Paving the Way for NFV Acceleration: A Taxonomy, Survey and Future Directions

XINCAI FEI, FANGMING LIU (CORRESPONDING AUTHOR), QIXIA ZHANG, and HAI JIN, National Engineering Research Center for Big Data Technology and System, Key Laboratory of Services Computing Technology and System, Ministry of Education, School of Computer Science and Technology, Huazhong University of Science and Technology, China
HONGXIN HU, Clemson University, USA

As a recent innovation, network functions virtualization (NFV), with its core concept of replacing hardware middleboxes with software network functions (NFs) implemented in commodity servers, promises cost savings and flexibility benefits. However, transitioning NFs from special-purpose hardware to commodity servers has turned out to be more challenging than expected, as it inevitably incurs performance penalties due to bottlenecks in both software and hardware. To achieve performance comparable to hardware middleboxes, there is a strong demand for a speedup in NF processing, which plays a crucial role in the success of NFV. In this article, we study the performance challenges that exist in general-purpose servers and simultaneously summarize the typical performance bottlenecks in NFV. Through reviewing the progress in the field of *NFV acceleration*, we present a new taxonomy of the state-of-the-art efforts according to various acceleration approaches. We discuss the surveyed works and identify the respective advantages and disadvantages in each category. We then discuss the products, solutions and projects emerged in industry. We also present a gap analysis to improve current solutions and highlight promising research trends that can be explored in the future.

CCS Concepts: • **Computer systems organization** → **Network Functions Virtualization**; *NFV Acceleration*; • **Networks** → Carrier Networks;

Additional Key Words and Phrases: Network Functions Virtualization, NFV acceleration, high performance

ACM Reference Format:
Xincai Fei, Fangming Liu (corresponding author), Qixia Zhang, Hai Jin, and Hongxin Hu. 2020. Paving the Way for NFV Acceleration: A Taxonomy, Survey and Future Directions. *ACM Comput. Surv.* 1, 1, Article 1 (January 2020), 40 pages. https://doi.org/10.1145/3397022

1 INTRODUCTION

Modern networks are becoming more sophisticated with the ubiquitous deployment of hardware-based network functions (NFs), also known as middleboxes [128]. These middleboxes play a key role in advanced traffic processing and provide a variety of network functionalities, from firewalls and intrusion detection systems for improving security to proxies and WAN optimizers for enhancing performance. Traditionally, middleboxes are usually tightly coupled with dedicated hardware. When launching a new service, more pieces of hardware are needed, which leads to a long service deployment cycle. With more capital investment and shorter hardware lifecycles, network operators face a severe situation of substantially lower profits, since launching new services is difficult and time-consuming.

Authors' addresses: Xincai Fei, god14fei@hust.edu.cn; Fangming Liu (corresponding author), fmliu@hust.edu.cn; Qixia Zhang, zhangqixia427@hust.edu.cn; Hai Jin, hjin@hust.edu.cn, National Engineering Research Center for Big Data Technology and System, Key Laboratory of Services Computing Technology and System, Ministry of Education, School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Rd, Wuhan, Hubei, 430074, China; Hongxin Hu, Clemson University, Clemson, SC 29634, Clemson, South Carolina, 29631, USA, hongxih@clemson.edu.

0360-0300/2020/1-ART1 $15.00
https://doi.org/10.1145/3397022

Network functions virtualization (NFV) is a novel paradigm for offering flexible network services, by leveraging virtualization technologies to decouple NFs from hardware middleboxes and make them execute in virtual machines (VMs), containers, or even natively. By this means, NFV abstracts general-purpose hardware (e.g., x86 servers) and brings considerable benefits: 1) reduced costs in purchasing regular commercial off-the-shelf (COTS) hardware instead of dedicated network equipment; 2) shortened time-to-market to deploy new network services and increased profits from these new services; 3) greater flexibility to scale up or down software-based NFs in an on-demand basis and migrate NFs to various locations whenever necessary; and 4) higher efficiencies in power, cooling and finding space for launching new network services. Based on a report by Global Market Insights company, the NFV market share will exceed 70 billion US dollars by 2024 [42].

Although NFV introduces substantial benefits, it is still challenging to put the deployment of software-based NFs into practice. One of the major obstacles is the difficulty in achieving high performance comparable to hardware-based middleboxes when running software NFs in a virtualized environment. For example, the processing delay of fully virtualized gateways results in a maximum of 132 μs, which is almost 8 times larger than the processing delay of decomposed gateways (i.e., 15 μs) [13]. In fact, it is vital for many NFV-based applications to achieve the desired performance. For instance, Voice over Internet Protocol services are particularly delay-sensitive and often require less than 150 milliseconds (ms) of end-to-end latency with low latency variance (i.e., jitter) [24]. Even a transient lack of connection can cause service disruption and thus violate the service level agreements (SLAs). For throughput-sensitive applications, Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) forwarding need to support 10 million of packets per second (Mpps) [1] for providing high performance and scalability. In addition, a high-performance IP security (IPsec) stack requires to sustain more than 190 gigabits per second (Gbps) of encrypted traffic on a single server [63]. Therefore, NFV solutions need to be carefully designed to satisfy the strict performance requirements of different applications.

As technology advances, modern commodity servers are generally equipped with multiple CPU cores, thus allowing workloads running on processors in parallel. Meanwhile, the fast development of high-speed network enables network interface cards (NICs) to evolve from 10 Gbps to 100 Gbps [72]. However, these advances do not guarantee high performance for NFV applications. The fundamental reason behind this is that the operating system's (OS) networking stack is not designed for fast middlebox processing but for general-purpose communications. Indeed, achieving high-performance packet processing from commodity servers is challenging and has been a hot research topic in both academia and industry (e.g., ETSI works [44–48]) for several years. In many cases, it is hardly possible to achieve the desired performance due to the following obstacles that are inherent in both software and hardware, which cumulatively slow down the packet processing in commodity servers and hinder high performance in NFV:

1) After the speed of NICs transiting from 10 Gbps to 100 Gbps, the processing time of each packet on the network links decreases from 67 nanoseconds (ns) to only 6.7 ns for the minimum 64B packet, in order to sustain line rate [90]. This imposes great pressure on the NFV system architectures, which will inevitably get worse in the case of a 100-gigabit interface [71].
2) General-purpose network stacks, such as the Linux network stack, lack optimized kernel-user space and fast input/output (I/O) data path for software packet processing. In fact, modern OSes can result in 10 to 20 times the speed slower than a 10-gigabit interface to process a packet for an application [20]. High cost induced from numerous user-kernel context switches and system call interruptions turn out to be the major performance bottlenecks for yielding high performance [74, 90].
3) Popular software switches, such as Open vSwitch [36, 112], which are used by (i) hypervisors, such as Kernel-based VM (KVM) [83], to interconnect VMs and/or (ii) bare metal deployments to interconnect containers (e.g., Docker), are incapable of providing desired performance. The reasons would be the inefficiencies of the software switch, the bottlenecks exist in virtualized network device drivers (e.g., virtio [123]), and inter-core communications for traffic steering.
4) Some existing NFV frameworks (see Section 4.4) do not fully exploit low-level hardware features, such as cache consistency in a 3-hierarchy style and memory concurrency, in order to improve

the cache hit and decrease the memory-access overhead. Besides, the capabilities of modern NICs are also not fully utilized, since they can not only do simple hashing of packets (e.g., Receive Side Scaling [27]) but also provide both packet classification and dispatching capabilities (e.g., Flow Director [58]). Fortunately, these problems are recently solved by Metron [72].

In the context of NFV, traffic usually needs to go through a sequence of NFs in a specific order, which is also known as service function chain (SFC), or service chaining [66]. Accordingly, it needs multiple VMs (or containers) to create a single network service, and those VMs could reside in the same or different servers. Traffic among VMs has to flow back and forth among different guest OSes within the same or different hosts (servers). The latter case brings a greater challenge for sustaining high performance, compared to service chains within a single server where packet processing incurs lower delay and consumes fewer CPU resources than going through the NICs [106].

To address the performance issues described above, a growing number of research efforts have been paid to improve NFV performance. Specifically, NFs are possible to achieve prominent performance improvement in an NFV platform by some forms of acceleration. Through a careful review of the progress in this field, we present a new taxonomy regarding all the surveyed works based on various acceleration approaches. We further discuss these works in detail and reveal their respective advantages and disadvantages in each category.

In this article, both acceleration of NFV and packet delivery for specific NFs (e.g., software switches and routers) are considered as the survey topic. Although Cerović et al. [20] have already written a related survey, many NFV-specific acceleration works are missing in their work since their major focus is on packet processing. In contrast, we focus on NFV acceleration and our work provides comprehensive reviews on the typical performance challenges in the context of NFV. We present a new taxonomy regarding a wide spectrum of NFV acceleration categories, while [20] just classifies the approaches into hardware- or software-based implementations and does not discuss some important aspects (e.g., NF offloading on smart NICs). We also discuss industrial products and solutions that are lacking in [20], which indicate the feasibility of NFV acceleration into practical applications. Other works just focus on only one aspect, such as I/O frameworks [38] or NFV datapath [86]. Besides, we present a gap analysis in order to improve the current solutions and introduce some promising research trends that can be explored in the future.

The roadmap of this article is organized as follows. In Section 2, we introduce the performance metrics of interest, take a close look into the typical performance bottlenecks in NFV and analyze the associated performance challenges. Next, we introduce some common frameworks, software switches/routers, and hardware accelerators used in the field of NFV acceleration in Section 3. In Section 4, we provide a taxonomy regarding the state-of-the-art approaches in this field. We survey these works in each category from Section 4.1 to Section 4.4, respectively. Then we briefly give a discussion and comparison of the surveyed works in Section 4.5. In Section 5, we discuss the emerging products, solutions and popular projects in industry. We present a gap analysis and highlight promising future trends in Section 6. Finally, Section 7 summarizes and concludes this article.

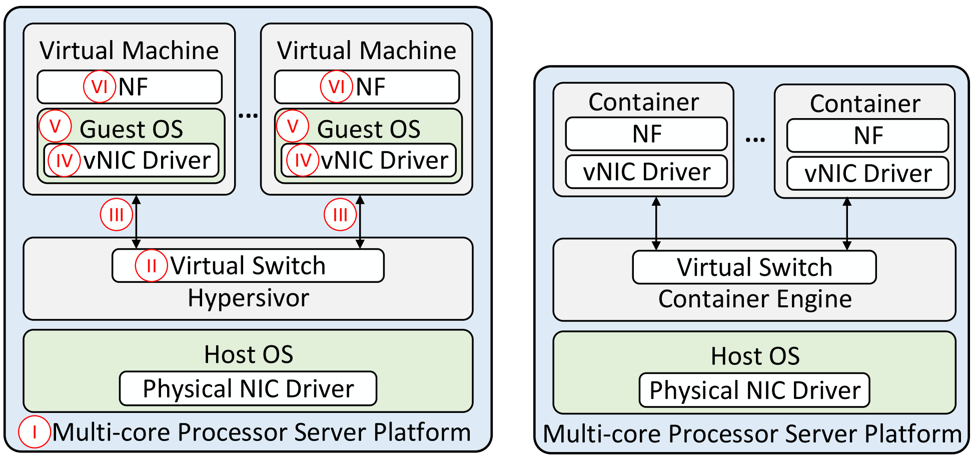
2 PERFORMANCE METRICS, BOTTLENECKS AND CHALLENGES IN NFV

In this section, we introduce the performance metrics of interest, and study and summarize the bottlenecks and challenges that affect the performance of NFV.

2.1 Performance Metrics of Interest

In an NFV platform, network performance is the analysis and review of collective statistics and metrics, in order to reflect the quality of network services. Broadly, there are multiple means to measure the performance in the context of NFV, as each service chain has different SLAs. Among the available options, the following performance metrics of interest are often considered as fundamental.

