementioned works that offload custom IDS/IPS functions to the SmartNICs.

TABLE 10. Comparison between various works offloading IDS/IPS functions to SmartNICs.

Work	Hardware used	Targeted attack	SmartNIC used	Key points
Pigasus [125]	FPGA, CPU	General	Intel Stratix	Multi-string matching on FPGA; FPGA is used as a primary engine; CPU is used as a secondary engine; Achieves 100Gbps using five CPU cores.
Fidas [126]	FPGA	General	Xilinx	Offload rule pattern matching and traffic classification to FPGA; Achieves lower latency and higher throughput than Pigasus.
Zhao et al. [127]	FPGA, CPU	IoT traffic attacks	N/A	FPGA-based traffic analysis; CPU-based threat detection using flow entropy algorithm.
SmartWatch [128]	P4 switches (ASIC) SmartNIC (CPU)	General	Netronome Agilio	Coarse-grained traffic analysis on P4 switches; Finer-grained analysis on SmartNIC.
ONLAD-IDS [129]	CPU cores	General	BlueField	Anomaly detection using ANOVA statistical method.
Tasdemir et al. [130]	CPU cores	SQL attacks	BlueField	Natural Language Processing (NLP) for SQL query analysis; ML classifiers for query classification.
Miano et al. [131]	ASIC, CPU	DDoS	N/A	Hardware and software-based packet filtering; Use cases target DDoS mitigation.

C. IPSec OFFLOAD

The Internet Protocol Security (IPSec) implements a suite of protocols to establish secure connections between end devices. This is achieved through the encryption and authentication of IP packets. IPsec comprises key modules including 1) *key exchange*, which facilitates the establishment of encryption and decryption keys through a mutual exchange between connected devices; 2) *authentication*, which verifies the trustworthiness of each packet's source; 3) *encryption* and *decryption*, which encrypts/decrypts payload within packets and potentially, based on the transport mode, the packet's IP header.

IPSec has a data plane (DP) and a control plane (CP). The CP is responsible for the key exchange and session establishment. The DP is used for encapsulating, encrypting, and decrypting packets. Traditionally, the CP and DP of IPSec are executed fully in the host, see Fig. 27 (a). This consumes CPU cores, increases latency, and decreases throughput. Once a packet is encrypted by the IPsec software, it is sent to the network over a traditional NIC.

The IPsec software can be offloaded to the SmartNIC to enhance security and performance, see Fig. 27 (b). The IPsec crypto operations (encryption/decryption) and encapsulation are executed by the hardware through the domain-specific accelerators. The accelerators include symmetric cryptography algorithms (e.g., Advanced Encryption Standard (AES)), asymmetric cryptography (e.g., RSA, Diffie-Hellman), and a True Random Number Generator (TRNG). This deployment model ensures transparency to the host, securing legacy workloads while benefiting from the offloading capabilities of IPsec.

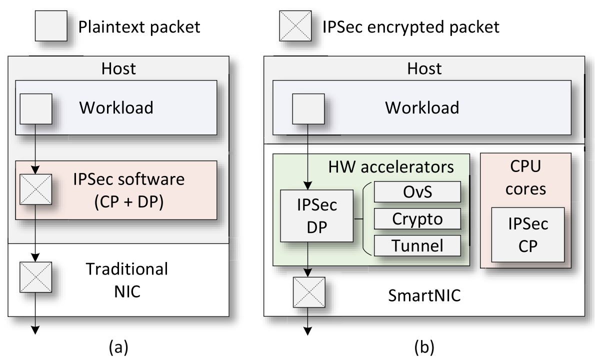


FIGURE 27. IPSec on the host (a) and IPSec offloaded to the SmartNIC (b).

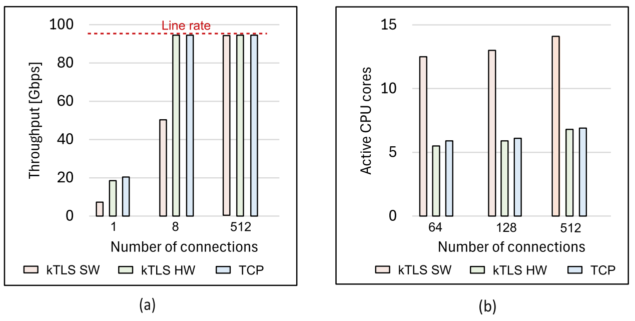


FIGURE 28. Throughput (a) and CPU core counts (b) of SW kTLS, HW kTLS, and plaintext TCP.

Diamond et al. [139] measured the performance of IPsec encryption in hardware on the BlueField SmartNIC. The results show that the offloaded IPSec is 10x faster than the fully software-based IPSec. Su et al. [140] evaluated IPSec using the encryption accelerator on an FPGA SmartNIC. The offloaded IPsec attained ∼19x and ∼483x throughput improvement at 64B and 1500B packet sizes, respectively.

D. TLS OFFLOAD

The prevalence of HTTPS servers using the TLS protocol exceeds 80% across all web pages. As the demand for accessing web servers continues to grow steadily, there is a need for an increased rate of bandwidth.

TLS operates on layer 4, on top of TCP. The TLS process, traditionally handled by user-space applications, has evolved with the advent of offloading techniques. The kernel TLS (kTLS) involves the offload of TLS operations

TABLE 11. Security offloaded service and the used accelerators.

Security offloaded service	Domain-specific accelerator				
	Match-action	RegEx	Symmetric Crypto	Asym. Crypto	TRNG
Stateful firewall	✓				
IDS/IPS	✓	✓			
IPSec			✓	✓	✓
TLS			✓	✓	✓
Storage encryption			✓	✓	✓

into the kernel, while Hardware (HW) kTLS offloads cryptographic functions to the domain-specific accelerators on the SmartNIC. With HW kTLS, the TLS handshake and the error handling (e.g., incorrect sequence number) are performed in software, while packets are encrypted and decrypted in hardware.

Consider Fig. 28 which shows the results of benchmarking the performance of HW kTLS with an Nginx server, reproduced from [141]. The throughput of kTLS SW is smaller than that of the HW kTLS when the number of connections is small. The HW kTLS achieved line rate with eight connections. The number of CPU cores on the host when using HW kTLS is lower than the SW kTLS and the unencrypted TCP, regardless of the number of connections.

Kim et al. [142] explored offloading the TLS handshake to the SmartNIC. Their proof of concept on a BlueField SmartNIC shows that there is a 5.9x throughput improvement over executing the handshake on a single CPU core. Novais and Verdi [143] performed experimental evaluations to assess the impact of TLS offload on a Chelsio SmartNIC. Their results suggest that hardware offloading improves the throughput, latency, and power consumption. Zhao et al. [144] further detailed experiments on offloading TLS to SmartNICs. Their results suggest that SmartNICs can be beneficial for latency-sensitive tasks, but require caution with computationally heavy loads.

E. DATA AT REST ENCRYPTION

SmartNICs can accelerate the encryption of data to be stored. Instead of using the CPU to encrypt the data, the domain-specific processor for encryption is used. Disk encryption protocol like AES-XTS 256/512-bit is used.

The implementation of encryption offload for storage through SmartNICs offers flexibility across various points in the storage data path. Encryption can occur directly on the storage device (e.g., Just a Bunch of Flash (JBOF)), securing data at rest. Alternatively, it may take place at the backend of the storage controller, ensuring the encryption of data in transit. Another option involves encryption at the initiator, with the initiator retaining control of the keys, and the encrypted data transmitted across the entire storage data path.

F. SUMMARY AND LESSONS LEARNED

SmartNICs significantly enhance the performance of security inspection. Table 11 summarizes the domain-specific accelerators used by the offloaded security functions. The key takeaways are:

- Offloading stateful firewall functions to the SmartNIC's embedded switch or programmable pipeline significantly boosts performance. The SmartNIC also allows running the firewall management and control planes on its CPU cores.
- The performance of IDS/IPS can be enhanced when their bypass function is offloaded to the hardware. Additionally, IDS/IPS can efficiently apply DPI using the RegEx hardware embedded in the SmartNIC. This is used in various security applications, including URL filtering, signature matching, content filtering, etc.
- SmartNICs facilitate the encryption of data-in-flight without compromising performance. Protocol stacks like IPSec or TLS can be easily offloaded to the SmartNIC, without requiring much development. Also, the SmartNIC provides APIs that enable developers to leverage the symmetric, asymmetric, and TRNG crypto processors.
- Data-at-rest can be encrypted by the SmartNIC, enabling faster storage encryption compared to the SW-based approach.
- The inherent programmability in SmartNICs opens avenues for developers to implement custom, novel, and performant security functionalities.

VIII. NETWORK OFFLOADS

Software-defined networking (SDN) and NFV are transformative technologies that have revolutionized the way networks are designed, deployed, and managed. Virtual switches play crucial roles in enabling the flexibility, scalability, and efficiency that modern networks demand, especially to connect VMs. The networking functions implemented as NFVs on the server strain the CPU, especially in networks with high traffic rates. Recently, SmartNICs have been used to offload the network functions from general-purpose CPUs. For instance, SmartNICs have been used to offload switching/routing, tunneling, measurement and telemetry, and others.

A. SWITCHING

Virtual switching emerged as a response to the need for hypervisors to seamlessly connect VMs with the external network [145]. Traditionally, virtual switches were running within the hypervisor, operating in software. However, this approach proved to be CPU-intensive, impacting overall system performance and preventing optimal utilization of available bandwidth [146], [147].

Software switches go beyond conventional layer-2 switching and layer-3 routing [148]; they facilitate rule matching on various packet fields and support diverse actions on packets.

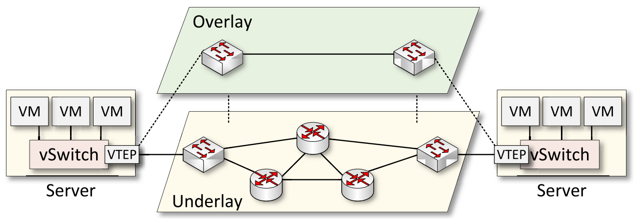


FIGURE 29. Tunneling.

These actions include forwarding, dropping, marking, and more.

1) SWITCHING OFFLOAD

SmartNICs, whether they use a NIC switch or a programmable pipeline, have lookups and ALUs implemented in hardware. These components can be used to implement the match-action functions required for switching packets. Instead of re-implementing all the functions required for switching, most SmartNICs allow offloading the datapath of existing software switches, such as OvS [145], an open source virtual switch. Note that it is possible to offload the datapath of proprietary switches, such as that of VMware's vSphere Distributed Switch [149]. Besides packet switching, virtual switches can handle additional tasks such as Network Address Translation (NAT), tunneling, and QoS functionalities such as rate limiting, policing, and scheduling.

B. TUNNELING AND OVERLAY

Tunneling is a technique that encapsulates and transports one network protocol over another. This is commonly used in virtualized environments to create isolated channels between VMs or between different segments of a virtualized network. Tunneling helps in overcoming the limitations of the underlying physical network and enables the creation of virtual networks that can span across physical boundaries. Various tunneling protocols are used in network virtualization, including Virtual Extensible LAN (VXLAN) [150], Geneve [151], Generic Routing Encapsulation (GRE) [152], etc. This subsection will discuss VXLAN, but the idea generalizes across all other tunneling protocols.

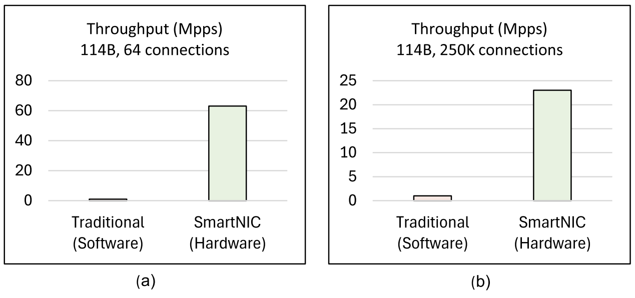


FIGURE 30. Throughput in million packets per second (Mpps) of software vs SmartNIC tunneling. (a) 114B packets and 64 connections; (b) 114B packets and 250,000 connections. Reproduced from [153].

VXLAN establishes a virtual network (i.e., overlay network) over an existing layer-3 infrastructure (i.e., underlay network) by creating tunnels between VMs. This overlay scheme enables the scalability of cloud-based services without the necessity to add or reconfigure the existing infrastructure. However, VXLAN introduces an additional layer of packet processing at the hypervisor level. Consider Fig. 29. Each packet leaving the VM must have an additional header to be transported over the underlay network. A VXLAN Tunnel End Point (VTEP) device encapsulates during packet transmission and decapsulates during packet reception. The VTEP is being implemented on software as part of the hypervisor stack. This process incurs additional CPU overhead [154]. As the number of flows scales up, overloading the CPU with packets for encapsulation/decapsulation can easily lead to network performance bottlenecks in terms of throughput and latency.

SmartNICs can offload tunneling functions from the host CPU [155]. They support inline encapsulation/decapsulation of VXLAN and other tunneling protocols. This logic is implemented in the embedded NIC switch [156] or the programmable pipeline [54]. Tunnels definition, which is part of the control plane, is implemented in software, on the CPU cores of the SmartNIC. This design not only improves throughput and reduces latency for the encapsulation/decapsulation operations, but also frees up CPU cycles on the host for other tasks. Fig. 30 compares the tunneling performance between the software and a BlueField SmartNIC, reproduced from [153]. With 114-byte packets and 64 connections, the SmartNIC tunneling is ∼60 times higher than the software-based. With 114-byte packets and 250,000 connections, the SmartNIC tunneling is ∼20 times higher than the software-based.