- **Raw interface throughput** describes the total amount of traffic that can be transferred over an I/O device such as a NIC over time, which is usually measured in megabits or gigabits per



(a) VM-based NFV Platform with a Hypervisor

(b) Container-based NFV Platform without Hypervisors

Fig. 1. Typical NFV performance bottlenecks.

second (i.e., Mbit/s or Gbit/s). This metric is used to weigh the processing speed of a physical network interface and is limited by the maximum interface bandwidth.

- **Packet throughput** describes how fast an I/O system can process packets, which is usually measured in millions of packets per second (i.e., Mpps). This metric is used to weigh the packet forwarding speed of a NIC and has much to do with the packet size. Generally, the smaller the packet size is, the greater the challenge is imposed on packet throughput.
- **End-to-end latency** is usually measured from a source that generates the traffic and to a sink that receives the traffic. The end-to-end latency can be further classified into four types of delays, i.e., processing delay to handle packets on a node, queuing delay to buffer packets before sending them out, transmission delay to push packets onto a link, and propagation delay to deliver packets through the link.

Even though both raw interface throughput and packet throughput can describe the throughput, the former reflects the data transmission speed through an interface, while the latter reflects how many packets of the same size can be transferred per second. Note that some surveyed works need to measure the throughput under different packet sizes, in this case, packet throughput is more intuitive for evaluation. Besides, other metrics such as packet loss rate and the latency variance, also known as jitter, can also be used to measure the network performance. Since most of the surveyed works evaluate their systems via throughput or latency, we mainly discuss the three listed metrics above in this paper.

2.2 Typical Performance Bottlenecks

NFV advocates to use software-based NFs implemented in commodity servers. Multiple NFs are often consolidated into a single server, where each NF runs on a VM basis in a virtualized environment. As such, a typical NFV platform is generally composed of some fundamental components such as host OS, virtual switches, virtual NICs and NFs. An NFV platform can run NFs in VMs or containers, or even run them as native processes. There are two typical NFV platforms (i.e., with/without hypervisors), both of which are plotted in Figure 1. Here we first focus on the performance bottlenecks in a VM-based NFV platform with a hypervisor, where the general location of each kind of performance bottleneck is shown in Figure 1(a). These bottlenecks can be categorized into six kinds, from the server hardware bottleneck to NF bottleneck. Note that some bottlenecks are not unique to virtualized environments, however, these bottlenecks would accumulate as NFV is adopted. We now give a closer examination on how these bottlenecks occur from a systematic view.

I. **Server Hardware Bottleneck.** As a whole, the server hardware characteristics greatly affect the performance of NFs running on it. Typical server parameters can be the number of cores, CPU frequency, cache size, memory size and latency, bandwidth of peripheral buses, NIC speed, instruction set, etc. An NFV platform usually requires to be equipped with powerful servers (e.g., Intel Xeon E5 v4 series processors with up to 22 cores, 44 threads, 55 MB LLC, 40 GbE NIC, and Intel Xeon E7 v4 series processors with up to 24 cores, 48 threads, 60 MB LLC, 40 GbE NIC) to eliminate server hardware bottleneck.

II. **Virtual Switch Bottleneck.** The virtual switch bottleneck is obvious since the virtual switch on the hypervisor can be viewed as an intermediate stop, through which all packets must go, no matter packets travel into and out of a host, or packets flow between VMs. The capability of the virtual switch plays a crucial role in a high-performance NFV platform.

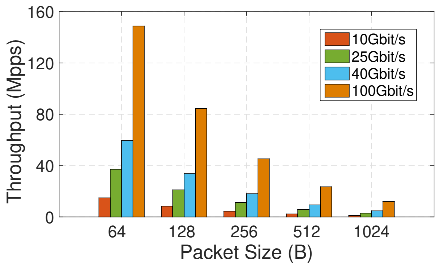
III. **Communication Bottleneck.** The communication bottleneck between the guest and host OS arises from double memory page translation and interrupt remapping, etc., between virtual hardware resources and physical hardware resources. The virtual CPUs in the guest OS run as QEMU [15] threads which are treated as normal processes in the host OS. These threads can be scheduled in any physical core in the host which increases cache misses and hampers the performance. This results in that some unnecessary OS features may also introduce high cost that limits the performance. For instance, the task scheduling mechanism of the OS is good for a fair CPU utilization among the processes; however, for NF processing that requires high performance, it fails and may lead to high context switches due to the fair CPU shares.

IV. **I/O Device Bottleneck.** The I/O device bottleneck comes with the virtual NIC drivers. It incurs overheads due to the high overhead of delivering packets between guest VMs and I/O device driver, since the driver cannot directly access the memory where a packet lies in the guest VM. Effective memory sharing mechanism should be provided by the driver for accessing the packet. For example, virtio [123] is a major virtualization standard for network device drivers and has been adopted by many NFV solutions; however, it is slow since the hypervisor has to emulate actual physical NICs, and this emulation is both complicated and inefficient.

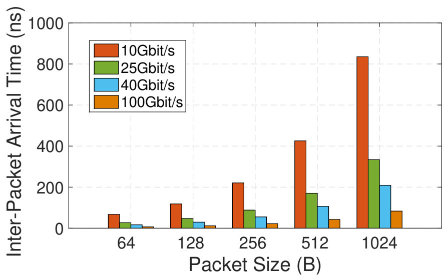
V. **Guest OS Bottleneck.** The guest OS bottleneck is similar to the bottleneck that exists in a non-virtualized environment. The packet processing in a traditional networking stack often incurs heavy kernel-user space context switches. Besides, the costly system call interruption for awakening the virtual CPU to handle the arrival packets further penalizes the performance.

VI. **NF Bottleneck.** The NF bottleneck occurs due to inefficient software design. NF design concerns whether all capabilities offered by the server hardware can be leveraged. Software features such as pipelining, multi-threading and deadlock avoidance should be carefully considered in the design of a new NF. With regard to exploiting the hardware architecture, deterministic allocation of CPUs can improve the performance of some NFs, since threads in CPUs may communicate with each other and benefit from sharing the same cache (improved cache hit ratio) when they need to exchange information. Furthermore, NFs can get faster access to the memory that is close to the CPU in which the thread is running than the memory connected to remote CPUs. For those NFs that need high throughput performance and frequent memory access, these software features need to be carefully integrated into the NF design.

As compared with the heavy-weight VM-based NFV platforms, container-based NFV platforms (e.g., Docker) can implement NFs as standard user-space processes inside lightweight container instances, which can be quickly started while incurring little overhead. In this way, more NFs or application instances can run in containers with fewer resource consumption, lower system overhead and higher performance as compared with VMs. However, container-based NFV platforms also have some performance bottlenecks, such as the server hardware bottleneck, the communication bottleneck, the NF bottleneck, etc. As plotted in Figure 1(b), since there is no hypervisor in container-based NFV platforms, all the container instances share the same host OS along with its hardware resources. Thus, the performance isolation in containers is not as strict as hypervisor-based platforms with fully virtualized hardware resources. In other words, containers are more fragile when facing Kernel attacks. For instance, an attacker can use some non-namespace resources in Docker containers to access the host OS. In this case, the container under attack may occupy



(a) Throughput.



(b) Inter-packet Arrival Time.

Fig. 2. The packet throughput and inter-packet arrival time under different packet sizes and interface bandwidths.

the entire host's resources, which will severely affect the operation of other containers and their running NFs. Performance isolation in container-based environments can also be broken by the large computation overhead related to the network stack [75]. As a matter of fact, containers with high traffic rates can cause the performance of co-located computing-driven containers to suffer an almost 6x slowdown. Besides, how to split microservices in container-based NFV platforms and how to efficiently communicate with other microservices will both affect the performance.

Due to numerous layers of bottlenecks in both software and hardware accumulated to the NFV platform, current NF deployments are just in small test scale, while large-scale NF deployments are severely limited. Though many high-performance software frameworks have been employed to address some performance issues via bypassing networking stacks, the accelerated guests only guarantee fast packet processing for specific NFs and the accelerated virtual switch still requires to copy packets between the hypervisor and the guest's memory in VM-based NFV platforms. Recent research finds that the PCI Express (PCIe), which is the interconnection between the host and device drivers, can also become the performance bottleneck even in combination with optimized I/O frameworks, since it has side effects on both throughput and latency of network applications [102]. For a high-performance service chain, there still exist a number of hindrances in the NFV platform, especially enabling high-speed communication between two guests or between host and guest. When a service chain is deployed in two or more NFV platforms, achieving this target will become more challenging.

2.3 Performance Challenges

For an NFV system that is designed for carrier networks, NF deployments often need to provide throughput with the order of tens of Gbps and latency with the order of tens of µs [108], in order to meet the strict performance requirements. With the development of NICs, there is a strong demand for higher packet processing speed. This, however, imposes great challenges on the platform to reach the theoretical upper bound of forwarding packets. To demonstrate this truth, Figure 2 shows the packet forwarding rates (throughput) and inter-packet arrival time under different packet sizes and interface bandwidths. As plotted in the figure, while the packet forwarding rate increases from 14.88 Mpps to 59.52 Mpps, the inter-packet arrival time drops from 67.2 ns to 16.8 ns when the interface bandwidth increases from 10 Gbps to 40 Gbps for 64B packet. Katsikas et al. [71] even declare that the per-packet time to process 64B packets at 40 Gbps is only 12.8 ns. To sustain the maximum packet forwarding rate, shorter inter-packet arrival time means fewer cycles that a CPU is allowed to process a packet.

Modern multi-core processors are common to adopt a multi-level cache design. Specifically, each CPU core has its dedicated L1 cache and L2 cache, but the last level cache (LLC) is shared among the cores of the same CPU socket. This multi-level cache architecture can largely reduce the CPU cycles for memory access. Note that both LLC and dynamic random access memory (DRAM) have local access and remote access. Beside, non-uniform memory access (NUMA) architecture is also common in modern multi-core systems to separate memory for different processors, where the

Table 1. Latency of memory access on Intel Xeon 5500 processor [87].

Data Source	**Latency**
L1 cache hit	4 cycles
L2 cache hit	12 cycles
LLC hit, unshared line	26-31 cycles
LLC hit, shared line in another core	65 cycles
LLC hit, modified line in another core	75 cycles
Remote LLC	100-300 cycles
Local DRAM	60 ns
Remote DRAM	100 ns

memory access time is proportional to the distance between the memory and the processor. In a NUMA-based system, accessing a processor's local memory is much faster than accessing remote memory (i.e., memory shared among processors or memory local to another processor).

Table 1 summarizes the latency of memory access with different data sources on a NUMA-based architecture with two Intel Xeon 5500 processors (each with 256 KB L2 cache and 8 MB shared LLC among 4 cores), which are based on the same microarchitecture as Intel Core i7 processor [87]. This processor architecture also brings a huge difference in processing packets, especially for packets with small sizes. For instance, the inter-arrival time of 64B packets is 12.8 ns at 40 Gbps, which allows a CPU to process the packets in 33 cycles with the main frequency of CPU at 2.6 GHz. Considering that both receiving or sending a packet and table lookup need memory access, it is still hard to sustain maximum packet forwarding rate within 33 cycles, only if LLC can hit in every memory access with a maximum latency of 31 cycles. If LLC misses, packets via remote DRAM require up to 100 ns. Even though superior processors like Intel Xeon E5 v4 processors allow direct data transfer to LLC with optimized LLC-to-core communications and Intel Xeon E7 v4 processors even increase the LLC by 60 MB (up to 2.5 MB per core), it is still challenging to reach theoretical packet forwarding rate in case that LLC misses.

In the context of NFV, multiple NFs are often running in a single server. However, in fact, few state-of-the-art packet processing schemes can guarantee that packets are delivered to the right core straight from the NIC. They either suffer from inefficient inter-core packet transfers among NFs or unbalanced workload among CPU cores [72]. One existing work, Metron [72] can process packets potentially at L1 cache speeds. In particular, Metron offloads part of the packet processing logic to the network to eliminate inter-core transfer and handles the remaining packets with tag-based hardware dispatching so as to achieve a potential L1 cache speed for processing them. Assisted with commodity hardware, Metron can process deep packet inspection at the speed of 40 Gbps and realize stateful NFs at the speed of a 100 GbE network card on a single server. However, it inherits the limitations of SNF [73], which makes it only applicable to fairly limited service chains. In short, considering that only a few tens of CPU cycles are available for each packet at 100 Gbps, it is a huge challenge for achieving either high throughput or low latency of NF chains.

3 COMMON SOFTWARE FRAMEWORKS, SWITCHES/ROUTERS AND HARDWARE FOR NFV ACCELERATION

Before investigating the state-of-the-art approaches in the field of NFV acceleration, we first give a general understanding of some necessary prerequisites. In this section, we briefly introduce the software frameworks, software switches/routers and popular hardware that are commonly associated with NFV acceleration.

3.1 Software I/O frameworks and Switches/Routers

High-performance software frameworks have been developed to provide high packet I/O speed since several years ago. Until now, the most popular examples are Intel Data Plane Development Kit (DPDK) [29], netmap [118], PF_RING [103] and PF_RING DNA (Direct NIC Access) [119], Snabb [107], and OpenOnload [129]. These frameworks achieve performance improvements either via bypassing the OS kernel and running applications in user space or sharing memory buffers that can be accessed by both kernel and applications [9, 116]. There are some software switches that run on top of a network I/O framework, such as VALE switch [120] and OvS-DPDK [41]. We also discuss some popular software switches (or routers) such as Open vSwitch (OvS) [36, 112], Berkeley Extensible Software Switch (BESS) [50] (a.k.a., SoftNIC [51]), and Click Modular Router [78]. Table 2 summarizes the general information of these frameworks and switches/routers.

DPDK. DPDK consists of a collection of data plane libraries and NIC drivers for high-speed packet processing. By calling DPDK libraries and application program interfaces (APIs) in the user space, applications continuously poll instead of handling interrupts for each arrived packet. In this way, the overheads inherent in traditional networking stack can be eliminated. DPDK also implements a set of optimizations including lock-free rings, thread affinity, NUMA awareness, and huge page setting. In particular, the lock-free rings in DPDK are based on the Linux kernel lock-free ring buffers to reduce locking overhead, which supports both one-to-one and many-to-many producer/consumer models. Thread affinity means that DPDK binds threads to logical cores to reduce context switches across cores and increase CPU cache hit rate. DPDK also implements per-core memory in NUMA to deal with cache consistency and uses 2 MB and 1 GB huge pages to reduce the case that translation lookaside buffer (TLB) misses. DPDK maintains hardware independence since it provides a programming framework for any processors. DPDK can boost the packet processing performance by 10 times, reaching more than 80 Mpps throughput by using a single processor [57]. However, the CPU overhead is not reduced since DPDK Poll Mode Drivers (PMDs) run in busy loops by constantly scanning either physical NIC ports on the host or vNIC ports in guests for the arrival of packets.

netmap. Netmap is a high-performance framework for packet I/O. It reduces or eliminates substantial overheads such as per-packet memory allocations, system calls and packet copying by introducing three optimization techniques, i.e., buffer preallocation, use of large batches and shared buffers. All of these are for fast transmission of packets through the NIC to applications in the user space. Note that netmap has two modes to allow users to decide whether packets should go through the host stack or not [20], while the netmap mode allows packets to be directly delivered to an application through the netmap API. The implementation of netmap brings minor modifications and does not depend on special hardware. According to [118], netmap can easily reach line rate on a 10G NIC with 64B frame size.

P4. P4 [16] is a high-level programming language used for protocol-independent packet processors, which can work as a general, flexible interface between the switches and the controller for matching header fields and parsing packets. A simple API is provided by P4 for connecting the actual hardware implementation of the switch. In short, P4 can achieve three goals: protocol independence (i.e., switches should be able to handle packets of different formats), reconfigurability (i.e., the controller can redefine how to parse and process packets) and target independence (i.e., programmers can describe the functions to process packets without knowing the details of hardware implementation). As compared with commonly-used programming languages (e.g., C or Python), P4 is a domain-specific language, which provides high-level abstraction for network programming and optimizes network data forwarding through a set of carefully-designed constructs.

PF_RING (DNA). PF_RING is an efficient packet capture technology based on the Linux kernel [5]. The core solution of PF_RING is to decrease the copy time during the transmission processes of packets. To this end, PF_RING polls packets from the NIC to ring buffers, and then the user-space applications read packets directly from the rings, by means of utilizing Linux New API (NAPI). However, this mechanism has to experience two polling periods for both NAPI and the application, and consumes extra CPU cycles. To reduce CPU consumption for polling packets, PF_RING DNA copies packets from the NIC to ring buffers by NIC NPU instead of NAPI. It has better performance but the disadvantage is that only one application has access to the ring at a time and applications need to exchange for each other to distribute incoming packets.

Snabb. Snabb is a fast packet processing toolkit written in the scripting language Lua. It allows developers to add functionality to the Snabb stack by loading modules into the Snabb engine, which makes Snabb extensible. These modules can be Lua or other native code objectives, which are

Table 2. Common software I/O frameworks and switches/routers.

Framework	Begin time	Organization	Description
DPDK	2012	DPDK	A set of data plane libraries and drivers
netmap	2012	Universita di Pisa	A high-performance network I/O framework
P4	2014	P4 Language Consortium	A language for programming protocol-independent packet processors
PF_RING (DNA)	1998	Ntop Team	A new type of network socket
Snabb	2012	Snabb Lab	An extensible networking toolkit
OpenOnload	2008	Solarflare	A high-performance network stack
Switches/Routers	**Begin time**	**Organization**	**Description**
VALE switch	2012	Universita di Pisa	A software Virtual Local Ethernet
OvS-DPDK	2014	Linux Foundation	DPDK accelerated OvS
OvS	2012	Linux Foundation	An open-source virtual switch
BESS	2015	Berkeley	A modular and software switch
Click	2000	MIT	A modular and software router

named as libraries and *Apps*, respectively. A specific application is modeled as an App network, and the Apps can be very simple or complex operations in NFs. As a result, developers can build customized software packet processing network by resembling reusable Apps. Snabb supports NFV as it allows fast data exchange between VMs and provides user-space drivers similar to DPDK.

OpenOnload. OpenOnload is a high-performance framework that not only significantly reduces latency and CPU utilization, but also increases packet rate and bandwidth [129]. It runs on standard Linux and requires no modifications to applications. Its high performance partly stems from detouring the OS stack so that the packet processing can be fully performed at user space. Note that OpenOnload is a hybrid stack, which allows applications to flexibly run either at user space or kernel space for any traffic to choose, whichever is appropriate. Applications using OpenOnload typically see a 4-time improvement in message rates.

The following is an introduction to the software switches or routers in the literature.

VALE switch. VALE is a software Virtual Local Ethernet. The ports on VALE can communicate using the netmap API. It is designed to be used to interconnect multiple VMs, working as a learning bridge in the host. It can achieve very high-speed communication between different types of VMs (up to 20 Mpps with short frame). However, as VALE is developed as a small extension of the netmap module, both QEMU and KVM hypervisors have to be modified for accommodating the new network backend.

OvS-DPDK. OvS-DPDK is a form of accelerated OvS with DPDK datapath support, by taking advantage of the DPDK libraries to bypass the host kernel. It appears like creating a user-space vSwitch on the host by using DPDK PMD driver. As a result, OvS-DPDK can achieve up to 10 times performance improvement than native OvS.

OvS. OvS is an open source software switch and usually served as a virtual switch in a virtualized environment residing on the hypervisor. OvS nearly supports all L2-L3 network switches such as NetFlow and sFlow. The original intention of OvS is not for speeding up packet processing but for easing the management and configuration of the virtual networks. While OvS uses Linux kernel networking to switch packets between VMs, the performance of native OvS is often unsatisfactory (the performance can be as much as 50 times worse than that of bare metal environment).

BESS. BESS is a modular framework for software switches to overcome the severe contradiction between limited NIC abilities and dynamic demands from the users. It allows developers to flexibly add NIC features in software while incurring negligible performance overhead. Borrowing the basic concept from Click, BESS makes it possible for a user to customize the packet processing by using a number of modules. When connecting two modules, a user can create a traffic class. For delivering packets between two modules, the user then assigns a dedicated core where the traffic class runs as a unique thread. BESS can achieve multiple-10G performance even on a single core.

Click. Click is a very popular software router that can be configured in a flexible and modular way. A click router consists of multiple elements, each of which processes packets to implement a single router function [140]. It is very easy to extend Click due to its modularized design. Except for routing, Click can also be used for rapid prototyping and new protocol development. Great efforts have been paid by researchers to improve the performance of Click, such as RouteBricks [28], DoubleClick [76] and FastClick [10].

Obviously, each software framework has its advantages and disadvantages. For instance, polling-based approaches such as DPDK obtain desired performance at the cost of high CPU resource consumption, while interrupt-driven frameworks such as netmap need large batch sizes, which can also harm performance like latency [153]. Gallenmüller et al. [38] present a comprehensive survey of the state-of-the-art high-performance packet I/O frameworks. They make a comparison among DPDK, netmap and PF_RING in representative packet forwarding scenarios. Hence, they reveal the impacts of caching and establish a model to evaluate the performance of the three frameworks. Besides, although there may exist other similar frameworks, we focus on the frameworks above since we regard them as the most relevant ones.

3.2 Popular Hardware

To achieve high performance, another popular method resorts to employing popular hardware such as network processor units (NPUs), graphics processing units (GPUs), field programmable gate arrays (FPGAs) and smart NICs (sNICs) for NFV acceleration. A common practice is to offload NF processing from the host to hardware, then NFs can perform much faster than when they are executed on a general-purpose CPU. In contrast to special hardware such as application specific integrated circuits (ASICs), these pieces of hardware have many advantages. Table 3 presents a brief introduction to the hardware that can be employed for NFV acceleration.

NPU. NPU is a programmable and general-purpose processor specially designed for packet processing. It adopts multi-core architecture with parallel processing capability, high-speed interface and I/O capacity, which is commonly used in the communication field to perform packet processing, protocol analysis, route lookup, voice/data convergence, etc. The execution logic inside an NPU depends on the software loaded during the runtime, which is developed using a dedicated instruction set, i.e., microcode. NPUs are used in the manufacture of a variety of network devices such as software routers, switches, firewalls and session border controllers (SBCs).

GPU. The original role of GPU is for graphics rendering. Now GPU has been widely used to supplement a CPU and accelerate many compute- and memory-intensive applications due to its extreme thread-level parallelism, which makes it very suitable for packet processing. GPU can be used for various NFs such as software routers [52], IPsec gateways and network intrusion detection systems (NIDSes) [43].

FPGA. FPGA contains an array of programmable logic blocks using a large number of gates. These logic blocks can be configured to perform complex functions. Besides, FPGA also includes memory elements to store states and data, and has a large number of parallel cores within an FPGA chip. However, low-level hardware description languages (HDLs) like Verilog and VHDL, have to be used to program FPGA [88], which make the programming efforts to be huge. On this account, high-level synthesis (HLS) tools [20] that allow developers to program FPGA with high-level programming languages [88] are adopted in many FPGA-based solutions.

sNIC. sNIC is an enhanced NIC which often has its own multi-core processors and an internal switch compared with traditional NICs. A standard OS can run within the sNIC, which makes it possible to offload almost any packet processing functions from CPU, such as security-related applications [98, 124]. Note that some sNICs can be manufactured using other hardware, such as FPGAs [100, 134].