C. OBSERVABILITY - MONITORING AND TELEMETRY

Observability is the ability to collect and extract telemetry information. During a network outage, effective observability facilitates diagnosing and troubleshooting problems. It can also help in detecting malicious events and identifying network performance bottlenecks.

Traditional packet observability solutions are typically implemented in hardware, situated outside the server. Examples include configuring port mirroring (e.g., Switched Port Analyzer (SPAN)) on switches/routers, see Fig. 31 (a), deploying network TAPs for replicating packets, see Fig. 31 (b), and exporting flow-based statistics using NetFlow [157] or IPFIX [158] to a remote collector, see Fig. 31 (c).

1) OFFLOADING PACKETS OBSERVABILITY TO SmartNICs

The traditional approaches to packet observability are all supported by SmartNICs. SmartNICs can mirror packets and send them to remote collectors. They can also export telemetry using flow-based telemetry solutions like NetFlow or IPFIX, or using packet-level telemetry streaming such as In-band Network Telemetry (INT) [159] and In-situ

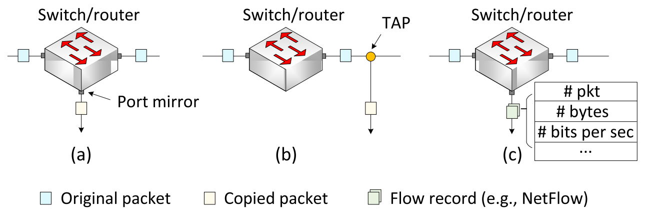


FIGURE 31. **(a) Port mirroring; (b) TAP; (c) NetFlow export.**

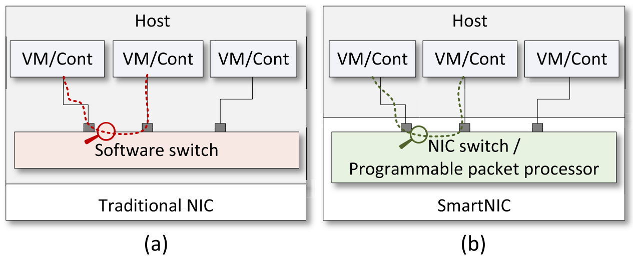


FIGURE 32. **VM and containers observability with (a) software switches and (b) SmartNICs.**

OAM [160]. SmartNICs can also monitor and aggregate telemetry locally, which avoids excessive traffic exports. Furthermore, since they incorporate programmable pipelines, they can be used to implement more complex packet telemetry than the traditional ones. For example, it is possible to implement streaming algorithms such as the Count-min Sketch (CMS) [161] to estimate the number of packets per flow in a scalable way, or a Bloom Filter [162] to test the occurrence of an element in a set. Such telemetry information can be very useful for a variety of applications (e.g., security [163], performance analysis [164], etc.).

2) OFFLOADING SYSTEM OBSERVABILITY TO SmartNICs

The SmartNIC also offers supplementary telemetry data related to the system in which it is located [165], such as the host. For example, the SmartNIC can provide telemetry data containing the CPU, memory, and disk usage of the host.

3) VM AND CONTAINERS OBSERVABILITY WITH SmartNICs

External approaches to packet observability cannot observe inter-VM/container traffic within the same server. While software-based approaches for monitoring VMs and containers exist, see Fig. 32 (a), they often burden the CPU, especially with high traffic rates [165]. SmartNICs provide hardware visibility on traffic between VMs or containers within the same server (see Fig. 32 (b)), alleviating the CPU burden on the host.

D. LOAD BALANCING

Load balancers play a crucial role in modern cloud environments by distributing network requests across servers in data centers efficiently. Traditionally, load balancers relied on specialized hardware, but now software-based solutions are prevalent among cloud providers. This shift offers flexibility and allows for on-demand provisioning on standard servers, though it comes with higher provisioning and operational expenses. While software-based load balancers offer greater customization and adaptability compared to hardware-based counterparts, they also entail considerable costs for cloud providers due to server purchase expenses and increased energy consumption.

Load balancers are categorized into two main types: Layer 4 (L4) and Layer 7 (L7). L4 load balancers function at the transport layer of the network stack. They associate a Virtual IP address (VIP) with a list of backend servers, each having its own dynamic IP (DIP) address. Routing decisions made by L4 load balancers are based solely on the packet headers of the transport/IP layers, considering factors such as source and destination IP addresses and ports. Thus, L4 load balancers do not inspect the payload content of the packets. On the other hand, L7 load balancers operate at a higher layer, specifically the application layer. These balancers are more intricate, as they analyze content within the packets, particularly focusing on application-layer protocols like HTTP. The L7 load balancer directs incoming requests to appropriate backend servers based on the specific service being accessed. For instance, differentiation may occur based on URLs.

1) OFFLOADING LOAD BALANCING TO SmartNICs

Several works have offloaded load balancing to SmartNICs. Cui et al. [166] proposed Laconic, a system that improves the performance of load balancing due to three key points: 1) Lightweight network stack: unlike traditional L7 load balancers, which heavily rely on the operating system's TCP stack, Laconic opts for a lighter packet forwarding stack on the load balancer itself. This approach minimizes overhead and leverages the end-hosts to achieve the desired end-to-end properties; 2) Lightweight synchronization for shared data structures: Laconic implements a concurrent connection table design based on the cuckoo hash table. This design efficiently manages hash conflicts and reduces the number of entries needing probing during lookups; 3) Acceleration with hardware engines: Laconic optimizes packet processing by transferring common packet rewriting tasks to hardware accelerators. This strategy alleviates the processing burden on the CPU cores of the SmartNIC. Huang et al. [167] offloaded the load balancer to an FPGA-based SmartNIC. The result shows that the system was able to load-balance at 100Gbps. Chang et al. [168] described a scheme that finds an optimal load balancing strategy for a network topology. It uses SmartNICs and programmable switches. Other works [169], [170], [171], [172] used variations of the methods above for load balancing.

2) RECEIVE SIDE SCALING (RSS)

SmartNICs commonly include an accelerator for RSS, which is a mechanism to distribute incoming network traffic across

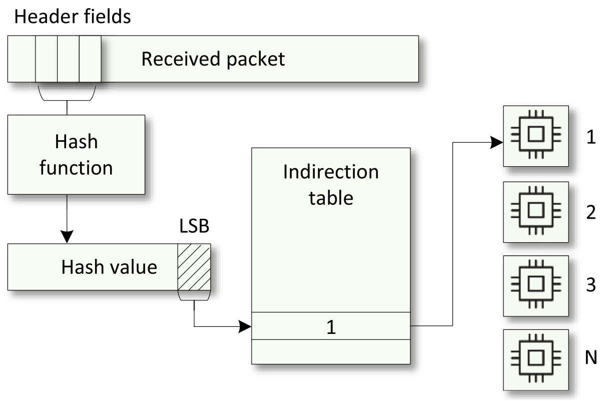


FIGURE 33. Receive side scaling (RSS).

multiple CPU cores. To achieve this, the SmartNIC calculates a hash value (Toeplitz hash [173]) based on header fields (such as the five-tuple) of the received network packet, see Fig. 33. The hash value's Least Significant Bits (LSBs) are then used as indices for an indirection table, the values of which are used to allocate the incoming data to a specific CPU core. Some SmartNICs allow steering packets to queues based on programmable filters [174].

E. 5G UPF

The User Plane Function (UPF) in 5G networks represents the data plane within the packet core. It connects the User Equipment (UE) from the Radio Access Network (RAN) to the data network, see Fig. 34. The UPF typically performs packet inspection, routing, and forwarding, and QoS enforcement. It processes millions of flows with a high connection rate. 5G networks implement the packet core as VNF running on general-purpose CPUs rather than dedicated appliances. General-purpose CPUs are not capable of guaranteeing high throughput and low latency, which are the requirements and the Key Performance Indicators (KPI) of 5G networks.

1) UPF OFFLOAD TO SmartNIC

The SmartNIC can be used to offload the UPF functions [175]. Specifically, the following functions are offloaded: GTPU tunneling: the encapsulation and decapsulation of packets run at line rate; policing: the SmartNIC will control the bit rates of the devices so that they do not exceed the Maximum Bit Rate (MBR); statistics: the counters and metrics are calculated and used for billing purposes; QoS: the SmartNIC performs Differentiated Services Code Point (DSCP) on flows to enable 5G QoS; Load balancing: the SmartNIC balances the traffic to the corresponding application; Network Address Translation (NAT): the SmartNIC translates IP addresses on traffic; etc.

Offloading the UPF will not only improve throughput and reduce latency, but it will also boost the number of users per server (7x according to [175]) and lower the Capital Expenditure (CapEx) per user.

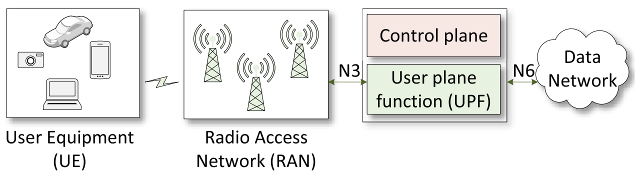


FIGURE 34. 5G network architecture. The packet core is implemented as VNF on general-purpose CPUs. The SmartNIC is being used to offload the UPF functions.

F. SUMMARY AND LESSONS LEARNED

SmartNICs significantly improve the performance of network functions and reduce their CPU consumption on the hosts. The key takeaways are:

- The packet switching functions (i.e., matching header fields and taking actions), can be accelerated with SmartNICs. This is because SmartNICs, whether they use a NIC switch or a programmable pipeline, have lookups and ALUs implemented in hardware.
- The performance of tunneling operations (encapsulation/decapsulation) can be significantly improved when offloaded to the SmartNIC. This also frees the CPU cores that were previously used for performing the tunneling operations.
- SmartNICs not only support traditional telemetry solutions but also allow the developer to devise custom fine-grained measurement schemes. They also enable inter-VM/container packet observability and host metrics telemetry.
- Offloading the UPF of 5G improves the performance of packet processing, increases the number of users per server, and decreases the per-user CAPEX.
- Instead of re-implementing all the switching functions, SmartNICs allow offloading the datapath of existing software switches.
- Developers can devise custom packet processing algorithms not supported by existing software switches.

IX. STORAGE

Traditionally, storage devices were directly attached to individual computers or servers. This method provided fast access to data but lacked scalability and centralized management. Network Attached Storage (NAS) emerged as a solution to these limitations. It involves connecting storage devices to a network, allowing multiple users and clients to access the storage resources over the network. NAS provided file-level access to data. Storage Area Network (SAN) provides a high-speed network that connects storage devices to servers, providing block-level access to storage resources. SANs offer higher performance and scalability compared to NAS.

Traditional remote storage mechanisms establish a connection between a local host initiator and a remote target. This process heavily burdens the host CPU, leading to a significant

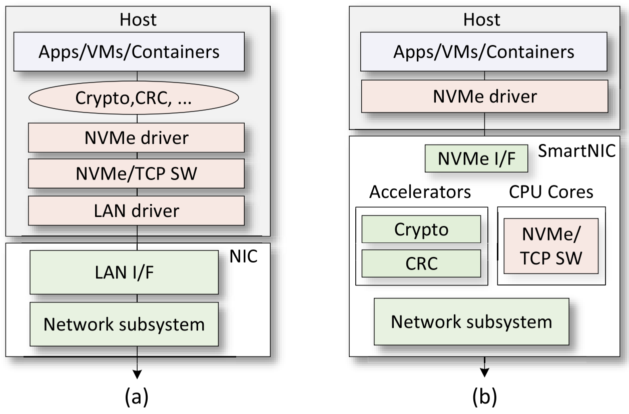


FIGURE 35. (a) NVMe-oF initiator without offload. (b) NVMe-oF initiator offloaded to the SmartNIC.

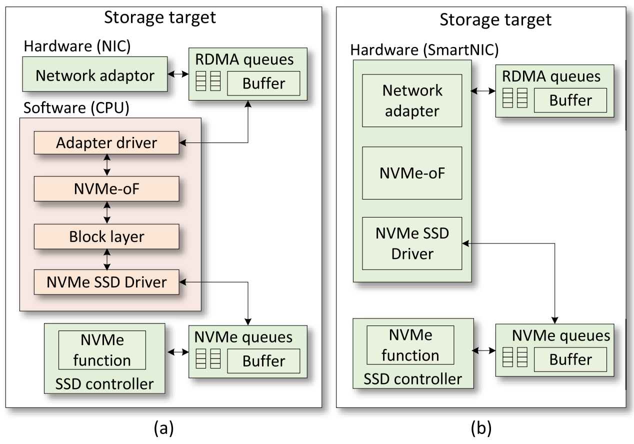


FIGURE 36. (a) NVMe-oF target without offload. (b) NVMe-oF target with offload. Reproduced from [176].

decrease in overall performance. SmartNICs can be used to offload the processing from the host CPU.