Similarly, different hardware also has its strengths and weaknesses. Though NPU has advantages such as high performance and programmability, its applications are still limited due to the high cost and domain-specific restriction (generally applied to dedicated communication devices). As different

Table 3. Common popular hardware for NFV acceleration.

Hardware	Feature	Use in NFV
NPU	Programmability	Optimized for packet processing in network devices
GPU	Massive parallelism	Improve the throughput of compute- and memory-intensive applications
FPGA	Reconfiguration	Accelerate complex packet processing with high performance and flexibility
sNIC	Flexibility and programmability	Offload almost any NFs from CPU for processing

vendors have different microcode standards for NPUs, developers usually get stuck in building an ecosystem. Though the microcode of some NPUs has supported to be compiled and generated using high-level languages like C, its conversion efficiency is poor. Based on [69], the advantages of GPU for packet processing are vectorization and memory latency hiding when compared to CPU. However, the communication between CPU and GPU introduces latency. The processing of large batches of packets on GPU further increases latency. Moreover, GPUs will no longer have the advantage of a big part of memory bandwidth compared to CPUs under random accesses, as the memory subsystem in GPUs is optimized for adjacent access. Compared with NPU, FPGA is more versatile due to its reconfiguration feature. Furthermore, FPGA is more power-efficient than GPU. In contrast to CPUs or GPUs, the major advantage of FPGA is that it can provide ultra-low latency through the PCIe channel between CPU and FPGA, which is just in the order of nanoseconds. However, the programming complexity of FPGA is non-trivial and hinders the development of this technology for years [8]. For sNICs, they typically have limited total compute and memory capabilities compared to CPUs.

Although there may exist other hardware for NFV acceleration, we focus on the listed hardware above since they appear in the surveyed works most frequently.

Besides, a standard specification called sing-root I/O virtualization (SR-IOV) [82] is commonly used to improve the throughput of NFs. It defines a mechanism to create shared network devices called virtual functions (VFs). This sharing makes a sing I/O device appear to be multiple separate physical devices. By providing independent memory space, interrupts and DMA streams, SR-IOV bypasses the host kernel in data transfer. However, SR-IOV is hardware-dependent since it needs the support of NICs. It also has theoretical limits as a physical device at most supports 256 VFs.

4 TAXONOMY AND SURVEY OF ACCELERATION APPROACHES

After we carried out a comprehensive investigation of the research efforts in the field of NFV acceleration, we propose a taxonomy for these works according to various acceleration approaches. As illustrated in Figure 3, we classify computation and communication acceleration as two main categories of NFV acceleration, since NFs are usually identified to be compute-intensive (e.g., IPsec gateway) or network-intensive (e.g., network address translation). Many solutions are proposed to accelerate these NFs with strict performance requirements. Recently, some works [132, 156] in the literature have studied how to improve NFV performance through network function parallelism, which are different from previous acceleration approaches. Hence we add this acceleration technology as an independent category in order to draw more attention. Besides, packet steering happens repeatedly in every step of a service chain to handle a packet and can cause severe inter-core transfers without proper acceleration. Therefore, we also add the packet steering acceleration as a separate category, which is very crucial for improving service chain performance in NFV.

For some categories, we classify them into more subcategories based on the specific means they employ. Specifically, the taxonomy contains the following aspects of NFV acceleration:

- **Computation Acceleration**: no matter which means to adopt, either with hardware offloading, software tuning or modularized reuse in NFs, the essence of the approaches is to reduce CPU resource consumption. According to this purpose, we classify those approaches into this category. (See Section 4.1)

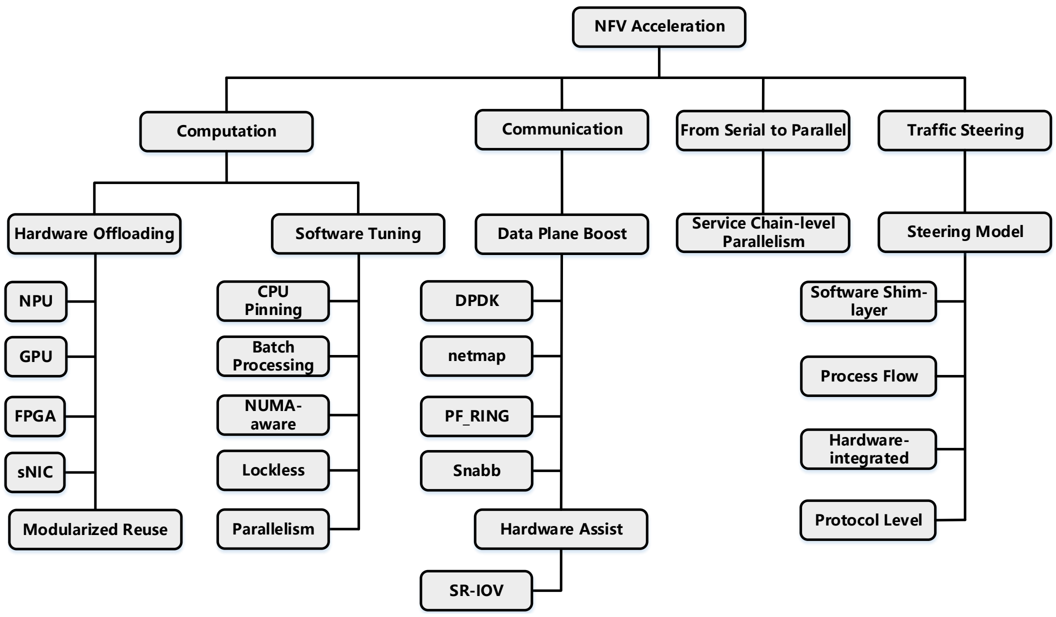


Fig. 3. Taxonomy of the surveyed NFV acceleration approaches.

- **Communication Acceleration**: in this category, the approaches generally exploit high performance I/O frameworks and I/O passthrough technology such as SR-IOV. The primary purpose is to speed up packet processing and improve throughput. (See Section 4.2)
- **From Serial to Parallel Acceleration**: we treat this category as NFV-specific, since it is optimized for decreasing latency by running some NFs of a service chain in parallel without violating NF dependencies. (See Section 4.3)
- **Traffic Steering Acceleration**: the purpose of this category is to decrease inter-core communications or deal with more complex situations in traffic steering, by utilizing efficient steering models. Considering that many NFs are stateful, this form of acceleration is vital to achieving high-performance service chains. (See Section 4.4)

Note that the classified characteristics of NFV acceleration approaches might be correlative, particularly the subcategories between computation and communication acceleration, and between communication and traffic steering acceleration. For instance, the benefits of software tuning can be multifold. On one hand, tuning techniques such as data prefetching and batch processing of packets ensure rapid NF computation in processors. On the other hand, techniques such as zero-copy packet delivery using shared memory enable fast communication between NFs. The branches of a certain subcategory can also be correlative, while we just highlight the dominant features. For example, some sNICs are manufactured using FPGAs [100, 134]. Besides, other techniques such as keeping the cache line aligned to avoid packet transfer among multiple cores, conform to the idea of traffic steering acceleration. In this taxonomy, we just classify software tuning in computation acceleration as a representative. Furthermore, our taxonomy also considers different levels of NFV acceleration, from an NF block [17] abstracted from a monolithic NF to a service chain that connects multiple NFs. Other studies on NFV acceleration [18, 21, 86] provide a rather coarse-grained taxonomy or focus on only one aspect. Specifically, Bronstein et al. [18] focus on hardware acceleration and advocate solving the compatibility problem by providing an abstraction to hide the complexity of the implementation. Chatras et al. [21] target on NF's portability across different platforms for delivering high performance, and classify acceleration solutions into software and hardware acceleration. Lettieri et al. [86] just focus on the NFV data-path. By contrast, our work is more comprehensive and covers the contents of the three studies above.

Besides, energy consumption should not be sacrificed in accelerating NFV. We would not discuss it further with few related works. Data traffic demands (e.g., synthesized or adversarial workloads) are of great importance in benchmarking NF performance, see Section 6.2 for details.

In the subsequent sections (from Section 4.1 to Section 4.4), we introduce and discuss the surveyed works according to this taxonomy. In Section 4.5, we briefly summarize and compare these works.

4.1 Computation Acceleration

Computation acceleration is often used to enable high performance for compute-intensive NFs, such as encryption and transcoding, which require significant CPU resources and involve complex CPU calculation. There are three types of methods in computation acceleration literature, namely, hardware offloading, software tuning and modularized reuse.

4.1.1 Hardware Offloading. The most common approach in computation acceleration is to utilize specialized hardware, which is referred to as look-aside accelerator in [44]. Moving NFs to accelerator hardware significantly saves CPU resources, which could be spent in other revenue-generated applications. Hardware offloading solutions have long been a research topic in developing high-performance applications. In the following, we review these works according to the hardware they employ.

NPU-based solutions. NPUs have been used for improving the performance of some specific NFs such as software routers and switches. In this respect, NP-Click [126] is proposed to close the gap between the ease of implementation and efficient implementation. NP-Click is based on Click and provides a programming model for the Intel IXP1200 network processor. The evaluation results on two applications (IPv4 packet forwarding and DiffServ interior node) demonstrate that NP-Click achieves very good performance. Ferkouss et al. [35] implement a high-performance OpenFlow [95] switch on EZchip NP4 network processor. As OpenFlow is rule-based, the authors leverage the flexibility of the switch by updating the flow entries on the fly. Meanwhile, using recursive flow classification (RFC) helps improve the parallelism of the classification process and better exploit the hardware resources. POF [130] builds a hardware-based prototype with the help of NPU. The goal of POF is to eliminate the dependency from forwarding devices (such as switches) without configuring specific protocols and promote the data plane. POF defines a generic flow instruction set to achieve this goal. By taking advantage of the hardware packet parsing modules in the NPU, the authors believe a POF optimized silicon chip can help improve the performance, thus achieving lower network cost and creating new value by enabling flexible network services.

GPU-based solutions. While GPU has higher computation capability and more sufficient programmability than NPU, PacketShader [52] takes the first step to exploit the power of GPUs in packet processing for developing a software router framework. By processing packets in batch, offloading specific packet processing operations to GPUs and scaling them in parallel, PacketShader achieves remarkable higher throughput compared to the CPU-only implementation, while the flexibility and programmability of a software router are still reserved. NBA [77] is a software-based packet processing framework that enables GPU but hides low-level details. It not only maximally exploits the hardware advances (such as batch processing, NUMA awareness, etc.), but also provides an abstraction layer for GPU offloading and achieves adaptive load balancing between CPU and GPU in various workload conditions. As such, the sample NFs of NBA reach up to 80 Gbps for both IPv4 and IPv6 routers, and to above 30 Gbps for IPsec and IDS. Snap [133] is a Click-based packet processing framework that surpasses previous software routers by exploiting the parallelism provided with GPUs. To make GPU elements easily integrated into a processing pipeline, Snap adds a collection of high-performance extensions to the native Click. To achieve this, the authors introduce a set of techniques, including packet slicing to reduce the consumption of bus bandwidth, and predicted execution to deal with the graph composed of Click elements and schedule packets in a pipeline that can be completed in order. They also accelerate common NFs with a collection of GPU-based elements. The evaluation results show that Snap can forward 44 Mpps with a packet size of 128 bytes, even when using elements to implement NFs such as an IP router, a packet classifier and a string matcher with a full payload. This outperforms the baseline CPU-based Click by nearly four times. DoubleClick [76] is another Click-based solution that focuses on boosting the performance of Click router by employing I/O and batch processing, including a variety of optimizations for current server architectures. For IPv4 packets with size of 64 bytes, the performance can be improved by nearly 10 times through exploiting these techniques. DoubleClick falls into this category since it adopts PacketShader's packet I/O engine for I/O batching.

To reduce the programming efforts for developing high-performance GPU-based network applications, Vasiliadis et al. [136] present a network traffic processing framework called GASPP, which is elaborated for modern GPUs and well-integrated into a GPU-based prototype. GASPP allows developers to build complex GPU-based NFs in a flexible and efficient manner through hiding complex network processing issues, while providing an interface that shows only the data to NFs. The authors also propose a novel packet scheduling mechanism to avoid redundant data transfer and fully use the GPU and the PCIe bus, for efficient memory sharing between network interfaces and GPU. Other optimization techniques such as zero-copy and batch packet processing are applied to the framework design. Based on the evaluation results, GASPP can obtain multi-gigabit throughput even for some compute-intensive and complex network operations that commonly exist in many NFs. In the case of NF consolidation, GASPP obtains as high as 16.2 times performance improvement compared with single-GPU implementations. There are also many works that focus on specific NFs with GPU acceleration, including Kargus [59], Gnort [135] and MIDeA [137] for intrusion detection, and SSLShader [61] for cryptography.

Besides, GPUNFV [146] also exploits the advantage of GPU that delegates packet processing to GPU threads in great parallel. Different from previous approaches, the authors advocate deploying an entire service chain on GPU, which not only potentially reduces the number of needed GPUs, but also decreases data switching between CPU and GPU. What makes it unique is that it efficiently supports both stateful and stateless NFs, by processing the packets of the same flow on the same GPU threads. The evaluation shows that GPUNFV increases the throughput by more than 2 times, compared to packet processing only with CPU. Besides, the waiting time of CPU is no longer required and GPUNFV can quickly get the optimal batching size of packets with the dynamic sizing.

However, the capacity strength of GPU compared to CPU in speeding up NFs has been questioned in an existing work [69]. They claim that the performance improvement of GPU comes from less memory access latency rather than its fast parallel computing, and applying optimization techniques such as group prefetching and software pipelining to GPU code usually results in more efficient resource utilization than native GPU acceleration. Aiming at this doubt, APUNet [43] proves the efficacy of GPU in accelerating NFs and discovers three facts. First, GPU's computational power makes a critical difference in accelerating many compute-intensive NFs. Second, GPU's acceleration power without data transfer outperforms the optimized CPU code. Third, the capability of PCIe communication hinders the offloading of GPU workload, rather than the capacity of GPU. By adopting integrated GPU in accelerated processing units (APUs) and addressing a few practical challenges, APUNet extracts more computational power from integrated GPU with the help of extensive zero-copy models and massive parallelism. It also achieves low-latency data transfer between CPU and GPU. As a result, APUNet improves the performance of many memory- or compute-intensive NFs such as IPsec gateways, SSL proxy and NIDS by 2.2 times to 4 times. However, the performance improvement is not obvious for simple NFs like IPv4 forwarding.

Although GPUs have demonstrated their power in accelerating NFs, Zhang et al. [149] claim that GPUs still have not been effectively integrated into NFV systems yet. They find GPUs are not fully utilized, data isolation cannot be ensured and great development efforts are required in existing GPU-accelerated NFV systems. Thus, they propose G-NET [149], an NFV system that supports several features such as spatial GPU sharing, GPU scheduling and data isolating in GPU. Specifically, by generalizing the CPU-GPU pipeline, data transfer and multi-threading, G-NET provides an abstraction for developing NFs. Therefore, software developers simply need to implement a few specific-purpose NFs, which can largely reduce the development and implementation costs. Through evaluation with various workloads, G-NET is proved to reduce the latency of an SFC by 44.3% and increase the throughput by 70.8%, as compared with the temporal GPU sharing virtualization method.

FPGA-based solutions. Since FPGA is more versatile than NPU and more power-efficient than GPU, ClickNP [88] chooses to accelerate NFs using FPGAs that are considered to be inexpensive. Generally, it is rather hard to tune the logic and debug the code of FPGA since it is usually programmed with hardware description languages. ClickNP overcomes the programming challenges by implementing it in a modularized fashion, with the help of commercial high-level synthesis tools. By exploiting the massive parallelisms in FPGA with a set of optimization techniques, ClickNP obtains high throughput up to 200 Mpps and ultra-low latency for various types of software NFs. Compared with existing GPU-accelerated and pure CPU-based counterparts, ClickNP not only improves the throughput but also reduces the latency. To further implement rich as well as high-performance NFs besides packet processing functionality, Emu [131] develops a new standard library for NFs on FPGAs, while providing an execution environment that is easy to program and debug. Using Emu, developers can rapidly implement and deploy various NFs by programming FPGAs with a high-level language, while supporting fine-grained debugging and profiling capabilities to monitor the states of NFs at runtime. Compared to Linux native counterparts and native Verilog implementations, the performance of Emu significantly outperforms the former in both latency and throughput, while incurring a negligible overhead compared to the latter.

Considering that FPGA is expensive (compared to general-purpose CPU) and the limited number of building blocks on an FPGA board, it is both resource-demanding and cost-inefficient if the entire NFs are implemented on FPGA. Moreover, employing FPGA to accelerate the whole service chain is inflexible, due to the fact that FPGA has to be reprogrammed when introducing new NFs and the reprogramming time can be hours for tuning the code. To overcome the problems above, Li et al. [91] present a novel framework with CPU-FPGA co-design called dynamic hardware library (DHL), while providing both high performance and flexibility for NFV systems. To achieve this, DHL abstracts accelerator modules in FPGA as a hardware function library and provides a set of programming APIs linked to these accelerator modules. In this way, DHL decouples the hardware development of FPGA accelerator modules and software development of NFs, which enables both hardware developers and software developers to intently apply their expertise. To ensure fast communication between FPGA and CPU while keeping very low latency for NFs, DHL also designs an efficient DMA engine by employing a series of optimizations to improve the efficiency of data transfer. The evaluation results show that DHL can reach high throughput by up to 40 Gbps that is similar with FPGA-only solutions, and only incur latency less than 10 μs, while enjoying the flexibility that is lacking in FPGA-only solutions.

sNIC-based solutions. To offload NFs from host CPUs to sNICs, UNO [84] proposes a generalized NF offloading architecture that is managed by an SDN (software-defined networking) controller, for achieving the maximum resource utilization of both sNIC and host. It is non-trivial to achieve this due to three challenges. First, the sNIC is incapable of hosting all NFs and network switching in it, due to its limited total compute and memory capacities. Second, the CPU-sNIC co-design framework imposes extra bandwidth consumption on account of frequent data transfer between the host architecture and the sNIC. Third, the fact that packets traveling through multiple NFs between the host and the sNIC put great pressure on the controller. To address these challenges, UNO introduces an NF agent which makes the SDN controller unaware of the existence of the sNIC. To intelligently balance the load between the host and sNIC, UNO leverages a mathematical method to decide where an NF should be placed (at the host or on the sNIC). For ease of management, UNO translates rules from the SDN controller into rules that are used for NF traversal between the host and sNIC with OneSwitch of UNO. Experiment results demonstrate that UNO can use less than 8 CPU cores, while reducing the power consumption by 2 times and the management overhead of the SDN controller by more than 50%. Besides, Floem [113], a programming system, also targets at offloading computation to sNIC, while providing programming abstractions to overcome many common problems for developers, such as expression of communication strategies, data consistency and reuse of the code. Floem can accelerate many datacenter applications with sNIC offloading, including key-value store and data analytics, and improve the performance of common NF tasks such as encryption and flow classification.

When NFs on sNICs are overloaded, UNO adopts a naive solution to alleviate this condition by migrating the bottleneck NF that occupies the least sNIC resource. This incurs increased latency of a service chain since packets have to travel through PCIe for two more times. To overcome this problem, Meng et al. [96] propose PAM, the push aside migration solution, to intelligently migrate the right NFs to avoid the hot spot on sNIC. Specifically, they select to migrate NFs that lie on the border between sNIC and CPU, while releasing the resources on sNIC to serve the bottleneck NFs. As a result, extra packet transmissions over PCIe can be avoided. Since it is non-trivial to select the right border NFs for migration, they further propose an effective algorithm to make the migration decisions by establishing a resource model. The evaluation demonstrates that PAM can relieve the hot-spot issue on sNIC, while reducing the latency of a chain of NFs by 18% compared with UNO.

4.1.2 Software Tuning. Software tuning refers to a set of optimization techniques in order to achieve high performance. There may exist many tuning techniques in the literature, while we choose to present those common ones including CPU pinning, zero-copy, batch processing, NUMA-aware, lockless and parallelism techniques. Table 4 gives a brief description of these techniques.

Software tuning techniques are widely adopted in many existing works. A representative example is Hyper-Switch [115]. As a scalable software-based virtual switch for virtualization platforms (such as Xen [12]), one of its features is to combine with the advantages of both driver domains and hypervisors, while eliminating the software overheads for achievable I/O performance inherent in hypervisors, without incurring associated memory sharing overheads for packet delivery between VM and the NIC driver. The other feature is a set of optimizations that enable high performance. Specifically, Hyper-Switch removes the costs of hypervisor entries and guest announcements by using state-aware batching of packets, reduces the packet arrival latency by employing preemptive copying, and considers CPU cache locality when using low-overhead techniques to dynamically offload packet processing to idle CPU cores, respectively. All these optimizations yield improved network performance. In the evaluation for measuring the total inter-VM network throughput, Hyper-Switch reaches throughput up to 81 Gbps while it only achieves about 31 Gbps and 47 Gbps on Xen and KVM, respectively.

To eliminate the performance overheads in a virtualized environment such as VMs or containers, NetBricks [108] takes a bold step for building and running NFs in a bare-metal environment while providing the equivalent isolation in software, resembling VMs and containers. By allocating a dedicated CPU core per NF or running an NF chain on a single CPU core, NetBricks shows several to ten times better performance than the case of NFs running on the basis of a VM or container, for both individual NF and NF chain. NetBricks also introduces a novel technique called zero-copy software isolation (ZCSI) and runs each NF chain as a single process to reduce packet I/O cost. To maximize the performance, each NF chain is entirely placed on the same CPU core in NetBricks, while replicating the processing graph across cores for scaling. NetBricks implements run-to-completion (RTC) processing of each packet and round-robin scheduling to schedule packets among processing nodes. In particular, the RTC model used in NetBricks ensures a packet to be continuously processed from the time it enters an NF to the time it exits, which can avoid inter-core transfer and reduce context switching cost so as to improve the performance. Besides, vTurbo [144] offloads a process to a specialized core called turbo core, which is separated from a CPU core with its exclusive time slice. Offloading workload to dedicated turbo core mitigates the influence of VM's regular core access latency. As a result, vTurbo prominently improves application-level performance in terms of network throughput.

Note that some works may belong to different acceleration categories. In terms of computation acceleration, for instance, NetVM [56] applies many low-level software tuning techniques in its system design. Building upon KVM hypervisor and Intel DPDK framework, NetVM inherits several advantages from DPDK. By using huge pages provided by DPDK, NetVM can work in a NUMA-aware manner. When forwarding packets between the VMs and host, NetVM only needs to copy a packet descriptor through the shared memory, while the packet data still stays in huge page without any copy operations. For shared memory management, NetVM achieves lockless memory design by allocating cores to each parallelized queues, so that the context switching for a contested lock can

Table 4. Common software tuning techniques.