A. NVMe-OF INITIATOR

Non-Volatile Memory Express (NVMe) is an interface specification for accessing a computer's non-volatile storage media usually attached via the PCI Express bus. It is typically used for accessing high-speed storage devices like Solid State Drives (SSDs). NVMe over Fabrics (NVMe-oF) extends NVME to operate over network fabrics such as Ethernet, Fibre Channel, or InfiniBand. The NVMe initiator initiates and manages communication with NVMe targets. It sends commands to NVMe targets to read, write, or perform other operations. The NVMe target refers to the NVMe storage device itself.

Fig. 35 (a) shows the traditional method of NVMe-oF using the TCP protocol and a regular NIC. The entire NVMe-oF initiator software stack operates on the host. Tasks such as cryptography and CRC computations further strain the host CPU and memory bandwidth.

1) NVMe-oF INITIATOR OFFLOAD

The NVMe-oF initiator functionality can be offloaded to the SmartNIC (Fig. 35 (b)), minimizing the overhead on the host. The SmartNIC exposes a high-performance PCIe interface and NVMe interface to the host. Requests from applications are simply forwarded to a lightweight NVMe driver on the host. The initiator stack on the SmartNIC leverages the hardware accelerators for tasks like inline cryptography and CRC offloading. The TCP stack can either remain on the CPU cores of the SmartNIC or be offloaded to the hardware itself, depending on performance considerations and SmartNIC capabilities. The division of NVMe-oF functions between hardware and software allows for optimization based on performance and SmartNIC capabilities.

Another offload to the SmartNIC is the NVMe-oF RDMA. The NVMe/RDMA data path is implemented in the hardware, with inline cryptography and CRC offloaded. This approach offers a high-performance, low-latency solution.

B. NVMe-OF TARGET

Another offload opportunity is offloading the storage target functions. On a storage target such as JBOF supporting NVMe-oF, there is a CPU positioned between the network and NVME SSDs, see Fig. 36 (a). This CPU runs software responsible for converting NVME-over-Fabrics Ethernet or InfiniBand signals to NVME PCIe signals. The software comprises various components, including a network adapter stack, NVME-over-Fabrics stack, operating system block layer, and NVME SSD stack. Both the network adapter and SSD utilize queues and memory buffers to interface with different software stacks.

When a request originates from the network, it arrives at the network adapter as an RDMA SEND with the NVME command encapsulated. The adapter then forwards it to its driver on the target CPU, which further passes it to the NVMe-oF target driver. The NVME command proceeds through the driver for the SSDs and then to the NVME SSD controller. Subsequently, the response follows the reverse path through the software layers.

1) NVMe-OF TARGET OFFLOAD

With the offload, the fast path is shifted to the hardware on the SmartNIC. Instead of burdening CPU cycles with millions of Input/Output Operations per Second (IOPS), the adaptor now handles the load using specialized function hardware. Software stacks remain in place for management traffic. The reduction in latency by removing the software from the data path is by a factor of three [176]. Moreover, the CPU usage with offload is negligible.

C. COMPRESSION AND DECOMPRESSION

The surge in data volumes has caused performance bottlenecks for storage applications. Data compression is a widely adopted technique that mitigates this bottleneck by reducing the data size. It encodes information using fewer bits than the original representation. Notably, machine learning,

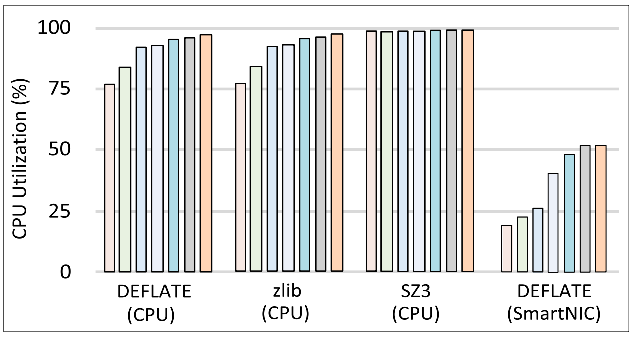


FIGURE 37. CPU utilization during compression with various algorithms (DEFLATE, zlib, SZ3) on seven datasets. Reproduced from [177].

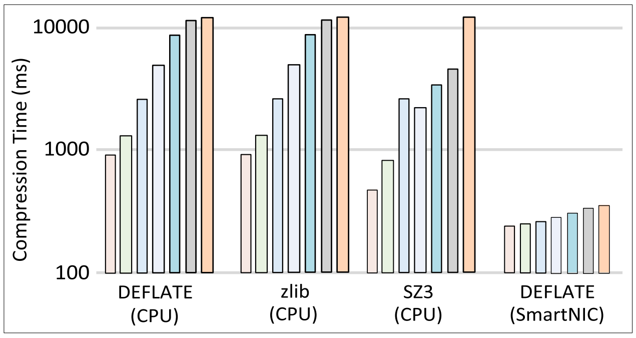


FIGURE 38. Compression time with various algorithms (DEFLATE, zlib, SZ3) on seven datasets. Reproduced from [177].

databases, and network communication rely on compression techniques—both lossless and lossy—to enhance their performance. Data compression is compute-intensive and time-consuming, especially with large sizes of data to be compressed.

1) OFFLOADING COMPRESSION TO SmartNICs

SmartNICs include onboard hardware accelerators that enable the offloading of compression and decompression tasks from host CPUs. This offloading alleviates the strain on host resources, resulting in savings and improved performance. Fig. 37 shows the CPU utilization when compression is executed entirely on the host (denoted as CPU) versus when executed on the compression hardware engine of the SmartNIC (denoted as SmartNIC). The experiment shows results for various compression algorithms (e.g., DEFLATE [178], zlib [179], SZ3 [180]) over seven datasets. The datasets are sorted in the figure by their sizes in ascending order– each dataset is a column in the figure. The experiment is reproduced from [177]. When the compression is executed entirely on the host, the CPU usage approaches 100%, especially with large datasets. With a SmartNIC, there is a significant reduction in the CPU utilization.

Fig. 38 shows the compression time needed when executed entirely on the host (denoted as CPU) versus when executed on the compression hardware engine of the SmartNIC. The experiment is reproduced from [177]. With a SmartNIC, there is a significant reduction in the compression time, regardless of the size of the dataset.

D. SUMMARY AND LESSONS LEARNED

Offloading storage functions to the SmartNICs improves the performance.

- Due to the hardware accelerators in the SmartNIC (e.g., compression, crypto), storage operations like compression, deduplication, and crypto will run faster than on the host's CPU.
- The SmartNIC can be deployed on the initiator or the storage target. In both deployments, the CPU usage on the hosting device is negligible, the latency is minimized, and the number of IOPS is improved.

X. COMPUTE

This section examines applications offloaded to the SmartNIC that are not specifically tailored to infrastructure functions. Instead, these applications leverage the SmartNIC for accelerated computing tasks.

A. MACHINE LEARNING

State-of-the-art deep ML models have significantly expanded in size, playing a critical role in various domains, including computer vision, Natural Language Processing (NLP), and others [46]. The scale of these models has seen a dramatic increase, with the number of parameters growing from 94 million in 2018 [181] to 174 trillion in 2022 [182]. This exponential growth owes much to advancements in parallel and distributed computing, enabling tasks related to model training to be distributed across multiple computing resources simultaneously. The practice of offloading parts of ML tasks to network resources traces back to the 2000s [183], a trend that continued with the advent of Software Defined Networking (SDN), where ML primarily operates within the control plane [184]. The recent emergence of programmable data planes (i.e., programmable switches, SmartNICs) has further spurred research and practical applications toward offloading ML phases, such as training and inference, to the hardware. Offloading ML tasks can occur on a single network device or across multiple devices, depending on network requirements and the complexity of the offloaded ML task.

1) ML TRAINING

The training of large ML models can be accelerated by following a distributed approach. This involves computing gradients on each device based on a subset of the data, which are then aggregated to update model parameters. Additionally, optimization of model parameters can be carried out in the data plane to maximize accuracy.

Itsubo et al. [185] devised a system that employs in-network gradient aggregation and parameter optimization for neural networks using an FPGA-based SmartNIC. Fig. 39 shows the high-level architecture of [185]. Gradient computation occurs on GPUs and is subsequently transmitted to the FPGA via PCIe, where aggregation takes place. Following

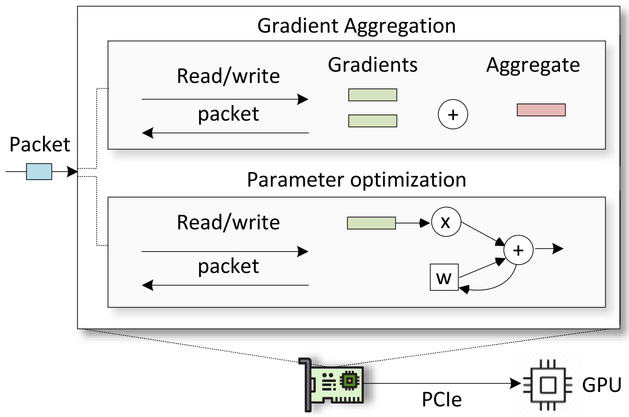


FIGURE 39. High-level architecture of [185].

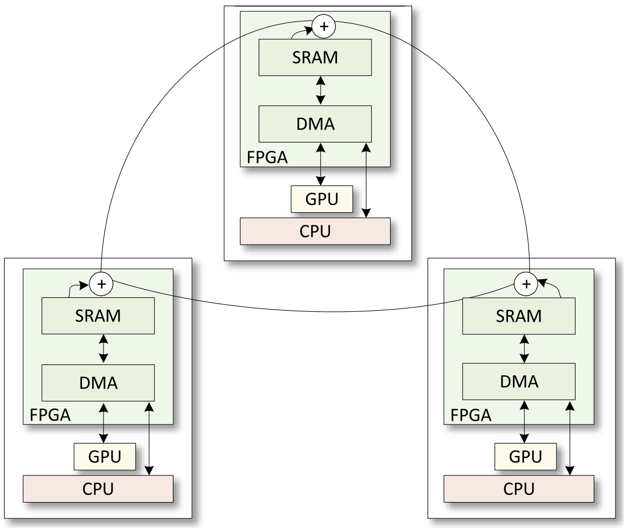


FIGURE 40. High-level architecture of [185].

aggregation, the FPGA can execute parameter optimization algorithms, including Stochastic Gradient Descent, Adagrad, Adam, and SMORMS3. The proposed framework achieves aggregation at 98.5% line rate and accelerates parameter optimization by $\approx$ 1.2 times. Tanaka et al. [186] opt for a different approach to aggregation, employing a ring network topology of FPGA-based SmartNICs using the allreduce algorithm [187]. In this setup, each node within the ring network awaits data from the preceding node, aggregates it upon reception using the all-reduce aggregation algorithm, and subsequently forwards it to the succeeding node (as illustrated in Fig. 40). This strategy not only alleviates CPU load but also establishes a direct memory link between the GPU and the FPGA, thus preserving accuracy, as floating-point operations required by the algorithm are supported at the hardware level. Similarly, Ma et al. [188] improve distributed ML training by performing the entirety of allreduce operations on an FPGA-based SmartNIC in a ring network topology. The proposed approach compresses the gradient before they are shared with other nodes, thus reducing bandwidth usage. The aggregation is performed on the SmartNIC of the end-host nodes.

2) ML INFERENCE

In the inference phase, various ML models such as decision trees, neural networks, and reinforcement learning algorithms undergo training on a general-purpose CPU. Once trained, these models are translated into rules that can be executed within the data plane of the device. This approach enables accelerated inference, enhancing the efficiency of real-time decision-making processes.

IIsy [189] explore the feasibility of deploying different classification algorithms on programmable data planes. In particular, IIsy can implement decision trees, K-means, Support Vector Machine (SVM), and Naïve Bayes to perform per-packet classification. The framework converts the code into match-action tables compatible with programmable data planes. IIsy's prototype is implemented over an FPGA-based SmartNIC using P4 [190]. Xavier et al. [191] developed a framework that translates decision trees into a P4-programmable data plane using if-else chain of statements. The proposed framework differs from [189] by introducing per-flow classification. BaNaNa Split [192] accelerates neural networks inside programmable switches and SmartNICs. This approach leverages the layered structure of neural networks by splitting them between the CPU and the network processor. However, BaNaNa Split necessitates quantization, a process that reduces the precision of neural network weights at the cost of diminishing accuracy.

B. KEY-VALUE STORES

Data centers face a growing demand for collecting and analyzing vast amounts of data. Typically, this data is stored in key-value stores due to their superior performance over traditional relational database systems. Popular key-value stores include Redis [194] and Memcached [195]. As data volumes increase, so does the frequency of reads and writes to these stores, leading to bottlenecks in the traditional network protocol stack and heavy CPU consumption.

The emergence of SmartNICs offers a solution by offloading key-value store operations to accelerate performance

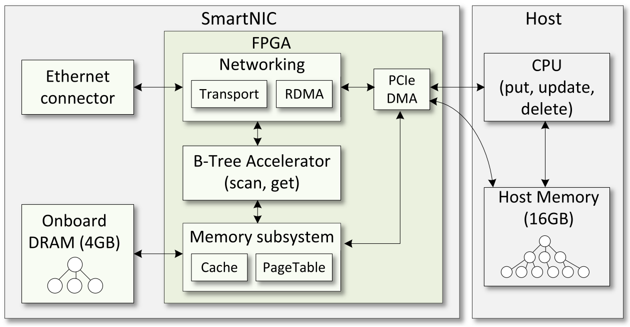


FIGURE 41. Honeycomb system architecture [193].

and reduce CPU load. One effective method is leveraging RDMA [196], [197], [198], [199], [200]. RDMA allows data to be read from or written to memory without involving the operating system and the traditional network stack. This results in lower latency, reduced CPU overhead, and higher bandwidth compared to traditional networking approaches.