Technique	Idea	Effect
CPU pinning	Execute a process (thread) only on the designated CPU	Maintain processor affinity and increase cache hits
Zero-copy	Do not copy data from one memory area to another	Save CPU cycles and memory bandwidth
Batch processing	Process multiple packets in batches	Reduce the traversals on the PCIe and I/O buses
NUMA-aware	Enable a CPU accessing its own local memory	Obtain faster memory access
Lockless	Manage shared memory in a lockless fashion	Avoid synchronization overhead
Parallelism	Perform multiple tasks simultaneously at the same time	Increase task processing speed

be avoided. As a result, though NetVM uses multiple pipelined VMs to compose complex NFs, it still achieves throughput up to 10 Gbps, which improves the throughput by over 2.5 times compared with the performance of SR-IOV. IX [14] is a user-space networking stack that contains several tuning techniques. IX provides a native, zero-copy API, supporting a batch mode for handling packets until they are finished. Namely, for each network processing stage, the data planes of IX execute all packets in batches. This approach amortizes API overheads and results in a good locality of both instruction and data. To support scalability for multi-core architectures, IX exploits multi-queue NICs to consistently hash the incoming packets to specific queues. This removes the synchronization overhead when running multiple NFs in the data plane. We will further discuss IX in Section 4.2.2 for its hardware-assist I/O processing.

In addition, some reviewed works such as DHL (batching) and NetBricks (zero-copy technique) also exploit these tuning ideas to assist them to achieve high performance.

4.1.3 Modularized Reuse. In practice, different NFs often perform the same or similar processing steps on the same packet. These similar steps are referred to as processing blocks in a processing graph, which is abstracted from an NF. Packets will be processed by each block with individual logic unit. Through in-depth inspection, these processing blocks show that they have very simple operations on packets, such as just drop and alert, and also some complicated functionalities such as payload match of packets or decompressing specific packets. Effective reuse of these processing blocks on NF chains yields prominent performance improvement, especially in terms of per-packet latency. We call this type of computation acceleration as modularized reuse.

For this purpose, CoMb [125] first discloses that middleboxes, which are built and managed as standalone devices, incur the inefficiencies in the use of infrastructure hardware resources as well as network management. Therefore, in CoMb, software-centric implementations of middleboxes are re-architected at both device- and network-level, where consolidated middleboxes share the same hardware platform. To achieve software element reuse, CoMb implements low-level processing components for capturing packets, parsing packet headers, reconstructing TCP sessions, etc. Even though running multiple reusable NF modules on the same hardware platform brings challenges on performance isolation, security and fault tolerance, and CoMb has not addressed the implementation challenges (e.g., software architecture, performance optimization, etc.), it still lights the way for modularized reuse in NFV platforms.

Chowdhury et al. [22] follow up the work of CoMb and propose a disaggregated packet processing architecture, named by MicroNF (μNF in short) based on the concept of CoMb. In μNF [22], the authors also demonstrate that monolithic NFs can lead to hardware resource wastage due to the repeated implementation and execution of common functionality in SFCs, such as packet parsing, packet classification, and session state reconstruction. μNF is a flexible service composition architecture that uses reusable, lightweight, independently deployable loosely-coupled components and communication primitives for composing and implementing NFs for SFCs. Performance optimizations including multi-socket NUMA machines and low-latency packet processing functionalities are also implemented in μNF. Through evaluation, the authors verify that μNF-based SFCs outperform monolithic NF based SFCs in throughput with fewer CPU cycles per packet on average. To further deal with the frequent packet transfer and expensive hardware resource consumption in modularized SFC, CoCo [97] provides a performance-aware approach for deploying modularized SFCs. In order to minimize hardware resource consumption and packet transfer cost, Coco compactly consolidates a set of lightweight components on the same VM (i.e., CPU core). Furthermore, instead of scaling out a monolithic NF, Coco only scales out the element which is overloaded, so as to reduce the scaling overhead. The proposed push-aside scaling up mechanism in Coco can effectively avoid occupying new CPU cores and further degrading the performance.

Slick [7] also identifies the potential in implementing custom, fine-grained packet processing blocks that could be reused throughout many NFs. They propose a solution for programming NFs via a central controlling strategy, by writing a high-level control program to determine what processing should be performed and how traffic should be routed to. However, Slick focuses on efficient resource utilization and has scalability issues since its elements cannot be shared within different NFs and only supports software data plane units. Instead, OpenBox [17] presents a software-defined framework for NFs that maximally utilizes modularized reuse acceleration. To manage NFs efficiently, OpenBox totally decouples the control plane from its data plane (where the processing steps are executed) for each NF using a novel protocol, while similar data plane entities are defined as OpenBox Instances (OBIs). The deployment of application logic is under the control of the OpenBox Controller (OBC), which is a centralized control plane. The OBC provides an abstraction layer to specify the NF logic as processing graphs and is responsible for merging multiple NF graphs so that each packet can go through as few blocks as possible. That is, the length of route between input and output blocks of a merged graph is shortest by reusing the software modules across NFs. OpenBox also supports hardware-based OBIs for accelerating data plane processing and customized building blocks in the control plane for developing new NFs. Consequently, OpenBox gains a significant improvement in network performance, while having great flexibility. However, OpenBox mainly deals with the case that a modularized SFC is consisted of repeated elements whose internal rules pertain to different NFs and not conflict with each other.

As a more aggressive variant of OpenBox, SNF [73] synthesizes NF service chains by removing the redundant operations (i.e., packet and I/O operations) for each traffic class. To merge packet processing graphs, SNF employs the technique of graph search and graph set, which can build an equivalent NF with a single read and single write operation for each packet. Thus, it gains much higher performance than a chain with redundant operations, especially the deployment case with longer chains of NFs running on an NFV platform. The limitations of this approach are also obvious [30]. First, SNF is not suitable for every service chain. In other words, not all chains of NFs can be synthesized to an equivalent NF with a single read and single write operation. Second, SNF does not scale on multi-core architectures, which are critical for large scale NF deployment. Finally, the efficiency of SNF relies heavily on the network operator's control and knowledge about the service chain. Inherited from SNF, Metron [72] integrates the benefits of SNF into its system design to eliminate processing redundancy (e.g., inter-core transfers), which achieves a potential L1 cache speed for processing packets. Different from SNF, Metron allows using multiple threads to leverage multiple cores within each NF, while exploiting hardware-based dispatching that processes specific traffic classes to those cores. However, Metron also inherits the limitations of SNF and requires special hardware equipment (e.g., OpenFlow Switches).

While most of the existing systems concentrate on achieving efficient packet processing across a service chain in L2/L3, they fall short in constructing more complex NFs at the transport layer and above. These complex NFs often require additional processing. For instance, if each NF in a chain performs TCP processing, TCP bytestream reconstruction may be performed multiple times, which results in extra resource consumption. To address the problem, a framework called Microboxes [92] is proposed to support transport-layer-and-above (L4-L7) NFs or even end-system NFs like a caching proxy, while consolidating the protocol processing inside a chain to remove the extra work. Micorboxes falls into this category since it splits the stack into modules based on the specific functionalities, so that network operators can build customized processing for each NF or chain based on each flow, i.e., only the required parts of processing would be performed. In order to separate stacks for both NFs and endpoint applications, Microboxes further designs µEvent communication interface to transparently manage them. The evaluation results show that Microboxes can double throughput via consolidating stack processing, and improve the throughput by 51% via building customized TCP processing.

Last but not least, Chowdhury et al. [23] survey the state-of-the-art works from both academia and industry in adopting microservice software architecture and realizing modular NF design and flexible service composition for re-architecting the NFV ecosystem, including Click, CoMb, μNF, OpenBox, Microboxes, Flurries, NetBricks, etc. They conclude that building large-scale cloud applications from reusable and independently deployable components is effective and beneficial to speeding up innovation in NFV. They also outline a set of research challenges for microservice-based NFV platforms.

4.2 Communication Acceleration

Communication acceleration is to enable high network I/O speeds for network-intensive NFs, which are operated in the data plane and have to forward a large number of packets with minimum delay. Typical NFs can be gateways in mobile packet core networks, broadband remote access servers (BRASs), SBCs, etc. We further categorize communication acceleration into data plane boost and hardware assist.

4.2.1 Data Plane Boost. Data plane boost, as introduced, generally resorts to high-performance software I/O frameworks to speed up packet processing in the data plane. In the context of NFV acceleration, many works are built upon these frameworks for high-performance networking.

In respect of DPDK, CuckooSwitch [157] creates a software switch with high scalability and resource-efficiency, for fast and compact forwarding table lookup based on the design of cuckoo hashing [31]. To process packet I/O between user-space and the NICs, CuckooSwitch employs DPDK to achieve high throughput for packet processing when handling a great deal of L2 rules. The evaluation results indicate that it can process 92.22 Mpps for a packet size of 64B when using eight 10-Gbps Ethernet interfaces on a COTS server. However, the benefits of CuckooSwitch are at the expense of losing flexibility in switching logic since the hardware Ethernet switches are replaced. The reviewed work NetVM, together with its container-based version OpenNetVM [154], is developed with more requirements for achieving high-speed communication. By taking advantage of DPDK, both NetVM and OpenNetVM are capable of running complex NFs at line rate (10 Gbps) on COTS servers. Moreover, they also enable high-speed inter-NF communication so that complex NFs can be spread across multiple VMs or containers. In particular, OpenNetVM greatly simplifies the development, management and optimization of NFs. Instead of running NFs on VMs, OpenNetVM runs NFs using lightweight Docker containers. Consequently, it offers NFs flexible control for routing packets in service chains with higher throughput. The evaluation results demonstrate that OpenNetVM achieves a throughput of 68 Gbps when the load between two NF replicas is balanced. For a chain of five NFs, it still achieves a throughput of up to 40 Gbps which proves its deployment potential in carrier networks. There are also some other works that are built on DPDK, such as the reviewed work DHL in FPGA-based solutions, and NFP [132] which we will describe it later (in Section 4.3) to highlight its NFV-specific from-serial-to-parallel acceleration.

Regarding netmap, mSwitch [54] extends the VALE switch to provide a logically separated switching fabric and the switching logic, thus achieving high performance with flexibility. Building on netmap, mSwitch accelerates the packet delivery as an in-kernel module, which provides a high-performance packet I/O and secure data path using memory buffers between virtual ports. Considering the scalability to serve numerous virtual ports, it is designed to support up to 120 virtual ports. When multiple NFs arrive at a shared destination NIC simultaneously, mSwitch implements a mechanism of destination port parallelism to improve the throughput. The validation of mSwitch shows that a learning bridge outperforms the FreeBSD one by 8 times; a modified OvS with minor code changes results in a up to 3 times speedup; a filtering module permits mSwitch to route packets to NFs (and discard undesired NFs); and a module can be performed as user-mode protocol stacks. ClickOS [94] is another well-known netmap-based platform on Xen hypervisor optimized for NF processing. For high-performance networking, it replaces the default software switch OvS on Xen with the VALE switch, which can be used to interconnect VMs using the netmap API, so that the bottlenecks of OvS can be avoided. ClickOS also makes considerable changes to the Xen's I/O subsystem, such as virtual network devices and the corresponding drivers. By running many lightweight VMs concurrently in a single commodity server, ClickOS achieves high scalability while reaching line rate on a 10-Gbps link. However, the development environment for NFs is fairly limited since the NFs must be designed with Click's predefined processing elements, and does not run within a standard Linux OS. To alleviate the virtualization overhead without the help of hardware-dependent technology such as SR-IOV and hardware passthrough, Garzarella et al. [39] propose ptnetmap, a netmap-based virtual passthrough network device. Ptnetmap is totally decoupled from the hardware, thus yielding high-speed communication among NICs, software switches, or netmap links. The features of ptnetmap lie on the ability to provide high throughput for both trusted and untrusted VMs (not supported in ClickOS and NetVM) and the unnecessity of dedicated polling cores/threads that are required for DPDK-based solutions. As a result, using ptnetmap, VMs can obtain line rate in a 10-Gbps link, and achieve over 20 Mpps and 70 Mpps to untrusted and trusted VMs, respectively.

There are also some works that integrate both DPDK and netmap in their system design. FastClick [10] is just such an example. Through reviewing a variety of characteristics provided by many high-performance software frameworks, the authors of FastClick derive modern general principles for the software design. Both DPDK and netmap version of FastClick are implemented. In comparison with other software NFs such as an IP router, FastClick shows up to about 2.3 times speedup.

In addition, Jose et al. [67] investigate how to utilize P4 for compiling programs to programmable data planes (e.g., the RMT and Intel's FlexPipe architectures). They first propose the design of a switch compiler utilizing abstractions to replace hardware implementation while mapping logical lookup tables to physical tables. In order to further optimize pipeline occupancy, latency or power consumption, they explore the interplay between the greedy-based algorithms and Integer Linear Programming (ILP) solutions. However, they only compile stateless data plane operations to RMT and FlexPipe (e.g., simple forwarding and routing), while not handling the stateful data plane tasks.

In the context of NFV, the vSwitch plays a key role in achieving high performance. Hence, a user-space vSwitch called SnabbSwitch [109] is devised to aim at carrier-grade performance. The novelties of SnabbSwitch exist in the implementation of vhost-user as well as the use of a trace compiler. By conducting extensive benchmarking experiments, SnabbSwitch is proven to outperform OvS-DPDK and can yield almost the same performance compared with hardware-based solutions like SR-IOV.

Maybe due to the limitation that OpenOnload [129] only supports its own NICs from Solarflare series and HP 570SFP+, few works can be found that are built on it for NFV acceleration. About PF_RING, more information can be found in [38], including the comparison with DPDK, netmap and Snabb.

4.2.2 Hardware Assist. To eliminate the overhead of virtualization, it has been long to use hardware passthrough technology for high performance. As it has some disadvantages such as a limited number of PCIe devices, another promising technology called SR-IOV is introduced to realize the scalability by creating more VFs, as described in Section 3.2.

There are some hardware-based solutions that rely on SR-IOV technology for high performance. Among them, Arrakis [111] exploits this hardware feature to get direct access to vNICs for eliminating the overhead of the OS process abstraction in the kernel. Specifically, it splits the traditional kernel in two roles, which not only allows packet processing to entirely bypass the kernel, but also provides both network and disk protection without the involvement of the redesigned kernel. The evaluation results show latency reduction by 2-5 times and throughput improvement by 9 times for a well-known NoSQL store compared with native Linux. Besides, IX [14] also relies on SR-IOV to implement a high-performance data plane OS. Based on hardware virtualization with multi-queue NICs, IX separates control plane functions such as management and scheduling from the data plane that focuses on packet processing. The data plane architecture is built on some software tuning techniques, as introduced in Section 4.1.2. Through designating networking queues and hardware threads to data planes, IX eliminates high context switching costs like synchronization overhead. The authors demonstrate that IX results in both high I/O performance and low latency by using its user-space networking stacks.

Kourtis et al. [79] exploit both SR-IOV and DPDK, and choose DPI as a representative use case. They implement the DPI NF using libpcap, DPDK and SR-IOV, respectively, thus demonstrating the benefits of using SR-IOV enabled devices with DPDK to enhance the high performance of DPI. The evaluation results show that using SR-IOV and DPDK jointly can achieve significantly better throughput, compared to traditional Linux protocol stack.

However, hardware-assist solutions inevitably limit the flexibility of the actual development of NFV as they break the software abstraction. Concretely, VMs or containers must be initiated on servers where SR-IOV capable NICs are placed, and the features offered to the VM depending on the capabilities of the specific NIC hardware. SR-IOV-based solutions also do not support VM migration. Moreover, theoretical limits are non-negligible factors when employing this technology. That is, most NICs can effectively support only about 8-16 VFs for 1-GbE ports and 60-64 VFs for 10-GbE ports, otherwise it will experience a performance hit for supporting more VFs.

4.3 From Serial to Parallel Acceleration

Note that current NFV acceleration efforts concentrate mainly on sequential service chaining, i.e., performing each NF of a service chain from a horizontal scope. A study from industrial networks indicates that 53.8% NF can be parallelized as a pair, i.e., the actions on packets of an NF pair will not conflict with each other. On the premise of satisfying parallelism, 41.5% NF pairs will not cause extra resource consumption [132].

Based on these observations, Sun et al. [132] propose a high-performance framework called NFP, that allows parallelized NF to boost the performance of service chains, especially from the perspective of latency reduction. Since NFP enhances NFV performance from a vertical scope and runs NF in parallel as much as possible, we categorize this type of NFV acceleration as *from serial to parallel acceleration* for service chains. To achieve this, NFP implements three logical components: (1) a policy specification strategy to intuitively define various NFs with different forms of connectivity into three types of rules, so that the parallelism optimization effect can be largely improved, (2) NFP orchestrator to determine NF dependencies and automatically translate the policies that describe the chaining intents into desired service graphs while incurring little resource overhead, and (3) NFP infrastructure that merges the processed packets to avoid consuming extra network bandwidth resource and uses a zero-copy packet delivery mechanism to support NF parallelism. With the help of these effective designs, NFP could optimize the performance of service chains with up to 35.9% lower latency.

Coincidentally, Zhang et al. [156] also discover the parallelism opportunities from chains of NFs. In response, they present ParaBox, a novel hybrid NFV framework that performs packets to NFs in parallel as far as possible, while merging the output together into the remaining sequential NFs. To guarantee correctness, they address a series of challenges including the analysis of NF parallel opportunity and the design of lightweight merge function. Through prototype implementation on top of the DPDK-enabled BESS, ParaBox shows not only service chain latency reduction, but also throughput improvement.

4.4 Traffic Steering Acceleration

It is crucial for an NFV platform to efficiently steer traffic through a service chain. From the time a packet enters the system until it is processed by the last NF, each link needs to decide where to process the packet. Without proper acceleration, the speed of traffic steering would fall behind the desired steering capability for maintaining the high performance of a service chain. Therefore, existing research efforts have proposed several traffic steering models in order to solve this problem.

Industry solutions such as NSH [114] encapsulate packets with the required information for forwarding through a service chain, but at the cost of increasing packet size.

In general, *traffic steering acceleration* is to exploit efficient steering models for providing flexible and efficient switching capabilities, in order to reduce the overhead incurred from inter-core communication and context switching, or to handle more complex situations (e.g., change some NFs during the life of a TCP session, support high-layer NF processing) in steering traffic. Considering that most NFs are stateful, the state of an NF often needs to be shared across multiple instances when performing elastic scaling [40, 68, 141]. Concretely, OpenNF [40] incurs high overhead during scaling events as it must first migrate all required states to the new NF instance for local access, while StatelessNF [68] heavily sacrifices the performance since all state accesses are remote. In response, S6 [68] improves the performance by proposing a distributed shared state model. Elastic NF scaling involves a problem that migrates states across NF instances, while ensuring a packet to be dispatched to the NF instance which has its state for processing the packet. The requirement is called the affinity between packets and their states, which should be considered in traffic steering. In the following, we present four steering models according to the literature.

4.4.1 Software Shim-Layer Model. This model relies on a software shim-layer (e.g., software switch) with installed forwarding rules to determine which NF or NIC that a packet should be routed to. The software shim-layer serves as an intermediate stop since each packet has to travel the shim-layer twice when it needs to transmit from one NF to another NF (or NIC).

Many existing works adopt this model due to its flexibility to move packets between NFs via a service chain. Some surveyed works, for example, NetVM [56] uses a global switch in the hypervisor to alleviate overhead of OvS in traffic steering. Thus, it provides a flexible switching capability without copying packets, which goes beyond DPDK vSwitches. ClickOS [94] replaces the default OvS with the VALE switch to allow VM switching using the netmap API. Equipped with sNIC, UNO [84] uses two switches in its hypervisor, one connects NF instances in place and the other connects to offloaded NFs. OpenBox [17] runs all NFs in VMs on top of KVM hypervisor, and implements a traffic steering application based on the OpenDaylight controller. Another work E2 [106] uses BESS to allow packet processing modules to be dynamically configured. Though with great flexibility, this model introduces considerable inter-core communications and thus affects performance.

4.4.2 Process Flow Model. With containerized NFs introduced, there are two forms to build a service chain. Run-to-completion (RTC) form initializes NFs as native processes and consolidates an entire SFC on the same CPU core. The pipelining form uses one core for each NF and constructs a service chain with multiple cores. To steer packets through the service chain, the process flow model usually contains two kinds of threads for receiving (RX) and transmitting (TX) packets respectively. No matter which forms to use, if regarding the threads that run NFs as a whole and combining two kinds of threads above, it shows a complete procedure for handling packets. Instead of relying on a shim layer, steering packets through a chain are in charge of the threads.

Taking OpenNetVM [154] as an example, packets from NICs are constantly read by the RX thread, which then puts the packets into the buffer queue of the target NF after looking up the flow table. After processed by the NF, the packets are then moved by the TX thread to subsequent NFs in the chain until forwarding out through another NIC. Two other works, NFP [132] and Flurries [152] that both built on top of OpenNetVM, also belong to this model.

Due to the fact that a packet is no longer required to go through each thread twice, this model incurs fewer inter-core communications compared to the shim-layer model.

4.4.3 Hardware-integrated Model. Apart from the software-based techniques above, existing research efforts also consider making full use of features provided by hardware (e.g., NICs, OpenFlow Switches) for faster traffic steering. We call this hardware-integrated model since it leverages the combination of software and hardware. As stated earlier, current NICs usually support two features: (i) Receive Side Scaling (RSS) does hashing on specific fields of a packet and (ii) Flow Director provides packet classification and dispatching capabilities, both of which help to dispatch packets to multiple or specified queues as well as specific cores. With high programming flexibility, OpenFlow switch performs packet lookup and forwarding according to flow tables at the speed of hardware.

Earlier surveyed works, like NetBricks [108], FastClick [10], SNF [73] and a new work FastPaas[155] all exploit RSS for traffic steering. Specifically, both NetBricks and FastPaas use RSS to direct packets to a CPU core, where all NFs in an SFC run as a single thread in a bare-metal environment. This avoids inter-core overhead by processing a packet via an SFC entirely on the same core. Moreover, FastPaas also leverages a new hardware technique, Intel MPX, to enable memory isolation among processes on the same core. FastClick uses RSS to partition received packets over multiple queues, thus enabling the reception of packets from the same device. SNF uses symmetric RSS to ensure that packets of a flow are always sent to the same processor. Instead, CoMb [125] uses flow director for dispatching traffic, which mainly focuses on consolidating NFs. Besides, OpenNF employs hardware OpenFlow Switch, in which a central controller installs and updates forwarding rules for dynamic service chaining. Recently, Metron [72] proposes to leverage both features of hardware OpenFlow switch and the NIC, for dispatching stateless and stateful operations respectively. By this means, Metron makes it possible to process packets at true hardware speed.

Without the aid of either a software shim-layer or dedicated threads, this model incurs just one or even zero inter-core communications.

4.4.4 *Protocol Level Model.*

To deal with the situation for more complex NF processing, there are also some works that advocate employing a protocol-level model for traffic steering, which enables dynamic service chaining or more efficient NF processing.

Dysco [147] proposes a session-level protocol to dynamically adjust a service chain, including the addition, removal and replacement of NFs, and schedule traffic between anchor nodes. Dysco adds a daemon to the Dysco agent for each NF, while adding a global controller for full-view management. Compared to an insufficient SDN-based approach, Dysco can reconfigure and change NFs in a variety of situations. As introduced earlier, another work Microboxes [92] is designed to support the proper level of protocol processing or even end-system NFs with modular engines, while eliminating extra work during protocol processing. Microboxes generates and consumes events rather than packets, thus supporting a wider spectrum of NF types.