Sun et al. [201] implemented SKV, a distributed key-value store accelerated with SmartNIC. The system offloads the data replication and failure detection components. It targets the Redis key-value store and is implemented using the BlueField SmartNIC. The evaluations show that the system reduces the latency by 21% and increases the throughput by 14% compared to being implemented fully on the host without SmartNIC acceleration.

Another aspect of the key-value store that was offloaded to the SmartNIC is the ordering of elements. Ordered key-value stores enable additional applications by allowing an efficient SCAN operation. Liu et al. [193] proposed Honeycomb, an FPGA-based system that provides hardware acceleration for an in-memory ordered key-value store. It focuses on the read-dominated workloads. Consider Fig. 41. The B-Tree accelerator implements the GET and SCAN operations. The CPU executes the PUT, UPDATE, and DELETE operations. The B-Tree is stored on the onboard DRAM in FPGA and on the memory of the host. Storing the B-Tree on the host allows better scalability since its memory is larger than that of the FPGA. The memory subsystem maintains a cache and communicates with the onboard DRAM. It also communicates with the host memory using PCIe. The implementation shows that the system increases the throughput of another ordered key-value store [202] by 1.8x.

Chen et al. [203] designed a heterogeneous key-value store where a primary instance runs on the host and a secondary instance runs on a SmartNIC. The system identifies the popular items and replicates them to the SmartNIC. The popular items are identified with moving window access counters. The server instance serves the read and write requests of all keys while the SmartNIC instance serves only the read request of popular items. This system targets read-intensive workloads with skewed access. The system was implemented on a BlueField-2 and the results show that the throughput is improved by 1.86x than a standalone RDMA key-value store.

C. TRANSACTION PROCESSING

High-performance transaction processing is important to enable various distributed applications. These systems need to manage a large number of requests from the network efficiently. One crucial aspect is determining how to schedule each transaction request to the most suitable CPU core.

Consider Fig. 42 (a) which shows the architecture of a transaction processing system without scheduling. A traditional NIC receives requests from the clients and dispatches them to the worker threads. The worker threads then execute the transaction, while considering the contention issues that

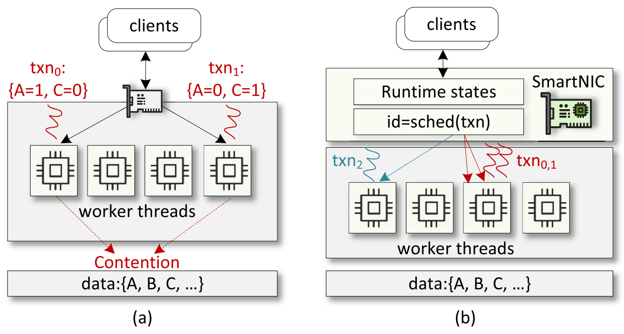


FIGURE 42. Transaction processing systems. (a) a system without scheduling, (b) scheduling using SmartNIC. Reproduced from [204].

might happen. Contention in this context means that two workers are accessing the same data and at least one of them is issuing a write. In Fig. 42 (a), two transactions (txn$_0$ and txn$_1$) are writing to the same data blocks A and C. In such a scenario, the transactions are typically aborted, causing the clients to resend the transactions, which degrades the performance.

Li et al. [204] proposed using a SmartNIC to schedule the transactions to the appropriate worker threads. The SmartNIC maintains the runtime states, giving it the flexibility to make accurate scheduling decisions. The SmartNIC queues the transactions belonging to the same worker thread. This avoids having the clients resend the transactions. The system is implemented on an FPGA-based SmartNIC, which further reduces the scheduling overhead. The system was implemented over the Innova-2 SmartNIC and the results show that the throughput is boosted by 2.68x and the latency is reduced by 48.8% compared to the CPU-based scheduling.

Schuh et al. [205] implemented Xenic, a SmartNIC-based system that applies an asynchronous, aggregated execution model to maximize network and core efficiency. It uses a data store on both the SmartNIC and the host. This data store provides fast access to host data via indexing. It also maintains a state to enhance the concurrency and contention issues. Xenic also aggregates work at all inputs and outputs of the SmartNIC to achieve communication efficiency. The system was implemented on a LiquidIO SmartNIC. The results show that Xenic improves the throughput of prior RDMA-based systems by approximately 2x, reduces the latency by up to 59%, and saves server threads.

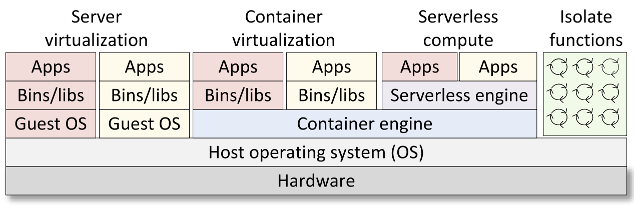


FIGURE 43. Architectures used in the cloud. Reproduced from [206].

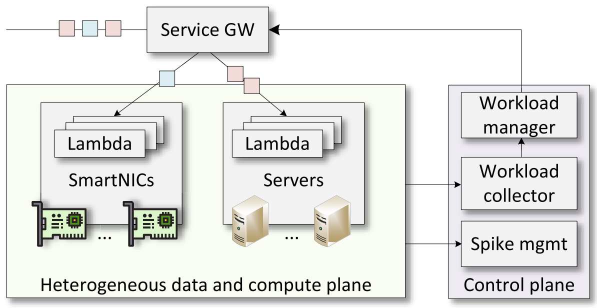


FIGURE 44. **SpikeOffload system architecture. Reproduced from [21].**

D. SERVERLESS COMPUTING

Fig. 43 shows the architectures used in the cloud today. The *server virtualization* allows guest operating systems to run on top of a host operating system. The applications and the libraries run on top of the guest OS and are isolated from other operating systems. The trend has shifted towards *container virtualization* where applications and their libraries are isolated but they share the same OS. The containers can be connected through software switches to enable their communications. The complexity and the scale of the cloud make it hard to manage and provision the infrastructure with tasks requiring fine-grained allocation of resources under changing workload demands. This has led to the *serverless compute* architecture, also known as Function as a Service (FaaS). In a serverless architecture, developers write code that represents functions (also known as Lambdas), and these functions are triggered by various events. The users are billed only for the resources consumed during the execution. The serverless workloads, which are targeted by these functions, are typically short-lived with strict computing and memory limits. The cloud provider will manage the infrastructure by creating containers and taking them down when the workload is completed. Examples of serverless computing frameworks include Amazon Lambda [207], Google Cloud Functions [208], and Microsoft Azure Functions [209].

Running the serverless computing functions on top of containers that are running on top of an OS incurs processing and networking overhead that increases the latency. Recently, cloud providers have been using the *isolate functions* architecture in which the functions are executed on a bare metal server.

1) EXECUTING LAMBDA FUNCTIONS ON SmartNICs

Recent efforts have explored the potential of executing Lambda functions on the SmartNICs. Choi et al. [206] proposed λ-NIC, a framework where Lambda functions are executed on the SmartNIC. It provides a programming abstraction, which resembles the match-action of the P4 language, to express the lambda functions. The framework analyzes the memory accesses of the functions to map them across the memory hierarchy of the SmartNIC. Because the workloads are short-lived, λ-NIC assigns a function to a single core on the SmartNIC. The system was implemented on a Netronome Agilio CX, and the results show that λ-NIC can decrease the average latency by 880x and improve the throughput by 736x.

Tootaghaj et al. [21] proposed SpikeOffload, a system that offloads serverless functions to the CPU cores of the SmartNICs, in the presence of transient traffic spikes, see Fig. 44. A workload collector module gathers the history of workloads and feeds the summary to the *workload manager* module. The workload manager module predicts the workload spikes based on the service time and the CPU loads of the servers and the SmartNICs. It then configures the service gateway (GW) to distribute the requests to the corresponding device (i.e., servers and SmartNICs) in the compute plane. SpikeOffload predicts the spikes in the workloads using ML. It starts the containers before the actual load arrives to mitigate the containers' cold start latency. The system was implemented on a BlueField-2, and the results show that the Service Level Agreement (SLA) violations for certain workloads can be reduced by up to 20%.

E. SUMMARY AND LESSONS LEARNED

SmartNICs extend their utility beyond infrastructure-related tasks, accelerating various compute functions. The key takeaways include:

- Machine learning tasks, encompassing distributed training and inference, experience significant performance enhancements when offloaded to SmartNICs. These devices efficiently aggregate model updates from multiple ML workers and optimize model parameters. Their programmable pipeline also enables the execution of certain ML models directly for line-rate inference.
- Key-value stores operations, which include retrieving and updating data, replicating stores, and detecting failures, can be offloaded to SmartNICs. This would bring notable throughput and latency improvements.
- SmartNICs can be used to schedule transactions, aggregate values, and solve contention in distributed systems, improving the latency and throughput.
- SmartNICs can execute serverless workloads (lambda functions), which reduces the load on the servers. They can also be used as an additional execution engine in a heterogeneous data and compute cluster.

XI. CHALLENGES AND FUTURE TRENDS

In this section, several research and operational challenges that correspond to SmartNICs are outlined. The challenges are extracted after comprehensively reviewing and diving into each work in the described literature. Further, the section discusses and pinpoints several initiatives for future work that could be worthy of being pursued. The challenges and the future trends are illustrated in Fig. 45

A. ARCHITECTURAL DIVERSITY AND VENDOR SPECIFICITY

SmartNICs can have different architectural models, each requiring unique programming approaches. Even within

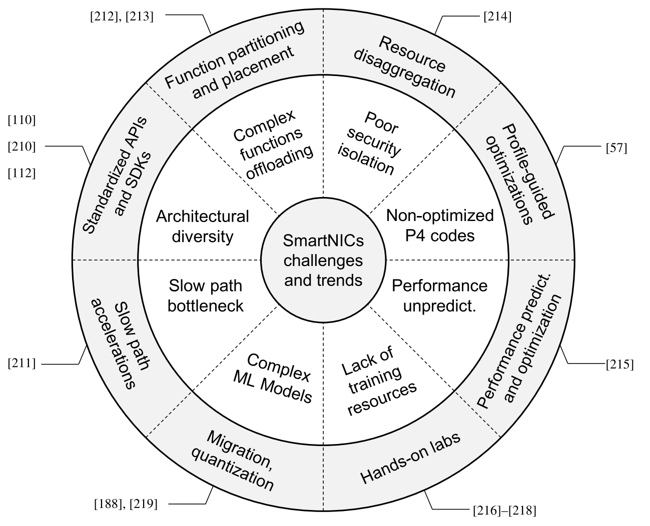


FIGURE 45. Challenges and future trends. The references represent examples of existing works that tackle the corresponding future trends.

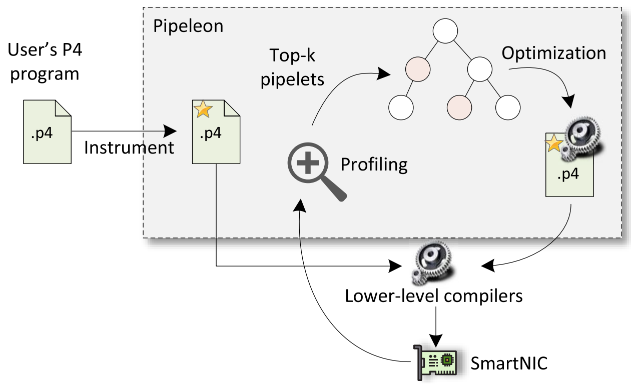


FIGURE 46. Pipeleon workflow. Reproduced from [57].

the same architecture, SmartNICs from different vendors may necessitate proprietary SDKs and distinct programming methods, which present several challenges:

- Vendor Lock-In: Developers may become dependent on a specific vendor's SDK, making it challenging to migrate to alternative SmartNICs or adopt new technologies. For instance, consider the scenario where a developer has written a packet processing logic using DOCA for BlueField SmartNICs. If they were to transfer this logic to Xilinx FPGA-based SmartNICs, they would need to rewrite the entire logic from scratch.
- Reduced Collaboration: Proprietary SDKs hinder collaboration and knowledge shared among developers, as expertise gained in one ecosystem may not be easily transferable to another.
- Increased Development Time and Costs: When developers need to tailor their code for each SmartNIC's proprietary SDK, it significantly increases development time and costs. Instead of focusing on advancing the functionality and performance of their applications, developers must spend valuable resources adapting their code to work with different SmartNIC architectures and vendor-specific APIs. This diversion of resources can slow down the pace of innovation within organizations and the industry as a whole.

Current and Future Initiatives: To address these challenges and foster innovation in the SmartNIC space, there is a growing need for standardized programming interfaces and open-source development frameworks. Standardization efforts could promote interoperability among SmartNICs from different vendors and enable developers to write code that is portable across various architectures. Additionally, open-source initiatives can encourage collaboration, drive community-driven innovation, and provide developers with more flexibility and control over their software stack. Several initiatives (e.g., OPI, IPDK, SONiC-DASH) aim to establish standard APIs for SmartNIC programming and administration, reducing vendor dependency. However, vendor-specific functions remain a challenge for generalization.

B. NON-OPTIMIZED P4 CODES

Developers have been using low-level optimization to enhance the performance of packet processing in SmartNICs. Recently, vendors are embracing P4 as a uniform programming model for SmartNICs [54], [81], [96]. While P4 allows ease of programming and offers a high-level standardized model, it does not guarantee the optimal performance on SmartNICs. This is because the P4 compilers were optimized for switch ASICs which have a different execution model than SmartNICs. With switch ASIC, if the program compiles, the packet processing executes at line rate. SmartNICs on the other hand follow the run-to-completion model, where packets are assigned to a particular processing engine during the lifetime. With multicore SmartNICs, the packets may experience variable latencies depending on the complexity of the program and its execution paths.

Current and Future Initiatives: A noteworthy work by Xing et al. [57] presented an automated performance optimization framework (Pipeleon) for P4 programmable SmartNICs, see Fig. 46. The framework uses profile-guided optimizations to adapt the P4 program based on the runtime profiles (e.g., traffic patterns, and table entries). The input to this framework is a P4 program which is then partitioned into smaller code snippets called pipelets. The framework leverages the reconfigurability of the SmartNICs (e.g., those that follow the disaggregated dRMT architecture [220], [221]) to realize a more efficient implementation. The framework was tested with BlueField2 and Agilio CX SmartNICs and the results show that the optimizations significantly improve the SmartNIC performance in various use cases by up to 5x. Due to such results, it would be beneficial to improve the existing P4 compilers to be tailored to SmartNICs and to consider runtime profiles.

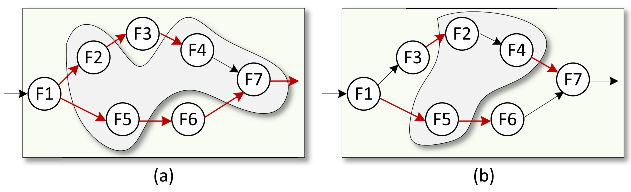


FIGURE 47. (a) Non-optimized placement; (b) optimized placement. The inter-device transmissions (red arrows) between SmartNIC and CPU lead to additional element graph latency. Reproduced from [213].

C. COMPLEX FUNCTIONS OFFLOADING

Effectively utilizing SmartNICs for running offloaded functions presents several challenges. First, SmartNICs have limited computational and memory resources, which restricts the number of functions that can be accommodated on them. Second, although it is technically possible to host switching exclusively on the SmartNIC, doing so incurs considerable latency costs for packets moving between the functions deployed on the SmartNIC and those on the host. This is due to the overhead caused by the multiple traversals across the host PCI bus. Third, distributing the functions between the host and the SmartNIC introduces management challenges.

Current and Future Initiatives: Le et al. [212] presented UNO, a system that splits switching between the host software and the SmartNIC. It uses linear programming formulation to determine the optimal placement for functions. UNO uses the traffic pattern and the load of the function as input. The experiments show that the savings in processors is up to eight host cores. UNO also reduces power by 2x. Another work by Wang et al. [213] optimizes the placement of functions according to the processing and the transmission latency. The system analyzes the dependencies and formulates the partition and placement problem using 0-1 linear programming. The system minimizes The inter-device transmissions between the SmartNIC and the CPU, see Fig. 47.

D. PERFORMANCE UNPREDICTABILITY

When offloading a function to SmartNICs, developers must refactor the core logic to align with the underlying hardware. Determining the optimal offloading strategy may not be straightforward. Moreover, the performance of ported functions can vary among developers, relying heavily on

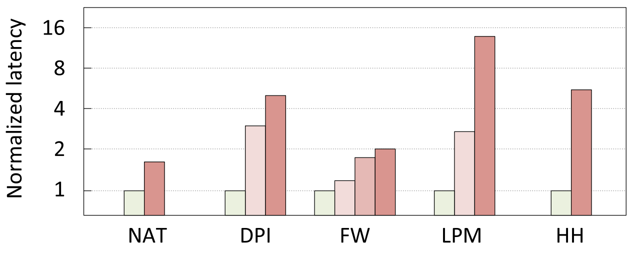


FIGURE 48. Normalized latency for different implementations of functions: Network Address Translation (NAT), DPI, Firewall (FW), LPM, Heavy Hitter (HH). Reproduced from [215].

their understanding of NIC capabilities, see Fig. 48. For instance, using the flow cache can offer orders of magnitude improvement in latency compared to DRAM [215]. This is entirely related to how the programmer implements the code. The performance is also influenced by traffic workloads (e.g., flow volumes, packet sizes, arrival rates). Additional functions on the SmartNIC can pose further challenges, particularly with memory-intensive functions potentially impacting cache utilization for others, and compute-intensive functions potentially causing head-of-line blocking at accelerators [215]. All these factors often lead to unexpected performance fluctuations when migrating a function to a SmartNIC. While benchmarking the program will produce performance results, it requires that the program be already developed on the SmartNIC.

Current and Future Initiatives: Performance prediction can help the developer gain insight prior to porting the code to the hardware. Clara [215] predicts the performance of an unported function on a hypothetical SmartNIC target. Initially, it constructs a model for a given SmartNIC. Then, it creates performance profiles for that SmartNIC by conducting hardware microbenchmarks, which encompass tests on memory latency, accelerator throughput, etc. Clara then creates a code and examines it to identify segments that could be fully offloaded to the SmartNIC. It evaluates the optimal mapping by incorporating constraints derived from the logical NIC model, performance parameters, and code segments. By resolving these constraints, Clara can establish a mapping that optimizes performance after porting. Finally, Clara tests with a PCAP file and assesses how packets would traverse the mapping, thereby providing predictions regarding latency and throughput.

E. POOR SECURITY ISOLATION

Commodity SmartNICs suffer from poor isolation between offloaded functions and between functions and data center operators [214]. This limitation is a result of the limited access controls on the NIC memory and the absence of virtualization for hardware accelerators. These shortcomings compromise the robustness and security of individual functions, especially in a multi-tenant environment. Additionally, any buggy or compromised code within the NIC poses a risk to all other functions running on it. Concrete attacks on popular SmartNICs including packet corruption, DPI rules stealing, and IO bus denial of service, are presented in [214].

Current and Future Initiatives: Zhou et al. [214] proposed S-NIC, a hardware design that enforces disaggregation between resources. S-NIC isolates functions at both the ISA level and the microarchitectural level. This ensures integrity and confidentiality, as well as mitigating against side-channel attacks. The design is cost-effective and requires minimal changes to the hardware (e.g., die area). However, it still incurs modest degradation in the performance. Future work could explore alternative architectures that have less impact on performance, or other software-based techniques to isolate the resources.

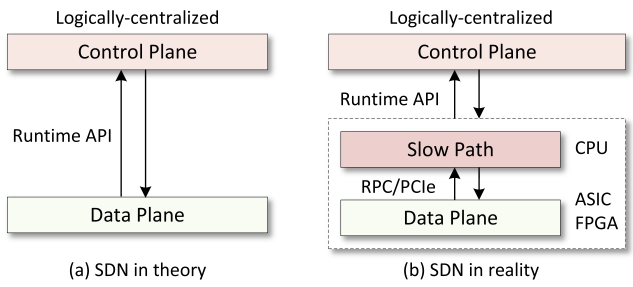


FIGURE 49. (a) SDN in theory; (b) SDN in reality. Reproduced from [211].

F. SLOW PATH BOTTLENECK

Over recent years, there has been a continuous improvement in the performance of packet-processing data planes, leading to their predominant implementation in hardware such as SmartNICs and programmable switches. Yet, there has been a lack of focus on the slow path, the interface between the data plane and the control plane, which is traditionally considered non-performance critical. The slow path is responsible for handling a subset of traffic that requires special processing (complex control flow, compute, memory resources). These tasks cannot be executed on the data plane, see Fig. 49. The slow path is executed on the CPU cores, whether on the host or the SmartNIC.

Lately, the slow path is becoming a major bottleneck, driven by the surge in physical network bandwidth and the increasing complexity of network topologies. There is a growth in slow-path traffic in tandem with user traffic.

Current and Future Initiatives: There is a need to re-evaluate the current approach to balancing workload distribution between the data plane and the slow path. Zulfiqar et al. [211] articulated the limitations of the current slow path and argued that the solution is to have a domain-specific accelerator for the slow path. A challenge with creating such an accelerator is to design a generic architecture with common primitives that support most of the slow path use cases. Ideally, the accelerator would have predictable response times, fast table updates, and support large memory pools. Further, the paper advocates extending the match-action model found in most packet processing devices to match-compute for the slow path.

G. ML OFFLOAD COMPLEXITY

Offloading the training or the inference in ML from the CPU/GPU to SmartNICs comes with a set of challenges that limit the scalability and innovation of the deployed models.

- Accuracy vs Compatibility tradeoff. Some hardware architectures do not support floating-point numbers and complex operations, which are required by advanced ML models, such as neural networks. Workarounds that are proposed to overcome these limitations come at the expense of sacrificing the accuracy of the ML model.

TABLE 12. Abbreviations used in this article.

Abbreviation	Term
ACL	Access Control List
AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
ANOVA	Analysis of Variance
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BCC	BPF Compiler Collection
BPF	Berkeley Packet Filter
CLI	Command Line Interface
CMS	Count-min Sketch
CPU	Central Processing Unit
DNN	Deep Neural Network
DIP	Dynamic IP
DOCA	Data Center-on-a-Chip Architecture
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
DPU	Data Processing Unit
DRAM	Dynamic Random Access Memory
eBPF	Extended Berkeley Packet Filter
ESnet	Energy Sciences Network
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Units
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HDL	Hardware Description Language
HPC	High Performance Computing
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IP	Internet Protocol
IPDK	Infrastructure Programmer Development Kit
IPU	Infrastructure Processing Unit
IPS	Intrusion Prevention System
IPSec	Internet Protocol Security
IT	Information Technology
JBOF	Just a Bunch of Flash
KPI	Key Performance Indicators
kTLS	Kernel TLS
LAN	Local Area Network
LUT	Lookup Table
LPM	Longest Prefix Matching
LSB	Least Significant Bit
MBR	Maximum Bit Rate
ML	Machine Learning
NAS	Network Attached Storage
NAT	Network Address Translation
NFV	Network Function Virtualization
NGFW	Next-Generation Firewall
NIC	Network Interface Card
NLP	Natural Language Processing
NVMe	Non-Volatile Memory Express
NVMe-oF	Non-Volatile Memory Express over Fabric
OFS	Open FPGA Stack
OPAE	Open Programmable Acceleration Engine
OPI	Open Programmable Infrastructure
OS	Operating System
OvS	Open vSwitch
P4	Programming Protocol-independent Packet Processor
PCIe	Peripheral Component Interconnect Express
PISA	Protocol Independent Switch Architecture
PMD	Poll Mode Driver
PNA	Portable NIC Architecture
PSA	Portable Switch Architecture
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RDMA	Remote Direct Memory Access
RPC	Remote Procedure Call
RSS	Receive Side Scaling
RTL	Register Transfer Level
SAN	Storage Area Network
SDK	Software Development Kit
SDN	Software Defined Network

TABLE 12. ***(Continued.)*** **Abbreviations used in this article.**

Abbreviation	Term
SoC	System on a Chip
SPAN	Switched Port Analyzer
SPDK	Storage Performance Development Kit
SSD	Solid State Drives
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TM	Traffic Manager
TRNG	True Random Number Generator
TSO	TCP Segmentation Offload
uBPF	Userspace BPF
UE	User Equipment
UPF	User Plane Function
URL	Uniform Resource Locator
VIP	Virtual IP
VM	Virtual Machine
VPP	Vector Packet Processor
VTEP	VXLAN Tunnel End Point
VXLAN	Virtual Extensible LAN
XDP	eXpress Data Path
xPU	Auxiliary Processing Unit

- Restriction on the adopted ML algorithm. Despite the continuous exploration of deploying ML models, such as neural networks and decision trees in SmartNICs, a multitude of algorithms, such as Principal Component Analysis (PCA), Genetic Algorithms, are yet to be explored. Additionally, models that are currently deployed are static and any update to the model requires temporarily halting the programmable network device until the new model is compiled and pushed.
- Flexibility of aggregate functions: In the context of training ML models, the traditional aggregate functions are ‘min’, ‘max’, ‘count’, ‘sum’, and ‘avg’. However, over time, several approaches started adopting and providing user-defined aggregate functions. Implementing such functions over some hardware architectures used in SmartNICs is not straightforward.

Current and Future Initiatives: Migration of functionality is one technique that can overcome the restrictions of updating the data plane on the fly. For instance, before the programmable network processor is updated, its functionalities are migrated to another device so that network communication is not interrupted. To deal with the lack of support of floating-points, approaches such as [219] translate floating-point numbers to integers using quantization (i.e., a fixed-point representation of decimal numbers). Such technique can also be used in complex neural network models that need to be simplified to fit in the data plane. To reduce communication overhead, Ma et al. [188] compresses the parameters (i.e., gradients) before sharing them in the network. Such approaches can enhance network performance, especially when numerous networking devices are cooperating.

H. LACK OF TRAINING RESOURCES

There is an evident lack of detailed documentation and training resources that adequately cover SmartNIC programming and configuration. While some vendors may provide reference applications, basic documentation, and training courses (e.g., [222]), they often fall short of providing the in-depth explanations and hands-on experience that developers need. This makes it difficult for newcomers to understand the intricacies of SmartNIC development and configuration.

Current and Future Initiatives: To address this issue, it is essential for vendors to invest in creating comprehensive training materials, including detailed documentation, tutorials, and hands-on labs. These resources should cover various aspects of SmartNIC programming and configuration, from basic concepts to advanced techniques. Additionally, vendors could offer interactive online courses or workshops led by experienced instructors to provide personalized guidance and support for learners. Some YouTube channels are posting the latest advances and updates on SmartNICs (e.g., STH [216], SNIA [217], OPI [218]). However, they are still not comprehensive enough to allow a beginner to start experimenting with SmartNICs.

XII. CONCLUSION

The evolution of computing has encountered significant challenges with the end of Moore’s Law and Dennard Scaling. The emergence of SmartNICs, which combine various domain-specific processors, represents a pivotal shift towards offloading infrastructure tasks and improving network efficiency. This paper has filled a critical void in the literature by providing a comprehensive survey of SmartNICs, encompassing their evolution, architectures, development environments, and applications. The paper has delineated the wide array of functions offloaded to SmartNICs, spanning network, security, storage, and compute tasks. The paper has also discussed the challenges associated with SmartNIC development and deployment, and pinpointed key research initiatives and trends that could be explored in the future. Evidence suggests that SmartNICs are poised to become integral components of every network infrastructure. Smaller networks, which often lack deep technical expertise, can leverage SmartNICs for offloading routine infrastructure tasks. On the other hand, larger and research-oriented networks, with experienced developers, will leverage SmartNICs for offloading complex tasks that are not well-suited for general-purpose CPUs.

REFERENCES

[1] G. E. Moore, “Cramming more components onto integrated circuits,” *Proc. IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.
[2] G. Moore, “Progress in digital integrated electronics,” in *Proc. Electron Devices Meeting*, 1975, pp. 11–13.
[3] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, “Design of ion-implanted MOSFET’s with very small physical dimensions,” *Proc. IEEE*, vol. 87, no. 4, pp. 668–678, Apr. 1999.
[4] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2011.
[5] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in *Proc. Spring Joint Comput. Conf. AFIPS (Spring)*, Apr. 1967, pp. 483–485.
[6] J. Faircloth, *Enterprise Applications Administration: The Definitive Guide to Implementation and Operations*. San Mateo, CA, USA: Morgan Kaufmann, 2013.

[7] S. Ibanez, M. Shahbaz, and N. McKeown, "The case for a network fast path to the CPU," in *Proc. 18th ACM Workshop Hot topics Netw.*, Nov. 2019, pp. 52–59.
[8] M. Metz. (2022). *SmartNICs and Infrastructure Acceleration Report 2022*. AvidThink. [Online]. Available: https://avidthink.com/announcements/smartnics-infrastructure-acceleration-report-2022/
[9] A. Ageev, M. Foroushani, and A. Kaufmann, "Exploring domain-specific architectures for network protocol processing," in *Proc. Cloud@MICRO Virtual Workshop*, Oct. 2021. [Online]. Available: https://cloudmicroworkshop.github.io/
[10] E. Tell, "A domain specific DSP processor," Institutionen för Systemteknik, Linkoping, Tech. Rep. LiTH-ISY-EX-3209, 2001.
[11] D. Caetano-Anolles. (2022). *Hardware—Optimizations—SSD—CPU—GPU—FPGA—TPU*. GATK. [Online]. Available: https://gatk.broadinstitute.org/hc/en-us/articles/360035531632-Hardware-optimizations-SSD-CPU-GPU-FPGA-TPU
[12] Google. *Encryption in Transit*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/436vh9jh
[13] J. Morra. *Is This the Future of the SmartNIC?*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/ydru5bcp
[14] Microsoft. *Azure SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4sj7m7mp
[15] S. Schweitzer, "Architectures, boards, chips and software," presented at the SmartNIC Summit, 2023. [Online]. Available: https://smartnicssummit.com/proceeding_files/a0q5f000003HkUt/20230614_PLEN_Schweitzer.PDF
[16] AMD. (2022). *AMD Collaborates With the Energy Sciences Network on Launch of Its Next-Generation, High-Performance Network to Enhance Data-Intensive Science*. [Online]. Available: https://tinyurl.com/ycyb382t
[17] VMware. *DPU-based Acceleration for NSX*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/238v6j5h
[18] Palo Alto Networks. (2022). *Intelligent Traffic Offload Uses SmartNIC/DPU for Hyperscale Security*. [Online]. Available: https://tinyurl.com/d322nda7
[19] Juniper Networks. (2021). *SmartNICs Accelerate the New Network Edge*. [Online]. Available: https://tinyurl.com/2uh6uh7t
[20] S. Vural. (2021). *SmartNICs in Telco: Benefits and Use Cases*. [Online]. Available: https://tinyurl.com/8amw8s74
[21] D. Z. tootaghaj, A. Mercian, V. Adarsh, M. Sharifian, and P. Sharma, "SmartNICs at edge for transient compute elasticity," in *Proc. 3rd Int. Workshop Distrib. Mach. Learn.*, Dec. 2022, pp. 9–15.
[22] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "In-network machine learning using programmable network devices: A survey," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 2, pp. 1171–1200, 2nd Quart., 2024.
[23] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "FABRIC: A national-scale programmable experimental network infrastructure," *IEEE Internet Comput.*, vol. 23, no. 6, pp. 38–47, Nov. 2019.
[24] GEANT. *GEANT Testbed*. Accessed: Aug. 2, 2024. [Online]. Available: https://geant.org/
[25] GEANT. (2023). *High-Performance Flow Monitoring Using Programmable Network Interface Cards*. [Online]. Available: https://resources.geant.org/wp-content/uploads/2023/02/GN4-3_White-Paper_High-Performance-Flow-Monitoring-Using-Programmable-NICs.pdf
[26] E. D. C. Pontes, M. Martinello, C. K. Dominicini, M. Schwarz, M. Ribeiro, E. S. Borges, I. Brito, J. Bezerra, and M. Barcellos, "FABRIC testbed from the eyes of a network researcher," in *Proc. Anais do II Workshop de Testbeds (WTESTBEDS)*, Aug. 2023, pp. 38–49.
[27] D. Cerovic, V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "Fast packet processing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3645–3676, 4th Quart., 2018.
[28] Linux Found. *DPDK*. [Online]. Available: https://www.dpdk.org/
[29] Ntop Eng. *PF_RING: High-Speed Packet Capture, Filtering and Analysis*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yzwc4t35
[30] T. Marian, K. S. Lee, and H. Weatherspoon, "NetSlices: Scalable multi-core packet processing in user-space," in *Proc. ACM/IEEE Symp. Architectures for Netw. Commun. Syst. (ANCS)*, Oct. 2012, pp. 27–38.
[31] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *Proc. 21st USENIX Secur. Symp. (USENIX Security)*, 2012, pp. 101–112.
[32] E. Freitas, A. T. de Oliveira Filho, P. R. X. do Carmo, D. Sadok, and J. Kelner, "A survey on accelerating technologies for fast network packet processing in Linux environments," *Comput. Commun.*, vol. 196, pp. 148–166, Dec. 2022.
[33] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi, "Survey of performance acceleration techniques for network function virtualization," *Proc. IEEE*, vol. 107, no. 4, pp. 746–764, Apr. 2019.
[34] X. Fei, F. Liu, Q. Zhang, H. Jin, and H. Hu, "Paving the way for NFV acceleration: A taxonomy, survey and future directions," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–42, Jul. 2021.
[35] P. Shantharama, A. S. Thyagaturu, and M. Reisslein, "Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 8, pp. 132021–132085, 2020.
[36] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira, "Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications," *ACM Comput. Surv.*, vol. 53, no. 1, pp. 1–36, Jan. 2021.
[37] L. Rosa, L. Foschini, and A. Corradi, "Empowering cloud computing with network acceleration: A survey," *IEEE Commun. Surveys Tuts.*, early access, Mar. 14, 2024, doi: 10.1109/COMST.2024.3377531.
[38] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87094–87155, 2021.
[39] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with P4: Fundamentals, advances, and applied research," *J. Netw. Comput. Appl.*, vol. 212, Mar. 2023, Art. no. 103561.
[40] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–36, May 2022.
[41] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, "A survey on data plane flexibility and programmability in software-defined networking," *IEEE Access*, vol. 7, pp. 47804–47840, 2019.
[42] W. L. da Costa Cordeiro, J. A. Marques, and L. P. Gaspary, "Data plane programmability beyond OpenFlow: Opportunities and challenges for network and service operations and management," *J. Netw. Syst. Manage.*, vol. 25, no. 4, pp. 784–818, Oct. 2017.
[43] Y. Gao and Z. Wang, "A review of P4 programmable data planes for network security," *Mobile Inf. Syst.*, vol. 2021, pp. 1–24, Nov. 2021.
[44] A. AlSabeh, J. Khoury, E. Kfoury, J. Crichigno, and E. Bou-Harb, "A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment," *Comput. Netw.*, vol. 207, Apr. 2022, Art. no. 108800.
[45] X. Chen, C. Wu, X. Liu, Q. Huang, D. Zhang, H. Zhou, Q. Yang, and M. K. Khan, "Empowering network security with programmable switches: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 3, pp. 1653–1704, 3rd Quart., 2023.
[46] R. Parizotto, B. L. Coelho, D. C. Nunes, I. Haque, and A. Schaeffer-Filho, "Offloading machine learning to programmable data planes: A systematic survey," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 1–34, Jan. 2024.
[47] W. Quan, Z. Xu, M. Liu, N. Cheng, G. Liu, D. Gao, H. Zhang, X. Shen, and W. Zhuang, "AI-driven packet forwarding with programmable data plane: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 762–790, 1st Quart., 2023.
[48] J. Gomez, E. F. Kfoury, J. Crichigno, and G. Srivastava, "A survey on TCP enhancements using P4-programmable devices," *Comput. Netw.*, vol. 212, Jul. 2022, Art. no. 109030.
[49] S. Han, S. Jang, H. Choi, H. Lee, and S. Pack, "Virtualization in programmable data plane: A survey and open challenges," *IEEE Open J. Commun. Soc.*, vol. 1, pp. 527–534, 2020.
[50] J. A. Brito, J. I. Moreno, L. M. Contreras, M. Alvarez-Campana, and M. B. Caamaño, "Programmable data plane applications in 5G and beyond architectures: A systematic review," *Sensors*, vol. 23, no. 15, p. 6955, Aug. 2023.
[51] A. Mazloum, E. Kfoury, J. Gomez, and J. Crichigno, "A survey on rerouting techniques with P4 programmable data plane switches," *Comput. Netw.*, vol. 230, Jul. 2023, Art. no. 109795.
[52] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, "A survey of fast recovery mechanisms in the data plane," *Authorea Preprints*, vol. 23, pp. 1253–1301, Jan. 2023.

[53] NVIDIA. *NVIDIA Mellanox BlueField-2 Data Processing Unit (DPU)*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yrky7ee5

[54] AMD. *Pensando DSC2-200 Distributed Services Card*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yr6eeez6

[55] AMD. *Xilinx Alveo SN1000 SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/pxacmnd9

[56] N. McKeown. *Why Does the Internet Need a Programmable Forwarding Plane*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/ffajhk9y

[57] J. Xing, Y. Qiu, K.-F. Hsu, S. Sui, K. Manaa, O. Shabtai, Y. Piasetzky, M. Kadosh, A. Krishnamurthy, T. S. E. Ng, and A. Chen, "Unleashing SmartNIC packet processing performance in P4," in *Proc. ACM SIGCOMM Conf.*, Sep. 2023, pp. 1028–1042.

[58] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 158–169.

[59] S. H. Venkata Krishnan, "Enabling applications to exploit SmartnICS and FPGAs," OpenFabrics Alliance, Austin, TX, USA, Tech. Rep., 2019.

[60] NVIDIA. *ConnectX-5 EN Card*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/nhcf26nr

[61] NVIDIA. *ConnectX-6 LX 25/50G Ethernet SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4at7npy5

[62] NVIDIA. *ConnectX-6 Dx 200G Ethernet SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2e59ts66

[63] NVIDIA. *ConnectX-7 400G Adapters*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/hndz6yxm

[64] Achronix. *Vectorpath Accelerator Card*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yc7xachz

[65] AMD. *Xilinx Alveo U50 Data Center Accelerator Card*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/nhbe4xbd

[66] AMD. *Xilinx Alveo U55C Data Center Accelerator Cards*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/mr4887yw

[67] AMD. *Alveo U200 and U250 Data Center Accelerator Cards*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2p9tzav3

[68] AMD. *Alveo U280 Data Center Accelerator Card*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/bdfzke7z

[69] Napatech. *NT200A02 SmartNIC With Link-Capture Software*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/y4xbyypy

[70] Silicom. *Silicom FPGA SmartNIC N501x Series*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4s9mwr88

[71] Silicom. *Silicom N5110A SmartNIC Intel Based*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yskzrzah

[72] Silicom. *FPGA SmartNIC FB2CDG1@AGM39D-2 Intel Based*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3rsbur47

[73] Silicom. *FPGA SmartNIC N6010/6011 Intel Based*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3syps38s

[74] Silicom. *FB4XXVG@Z21D TimeSync SmartNIC FPGA Xilinx Based*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4vdbp3jd

[75] NVIDIA. *Mellanox Innova-2 Flex Open Programmable SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3wdy3hxd

[76] AMD. *Pensando Giglio Data Processing Unit*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yst9b77m

[77] AMD. *Pensando DSC2-100 100G 2p QSFP56 DPU and DSC2-25 10/25G 2p SFP56 DPU Distributed Services Cards for VMware VSphere Distributed Services Engine*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/38ax5jkb

[78] Asterfusion. *Helium EC2004Y*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3bkpn6yv

[79] Asterfusion. *Helium Ec2002p*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/psfr4w6d

[80] Broadcom. *Stingray PS225 SmartNIC Adapters*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/5f3rpu45

[81] Intel. *Infrastructure Processing Unit (Intel IPU) ASIC E2000*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/5d3rbjfb

[82] Marvell. *Marvell LiquidIO III*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/a7r69vpc

[83] Netronome. *Agilio FX 2x10GbE SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://www.corigine.com/UploadFiles/pdf/2021-07-22/124033028154158.pdf

[84] Netronome. *Agilio CX 2x40GbE SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://www.corigine.com/UploadFiles/pdf/2021-07-22/124033028154158.pdf

[85] NVIDIA. *NVIDIA BlueField-3 Networking Platform*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3e5v2xd2

[86] AMD. *Xilinx Alveo U25N SmartNIC*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2dwz7dxe

[87] AMD. *Alveo U45N Data Center Accelerator Card*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/mvtbshy3

[88] Intel. *FPGA Product Catalog*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/ykvxkj3c

[89] Napatech. *SmartNIC and IPU Hardware Portfolio*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yxcbx2p9

[90] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartNICs using iPipe," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 318–333.

[91] T. Cui, W. Zhang, K. Zhang, and A. Krishnamurthy, "Offloading load balancers onto SmartNICs," in *Proc. 12th ACM SIGOPS Asia–Pacific Workshop Syst.*, Aug. 2021, pp. 56–62.

[92] N. Systems, "Programming netronome Agilio® SmartNICs," Netronome, Cranberry township, PA, USA, Whitepaper 17, 2017, pp. 1–14.

[93] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[94] P4 Lang. Consortium. *P4_14 Language Specification*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/hzujjzt7

[95] P4 Lang. Consortium. *P4_16 Language Specification*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/5fvfnd8t

[96] *P4 Portable NIC Architecture (PNA)*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3v6etke2

[97] AMD. (2023). *Xilinx Vivado Design Suite 2023*. Accessed: Aug. 2, 2024. [Online]. Available: https://www.xilinx.com

[98] Intel. *Intel P4 Suite for FPGA*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/42rztah2

[99] Linux Found. *DPDK Supported Hardware*. Accessed: Aug. 2, 2024. [Online]. Available: https://core.dpdk.org/supported/

[100] Linux Found. *DPDK Pipeline Application*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/udutp3jf

[101] Linux Found. *Generic Flow API (rte_flow) Documentation*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3pwwnnx2

[102] S. Horman, "OvS hardware offload with TC flower," presented at Netronome, 2017.

[103] NVIDIA. *DOCA Flow*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/bdfx7u98

[104] NVIDIA. (2019). *Mellanox ASAP2 Accelerated Switching and Packet Processing, ConnectX ASAP2—Accelerated Switcha Packet Processing*. [Online]. Available: https://network.nvidia.com/files/doc-2020/sb-asap2.pdf

[105] NVIDIA. *DOCA Developer Guide*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2usa47hs

[106] Intel. *P4 Insight*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2v2xajrf

[107] AMD. *Xilinx Vitis Networking P4*. Accessed: Aug. 2, 2024. [Online]. Available: https://docs.amd.com/r/en-US/ug1308-vitis-p4-user-guide

[108] AMD. *Xilinx XRT and Vitis Platform Overview*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/y5jdsypx

[109] Intel. *Intel Open FPGA Stack*. Accessed: Aug. 2, 2024. [Online]. Available: https://www.intel.com/

[110] Linux Found. *Open Programmable Infrastructure Project*. Accessed: Aug. 2, 2024. [Online]. Available: https://opiproject.org/

[111] Linux Found. *IPDK Documentation*. Accessed: Aug. 2, 2024. [Online]. Available: https://ipdk.io/documentation/

[112] Linux Found. *Sonic-Dash*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/utcjchme

[113] L. Xin. (2022). *SONiC, Programmability & Acceleration*. [Online]. Available: https://tinyurl.com/musxey96

[114] J. Thönes, "Microservices," *IEEE Softw.*, vol. 32, no. 1, p. 116, Jan. 2015.

[115] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, Nov. 2010, pp. 267–280.

[116] Cisco. (2015). *Cisco Global Cloud Index 2015–2020*. [Online]. Available: https://tinyurl.com/2ery68x4

[117] V. Stafford, "Zero trust architecture," NIST, Gaithersburg, MD, USA, Tech. Rep. 800-207, 2020.

[118] D. Basak, R. toshniwal, S. Maskalik, and A. Sequeira, "Virtualizing networking and security in the cloud," *ACM SIGOPS Operating Syst. Rev.*, vol. 44, no. 4, pp. 86–94, Dec. 2010.

[119] NVIDIA. *DOCA Open VSwitch Layer-4 Firewall*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/bdfctkaj

[120] AMD. *Achieve High Throughput: A Case Study Using a Pensando Distributed Services Card With P4 Programmable Software-Defined Networking Pipeline*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yj9ttvnh

[121] M. Gonen. *Accelerating the Suricata IDS/IPS With NVIDIA BlueField DPUs*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/ys8n6mmz

[122] Zeek Project. *Zeek, an Open Source Network Security Monitoring tool*. Accessed: Aug. 2, 2024. [Online]. Available: https://zeek.org/

[123] Open Inf. Secur. Found. *Suricata*. Accessed: Aug. 2, 2024. [Online]. Available: https://suricata.io/

[124] Cisco. *Snort—Network Intrusion Detection and Prevention System*. Accessed: Aug. 2, 2024. [Online]. Available: https://www.snort.org/

[125] Z. Zhao, H. Sadok, N. Atre, J. Hoe, V. Sekar, and J. Sherry, "Achieving 100 Gbps intrusion prevention on a single server," in *Proc. 14th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2020, pp. 1083–1100.

[126] J. Chen, X. Zhang, T. Wang, Y. Zhang, T. Chen, J. Chen, M. Xie, and Q. Liu, "Fidas: Fortifying the cloud via comprehensive FPGA-based offloading for intrusion detection: Industrial product," in *Proc. 49th Annu. Int. Symp. Comput. Archit.* Chicago, IL, USA: Industrial, Jun. 2022, pp. 1029–1041.

[127] Y. Zhao, G. Cheng, Y. Duan, Z. Gu, Y. Zhou, and L. Tang, "Secure IoT edge: Threat situation awareness based on network traffic," *Comput. Netw.*, vol. 201, Dec. 2021, Art. no. 108525.

[128] S. Panda, Y. Feng, S. G. Kulkarni, K. K. Ramakrishnan, N. Duffield, and L. N. Bhuyan, "SmartWatch: Accurate traffic analysis and flow-state tracking for intrusion prevention using SmartNICs," in *Proc. 17th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2021, pp. 60–75.

[129] M. Wu, H. Matsutani, and M. Kondo, "ONLAD-IDS: ONLAD-based intrusion detection system using SmartNIC," in *Proc. IEEE 24th Int. Conf. High Perform. Comput. Commun. 8th Int. Conf. Data Sci. Syst. 20th Int. Conf. Smart City 8th Int. Conf. Dependability Sensor, Cloud Big Data Syst. Appl. (HPCC/DSS/SmartCity/DependSys)*, Dec. 2022, pp. 546–553.

[130] K. Tasdemir, R. Khan, F. Siddiqui, S. Sezer, F. Kurugollu, and A. Bolat, "An investigation of machine learning algorithms for high-bandwidth SQL injection detection utilising BlueField-3 DPU technology," in *Proc. IEEE 36th Int. System-on-Chip Conf. (SOCC)*, Sep. 2023, pp. 1–6.

[131] S. Miano, R. Doriguzzi-Corin, F. Risso, D. Siracusa, and R. Sommese, "Introducing SmartNICs in server-based data plane processing: The DDoS mitigation use case," *IEEE Access*, vol. 7, pp. 107161–107170, 2019.

[132] Open Inf. Secur. Found. *Ignoring Traffic*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/f2kn3snm

[133] R. Yavatkar. *SmartNICs Accelerate the New Network Edge*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2af6yfp3

[134] M. Ceška, V. Havlena, L. Holík, J. Korenek, O. Lengál, D. Matoušek, J. Matoušek, J. Semric, and T. Vojnar, "Deep packet inspection in FPGAs via approximate nondeterministic automata," in *Proc. IEEE 27th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, Apr. 2019, pp. 109–117.

[135] Y.-H. Yang and V. Prasanna, "High-performance and compact architecture for regular expression matching on FPGA," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 1013–1025, Jul. 2012.

[136] D. Matoušek, J. Korenek, and V. Puš, "High-speed regular expression matching with pipelined automata," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2016, pp. 93–100.

[137] D. Luchaup, L. De Carli, S. Jha, and E. Bach, "Deep packet inspection with DFA-trees and parametrized language overapproximation," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2014, pp. 531–539.

[138] M. Češka, V. Havlena, L. Holík, O. Lengál, and T. Vojnar, "Approximate reduction of finite automata for high-speed network intrusion detection," *Int. J. Softw. tools Technol. Transf.*, vol. 22, no. 5, pp. 523–539, Oct. 2020.

[139] N. Diamond, S. Graham, and G. Clark, "Securing InfiniBand traffic with BlueField-2 data processing units," in *Proc. Int. Conf. Crit. Infrastructure Protection*, 2022, pp. 277–300.

[140] Q. Su, S. Wu, Z. Niu, R. Shu, P. Cheng, Y. Xiong, Z. Liu, and H. Xu, "Meili: Enabling SmartNIC as a service in the cloud," 2023, *arXiv:2312.11871*.

[141] T. T. Bar Tuaf, Tal Gilboa. (2020). *KTLS Offload Performance Enhancements for Real-Life Applications*. [Online]. Available: https://tinyurl.com/24ep7pwc

[142] D. Kim, S. Lee, and K. Park, "A case for SmartNIC-accelerated private communication," in *Proc. 4th Asia–Pacific Workshop Netw.*, Aug. 2020, pp. 30–35.

[143] F. Novais and F. L. Verdi. *Unlocking Security to the Board: An Evaluation of SmartNIC-Driven TLS Acceleration With KTLS*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2p92nsnj

[144] J. Zhao, M. Neves, and I. Haque, "On the (dis)advantages of programmable NICs for network security services," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2023, pp. 1–9.

[145] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer," in *Proc. Hotnets*, 2009.

[146] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 120–125.

[147] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff, "Revisiting the open vSwitch dataplane ten years later," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 245–257.

[148] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, and P. Shelar, "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Networked Syst. design Implement. (NSDI)*, 2015.

[149] VMware. *VSphere Distributed Switch*. [Online]. Available: https://tinyurl.com/2bpwzubd

[150] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, *Virtual Extensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks*, document RFC 7348, 2014.

[151] J. Gross, I. Ganga, and T. Sridhar, *Geneve: Generic Network Virtualization Encapsulation*, document RFC 8926, 2020.

[152] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, *Generic Routing Encapsulation (GRE)*, document RFC 2784, 2000.

[153] I. Burstein, "NVIDIA data center processing unit (DPU) architecture," in *Proc. IEEE Hot Chips 33 Symp. (HCS)*, Aug. 2021, pp. 1–20.

[154] J. Weerasinghe and F. Abel, "On the cost of tunnel endpoint processing in overlay virtual networks," in *Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput.*, Dec. 2014, pp. 756–761.

[155] L. Luo, "Towards converged SmartNIC architecture for bare metal & public clouds," in *Proc. APNet Ind. Talks*, 2018, pp. 1–28.

[156] NVIDIA. *Virtual Switch on DPU*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/5n8eb6bz

[157] B. Claise, "Cisco systems NetFlow services export version 9," RFC Editor, USA, Tech. Rep. RFC 3954, 2004.

[158] B. Claise, M. Fullmer, P. Calato, and R. Penno. (2005). *IPFIX Protocol Specification*. Internet-Draft. [Online]. Available: https://www.ietf.org/proceedings/60/slides/ipfix-4.pdf

[159] P4 Work. Group. *In-band Network Telemetry (INT) Dataplane Specification*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4x9shr45

[160] F. Brockners, S. Bhandari, D. Bernier, and T. Mizrahi, "In situ operations, administration, and maintenance (IOAM) deployment," RFC Editor, USA, Tech. Rep. RFC 9378, 2023.

[161] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[162] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[163] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey," *Comput. Netw.*, vol. 57, no. 18, pp. 4047–4064, Dec. 2013.

[164] Z. Zeng, L. Cui, M. Qian, Z. Zhang, and K. Wei, "A survey on sliding window sketch for network measurement," *Comput. Netw.*, vol. 226, May 2023, Art. no. 109696.

[165] J. White, J. Kim, M. Baldi, Y. Li, and D. McIntyre. *XPU Accelerator Offload Functions*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/rzyfx5b4

[166] T. Cui, C. Zhao, W. Zhang, K. Zhang, and A. Krishnamurthy, "Laconic: Streamlined load balancers for SmartNICs," 2024, *arXiv:2403.11411*.

[167] X. Huang, Z. Guo, and M. Song, "FGLB: A fine-grained hardware intra-server load balancer based on 100 g FPGA SmartNIC," *Int. J. Netw. Manage.*, vol. 32, no. 6, Nov. 2022, Art. no. e2211.

[168] B. Chang, A. Akella, L. D'Antoni, and K. Subramanian, "Learned load balancing," in *Proc. 24th Int. Conf. Distrib. Comput. Netw.*, Jan. 2023, pp. 177–187.

[169] Z. Ni, C. Wei, T. Wood, and N. Choi, "A SmartNIC-based load balancing and auto scaling framework for middlebox edge server," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2021, pp. 21–27.

[170] H. Tajbakhsh, R. Parizotto, M. Neves, A. Schaeffer-Filho, and I. Haque, "Accelerator-aware in-network load balancing for improved application performance," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2022, pp. 1–9.

[171] R. Durner, A. Varasteh, M. Stephan, C. M. Machuca, and W. Kellerer, "HNLB: Utilizing hardware matching capabilities of NICs for offloading stateful load balancers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[172] Y. Zhang, J. Bi, Z. Li, Y. Zhou, and Y. Wang, "VMS: Load balancing based on the virtual switch layer in datacenter networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1176–1190, Jun. 2020.

[173] H. Krawczyk, "New hash functions for message authentication," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 1995, pp. 301–310.

[174] Linux Found. *Scaling in the Linux Networking Stack*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4fjv42hj

[175] Napatech. *5G User Plane Function Offload*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4jxxeh8t

[176] R. Davis. *NVIDIA BlueField Partner's DPU Storage Solutions and Use Cases*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/2s4kmkrp

[177] Y. Li, A. Kashyap, Y. Guo, and X. Lu, "Characterizing lossy and lossless compression on emerging BlueField DPU architectures," in *Proc. IEEE Symp. High-Performance Interconnects (HOTI)*, Aug. 2023, pp. 33–40.

[178] L. Peter, *DEFLATE Compressed Data Format Specification Version 1.3*, document RFC 1951, 1996.

[179] L. Peter and J. Gailly, *ZLIB Compressed Data Format Specification Version 3.3*, document RFC 1950, 1996.

[180] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao, Z. Chen, and F. Cappello, "SZ3: A modular framework for composing prediction-based error-bounded lossy compressors," *IEEE Trans. Big Data*, vol. 9, no. 2, pp. 485–498, Apr. 2023.

[181] E. de Rothschild. (Jun. 2023). *Ai Insights—Is the Acceleration of the Power of AI Models a Recent Phenomenon?*. [Online]. Available: https://www.linkedin.com/pulse/ai-insights-acceleration-power-models-recent-phenomenon

[182] Z. Ma et al., "BaGuaLu: Targeting brain scale pretrained models with over 37 million cores," in *Proc. 27th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, Apr. 2022, pp. 192–204.

[183] A. Moody, J. Fernandez, F. Petrini, and D. K. Panda, "Scalable NIC-based reduction on large-scale clusters," in *Proc. ACM/IEEE Conf. Supercomputing*, Nov. 2003, p. 59.

[184] A. Santos da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2016, pp. 27–35.

[185] T. Itsubo, M. Koibuchi, H. Amano, and H. Matsutani, "Accelerating deep learning using multiple GPUs and FPGA-based 10GbE switch," in *Proc. 28th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2020, pp. 102–109.

[186] K. Tanaka, Y. Arikawa, T. Ito, K. Morita, N. Nemoto, F. Miura, K. Terada, J. Teramoto, and T. Sakamoto, "Communication-efficient distributed deep learning with GPU-FPGA heterogeneous computing," in *Proc. IEEE Symp. High-Perform. Interconnects (HOTI)*, Aug. 2020, pp. 43–46.

[187] NVIDIA. *NVIDIA DOCA Allreduce Application Guide*. Accessed: Aug. 2, 2024. [Online]. Available: https://docs.nvidia.com/doca/sdk/nvidia+doca+allreduce+application+guide/index.html

[188] R. Ma, E. Georganas, A. Heinecke, S. Gribok, A. Boutros, and E. Nurvitadhi, "FPGA-based AI smart NICs for scalable distributed AI training systems," *IEEE Comput. Archit. Lett.*, vol. 21, no. 2, pp. 49–52, Jul. 2022.

[189] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proc. 18th ACM Workshop Hot topics Netw.*, Nov. 2019, pp. 25–33.

[190] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4->NetFPGA workflow for line-rate packet processing," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2019, pp. 1–9.

[191] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[192] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the AI accelerator?" in *Proc. Morning Workshop In-Network Comput.*, Aug. 2018, pp. 20–25.

[193] J. Liu, A. Dragojević, S. Fleming, A. Katsarakis, D. Korolija, I. Zablotchi, H.-C. Ng, A. Kalia, and M. Castro, "Honeycomb: Ordered key-value store acceleration on an FPGA-based SmartNIC," *IEEE Trans. Comput.*, vol. 73, no. 3, pp. 857–871, Mar. 2024.

[194] Redis. (2024). *The Real-Time Data Platform*. [Online]. Available: https://redis.io/

[195] B. Fitzpatrick. (Aug. 1, 2004). *Distributed Caching with Memcached*. Accessed: Aug. 2, 2024. [Online]. Available: https://www.linuxjournal.com/article/7451

[196] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *Proc. 11th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2014.

[197] C. Mitchell, Y. Geng, and J. Li, "Using one-sided RDMA reads to build a fast, CPU-efficient key-value store," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2013, pp. 103–114.

[198] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 265–306.

[199] A. Kalia, M. Kaminsky, and D. Andersen, "Design guidelines for high performance RDMA systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2016, pp. 437–450.

[200] B. Cassell, T. Szepesi, B. Wong, T. Brecht, J. Ma, and X. Liu, "Nessie: A decoupled, client-driven key-value store using RDMA," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3537–3552, Dec. 2017.

[201] S. Sun, R. Zhang, M. Yan, and J. Wu, "SKV: A SmartNIC-offloaded distributed key-value store," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2022, pp. 1–11.

[202] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proc. 16th USENIX Symp. Networked Syst. Design Implement. (NSDI*, 2019, pp. 1–16.

[203] H.-H. Chen, C.-H. Chang, and S.-H. Hung, "HKVS: A framework for designing a high throughput heterogeneous key-value store with SmartNIC and RDMA," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Oct. 2022, pp. 99–106.

[204] J. Li, Y. Lu, Q. Wang, J. Lin, Z. Yang, and J. Shu, "AlNiCo SmartNIC-accelerated contention-aware request scheduling for transaction processing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 951–966.

[205] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "Xenic: SmartNIC-accelerated distributed transactions," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, Oct. 2021.

[206] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-NIC: Interactive serverless compute on programmable SmartNICs," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 67–77.

[207] Amazon. *Serverless Function, FaaS Service, AWS Lambda*. Accessed: Aug. 2, 2024. [Online]. Available: https://aws.amazon.com/lambda/

[208] Google. *Google Cloud Functions*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/acayx98p

[209] Microsoft. *Azure Functions*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/a7wat88a

[210] Linux Found. *IPDK*. Accessed: Aug. 2, 2024. [Online]. Available: https://ipdk.io/

[211] A. Zulfiqar, B. Pfaff, W. Tu, G. Antichi, and M. Shahbaz, "The slow path needs an accelerator too!" *ACM SIGCOMM Comput. Commun. Rev.*, vol. 53, no. 1, pp. 38–47, Jan. 2023.

[212] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. V. Lakshman, "UNO: Uniflying host and smart NIC offload for flexible packet processing," in *Proc. Symp. Cloud Comput.*, Sep. 2017, pp. 506–519.

[213] S. Wang, Z. Meng, C. Sun, M. Wang, M. Xu, J. Bi, T. Yang, Q. Huang, and H. Hu, "SmartChain: Enabling high-performance service chain partition between SmartNIC and CPU," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–7.

[214] Y. Zhou, M. Wilkening, J. Mickens, and M. Yu, "SmartNIC security isolation in the cloud with S-NIC," in *Proc. 19th Eur. Conf. Comput. Syst.*, 2024, pp. 851–869.

[215] Y. Qiu, Q. Kang, M. Liu, and A. Chen, "Clara: Performance clarity for SmartNIC offloading," in *Proc. 19th ACM Workshop Hot topics Netw.*, Nov. 2020.

[216] The Official ServeTheHome.com YouTube Channel. *Servethehome*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/yc58uapm

[217] The Official SNIA YouTube Channel. *SNIAVideo*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/3bhdb7kd

[218] The Official OPI YouTube Channel. *The Open Programmable Infrastructure*. Accessed: Aug. 2, 2024. [Online]. Available: https://www.youtube.com/@OPI_project

[219] K. A. Simpson and D. P. Pezaros, "Revisiting the classics: Online RL in the programmable dataplane," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2022, pp. 1–10.

[220] J. Xing, K. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen, "Runtime programmable switches," in *Proc. 19th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2022, pp. 651–665.

[221] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, A. Orda, and T. Edsall, "DRMT: Disaggregated programmable switching," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 1–14.

[222] NVIDIA. *Introduction to DOCA for DPUs*. Accessed: Aug. 2, 2024. [Online]. Available: https://tinyurl.com/4tux5eb9

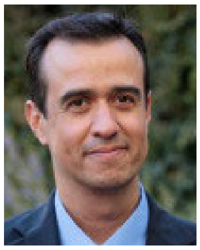
ELIE F. KFOURY (Member, IEEE) received the Ph.D. degree in informatics from the University of South Carolina (USC), in 2023. He was a Research and Teaching Assistant with the Computer Science Department, American University of Science and Technology, Beirut. He is currently an Assistant Professor with the Integrated Information Technology Department, USC. As a member of the Cyberinfrastructure Laboratory, he developed training materials using virtual labs on high-speed networks, TCP congestion control, programmable switches, SDN, and cybersecurity. He is the co-author of a book *High-Speed Networks: A Tutorial*, that is being used nationally for deploying, troubleshooting, and tuning Science DMZ networks. His research interests include P4 programmable data planes, computer networks, cybersecurity, and blockchain.

SAMIA CHOUEIRI received the master's degree in computer and communications engineering with an emphasis in mechatronics engineering from the American University of Science and Technology, Beirut. She is currently pursuing the Ph.D. degree with the College of Engineering and Computing, University of South Carolina (USC). She was a Teaching Assistant and a Laboratory Instructor with the American University of Science and Technology. Her research interests include SmartNICs, P4 switches, cybersecurity, and robotics.

ALI MAZLOUM (Graduate Student Member, IEEE) received the bachelor's degree in computer science from the American University of Beirut (AUB). He is currently pursuing the Ph.D. degree with the College of Engineering and Computing, University of South Carolina (USC), USA. His research interests include P4 programmable data planes, SmartNICs, cybersecurity, network measurements, and traffic engineering.

ALI ALSABEH received the M.S. degree in computer science from the American University of Beirut. He is currently pursuing the Ph.D. degree with the College of Engineering and Computing, University of South Carolina, USA. He was a Graduate Research Assistant and a Teacher Assistant with the American University of Beirut. He is a member of the CyberInfrastructure Laboratory (CI Lab), where he developed training materials for virtual labs on network protocols (BGP and OSPF) and their applications (BGP attributes, BGP hijacking, and IP spoofing), as well as SDN (OpenFlow and interconnecting SDN with legacy networks). His research interests include malware analysis, network security, and P4 programmable switches.

JOSE GOMEZ is currently pursuing the Ph.D. degree with the College of Engineering and Computing, University of South Carolina. He is with the Cyberinfrastructure Laboratory developing a system based on P4 switches to enable programmability in non-programmable networks. His research interests include P4 programmable data planes, TCP congestion control, passive measurements, and buffer sizing.



JORGE CRICHIGNO (Member, IEEE) received the bachelor's degree in electrical engineering from the Catholic University of Paraguay, in 2004, and the Ph.D. degree in computer engineering from The University of New Mexico, in 2009. He is currently a Professor with the College of Engineering and Computing, University of South Carolina (USC). He has over 15 years of experience in the academic and industry sectors. Before joining USC, he was an Associate Professor and the Chair of the Department of Engineering, Northern New Mexico College. His work has been funded by Google, NSF, and the Department of Energy. His research interests include the practical implementation of high-speed networks and network security. These include the design and implementation of high-speed switched networks, TCP optimization, experimental evaluation of congestion control algorithms tailored for friction-free environments, and scalable flow-based intrusion detection systems.