4.5 Discussion and Comparison of the Reviewed Works

In the previous sections, we have reviewed various acceleration works for addressing the performance issues in NFV based on our taxonomy in Figure 3. We sort each work into the corresponding categories in Table 5, along with some key characteristics such as acceleration method, application scope and adopted techniques. Readers can refer to it to get a quick and general understanding of each acceleration work.

Note that some works may belong to multiple categories, depending on the practical system design. The acceleration methods of these surveyed works can also contain multiple types, ranging from the used hardware and software frameworks to the adopted tuning techniques and optimization design (i.e., reuse, pipeline and parallelism). In some works, we omit the least important acceleration methods to highlight the dominant ones. Specifically, for the adopted techniques, we choose zero-copy, batch processing and parallelism from many common performance-enhancing techniques since they are widely used throughout many works. Some works are designed for service chains, while some focus on specific NFs including IDS, cryptography, software switches and IP routers. Nearly all surveyed papers carry out their experiments on a real testbed consisting of one or more servers. This makes the results more convincing than simulation experiments. For the objective, a majority of studies consider increasing the performance of their systems, either in terms of high throughput or low latency. Certainly, there are also some works that focus on other objectives such as high scalability, flexibility and load balance. Similarly, we just present the primary objectives while leaving out the secondary ones. These objectives are orthogonal and complementary to high-performance achievement.

There are no ideal solutions since the works in each category have their advantages and disadvantages. For the hardware offloading solutions in computation acceleration, the high performance

Table 5. A Taxonomy of representative works for NFV acceleration.

Work	Category	Acceleration Method	Application	Technique			Evaluation Setup	Objective
				zc	bat	par		
NP-Click [126]	comp	NPU, Click	Software router			✓	Two applications	High performance
Ferkouss et al. [35]	comp	NPU	OpenFlow switch				Numerical results	Good performance
POF [130]	comp	NPU	OpenFlow switch				Prototyping	Protocol-oblivious
PacketShader [52]	comp	GPU	Software router		✓	✓	Local testbed	High throughput
NBA [77]	comp	GPU	Packet processing	✓	✓	✓	Local testbed	High performance
Snap [133]	comp	GPU, Click	Software router	✓	✓	✓	Commodity PC	High performance
DoubleClick [76]	comp	GPU, Click	Software router		✓		Local testbed	High throughput
GASPP [136]	comp	GPU	Stateful processing	✓	✓	✓	Local testbed	High throughput
Kargus [59]	comp	GPU	Intrusion detection		✓	✓	Two workloads	High throughput
Gnort [135]	comp	GPU	Intrusion detection		✓	✓	Local testbed	High throughput
MIDeA [137]	comp	GPU	Intrusion detection		✓	✓	Local testbed	High throughput
SSLShader [61]	comp	GPU	Cryptography		✓	✓	Local testbed	High throughput
GPUNFV [146]	comp	GPU	Service chain	✓	✓	✓	Local testbed	High performance
APUNet [43]	comp	APU	Network application	✓	✓	✓	Local testbed	Revitalize GPU
G-NET [149]	comp	GPU	Service chain	✓	✓	✓	Local testbed	Spatial GPU sharing
ClickNP [88]	comp	FPGA, Click	Packet processing NF		✓	✓	Local testbed	High performance
Emu [131]	comp	FPGA	Any NF			✓	Local testbed	Rapidly prototype NF
DHL [91]	comp	FPGA	Specific NF		✓	✓	Local testbed	High flexibility
UNO [84]	comp	sNIC	Packet processing NF	✓		✓	Local testbed	Unifying offload
Floem [113]	comp	sNIC	common NF tasks		✓	✓	small-scale clusters	High throughput
PAM [96]	comp	sNIC	Service chain	✓			Local testbed	Low latency
Hyper-Switch [115]	comp	Tuning	Virtual switch		✓	✓	Prototype in Xen	High scalability
NetBricks [108]	comp	Tuning	Packet processing NF	✓	✓	✓	Local testbed	High I/O efficiency
vTurbo [144]	comp	Turbo core	VM		✓		Local testbed	High throughput
NetVM [56]	comp, comm	Tuning, DPDK	Complex NFs	✓	✓	✓	Local testbed	High performance

(Continued)

Table 5. Continued.

Work	Category	Acceleration Method	Application	Technique			Evaluation Setup	Objective
				zc	bat	par		
IX [14]	comp, comm	Tuning, SR-IOV	Key-value store	✓	✓		Testbed cluster	Throughput, latency
Slick [7]	comp	Reuse	Service chain				Mininet emulation	Programming NF
OpenBox [17]	comp, tras	Reuse, RSS	Service chain				Local testbed	High performance
SNF [73]	comp, tras	Reuse, RSS	Specific NF chains		✓	✓	Local testbed	High throughput
Metron [72]	comp, tras	Reuse	Specific NF chains		✓	✓	Local testbed	Hardware speed
Microboxes [92]	comp, tras	Reuse, DPDK	Complex NFs	✓	✓	✓	CloudLab testbed	High throughput
CuckooSwitch [157]	comm	DPDK	Software switch	✓	✓	✓	Local testbed	High performance
OpenNetVM [154]	comm, comp	DPDK, tuning	Service chain	✓	✓		Local testbed	High performance
mSwitch [54]	comm	netmap	Software switch		✓	✓	Local testbed	High scalability
ClickOS [94]	comm	netmap	Various NFs		✓		Local testbed	High performance
ptnetmap [39]	comm	netmap	Virtual device	✓	✓		Local testbed	High performance
FastClick [10]	comm, tras	DPDK, netmap, RSS	IP router	✓	✓	✓	Prototyping	Fast I/O
Jose et at. [67]	comm, tras	P4	Software switch			✓	Benchmarking	Low latency
SnabbSwitch [109]	comm	Snabb	Specific NFs	✓			Benchmarking	High throughput
Arrakis [111]	comm	SR-IOV	Widely used NFs	✓		✓	Cloud workloads	Latency, throughput
Kourtis et al. [79]	comm	SR-IOV, DPDK	DPI				Case study	High throughput
NFP [132]	fstp, tras	Parallelism, proflow	Service chain	✓	✓	✓	Local testbed	Low latency
ParaBox [156]	fstp	Parallelism	Service chain		✓	✓	Three machines	Low latency
E2 [106]	tras	BESS	NFV applications	✓	✓		Local testbed	High performance
Flurries [152]	tras	proflow	Service chain	✓	✓		Local testbed	Customizability
RouteBricks [28]	tras	RSS, Click	Software router		✓	✓	Server Parallelism	High throughput
CoMb [125]	tras	Flow Director	NF deployment			✓	Prototyping	Load balance
FastPaas [155]	tras	RSS, MPX	Service chain		✓		Test server	Memory isolation
Dysco [147]	tras	Session protocol	Service chain		✓		Eleven hosts	Traffic steering

Annotation: comp - computation acceleration; comm - communication acceleration; fstp - from serial to parallel acceleration; tras - traffic steering acceleration; Reuse - modularized reuse; Parallelism - service chain-level parallelism; proflow - process flow model; Tuning - software tuning; zc (zero-copy), bat (batch processing), par (parallelism)

(a) ACE-NIC 100.

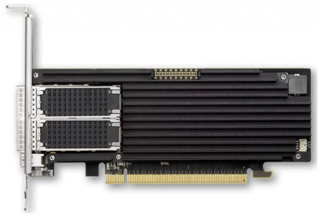

(b) NT200B01 sNIC.

Fig. 4. The product photos of ACE-NIC 100 and NT200B01 sNIC.

is at the expense of hardware dependency. Besides, for instance, PacketShader incurs high latency for batch operations; NBA has data transfer bottleneck in PCIe communication; NetBricks requires rewriting NFs as it abandons VMs or containers to provide isolation. Some works only support specific NFs and have difficulty in programming since they are limited by the component capabilities they use, such as ClickNP and SNF. Other works like OpenBox do not provide isolation among NFs, and Slick fails to share elements across multiple NFs. For those solutions that use software I/O frameworks, there are also many factors that limit the performance. Specifically, the switching logic in CockooSwitch is inflexible and CockooSwitch relies on OvS which will harm the performance, while the same case occurs in Hyper-Switch. The protection mechanism of the host stack is sacrificed in OpenNetVM. In contrast, the runtime of ClickOS for developing NFs is very restricted as it does not support a standard Linux environment [154]. For from serial to parallel acceleration, the most important problem is how to balance the tradeoff between the parallelized overhead and the performance improvement. Finally, though Metron reduces the inter-core packet transfers, it uses SNF so it is only applicable to fairly limited service chains.

5 ECOSYSTEM AND APPLICATION OF NFV ACCELERATION IN INDUSTRY

Besides the widespread study in academia, NFV acceleration has also witnessed a great deal of success in industry. According to the prediction of IHS Markit report [93], the NFV market is expected to increase at a compound annual growth rate of 42% by 2020, reaching a market scale of 15 billion US dollars. Many solutions and products have been developed by vendors and providers. There are also a number of industrial projects that have been launched for high-performance requirements. All of these form an integrated ecosystem of NFV acceleration in industry, proving the feasibility from academic research to practical applications.

5.1 Emerging Products and Solutions in Industry

For obtaining carrier-grade performance in telecom, Ethernity introduces ACE-NIC 100 [101], a sNIC that provides up to 100 Gbit Ethernet connectivity. ACE-NIC is built on carrier-grade flow processor technology called ENET and offers hardware acceleration using programmable FPGA. As such, this sNIC provides first-class performance for both telecom and enterprise NFV networks by offloading specific NFs onto an optimized adapter. To reduce the overall CPU overhead, ACE-NIC provides three types of product features. First, considering that DPDK requires dedicated CPU cores to process flows, ACE-NIC 100 can offload the DPDK applications onto the NIC, thus improving the quality of experience for NFs. Second, ACE-NIC offers security acceleration via offloading IPSec and implements end-to-end encryption for security purpose, while the offload functionalities provided by traditional NICs are still reserved. Third, to manage VM traffic efficiently, ACE-NIC also offloads encapsulation or decapsulation of overlay network traffic to the sNIC. Beyond that, Napatech launches its product called NT200B01 [99], a 2-port 100G sNIC specially designed for NFV. By exploiting the programmability and reconfiguration of FPGAs, NT200B01 serves as a common hardware platform that supports various acceleration solutions, while extending the lifetime of the sNIC and supporting on-the-fly configuration. It is designed to support multiple data rates such as 4×10 Gbps and 2×100 Gbps. This allows the same sNIC to be used in different locations in the network. Compared to the software-only implementation, NT200B01 achieves 10 times

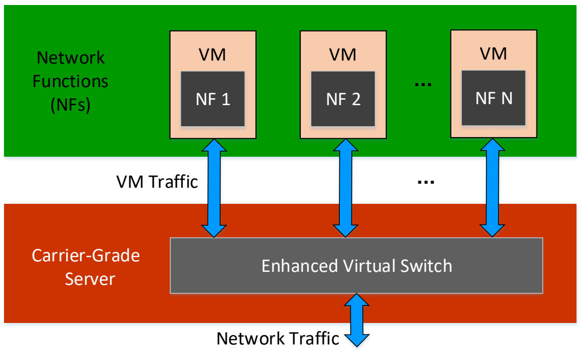


Fig. 5. An illustration of the enhanced vSwitch within the carrier-grade server for communication.

higher throughput and 20 times lower latency. NT200B01 can be applied to multiple virtualization scenarios, including virtual switch acceleration, hardware offloading and VM-to-VM monitoring. The product photos of both sNICs are shown in Figure 4.

In addition, Barefoot Tofino [11] is a new-generation, P4-based, and end-user programmable Ethernet switch, which can run up to speeds of 6.5 Tbit/s. Tofino is built on a protocol-independent switch architecture (PISA), where its forwarding logic depends on the P4 program during runtime. Since Tofino does not bake the program into the chip, it can handle different protocols by simply adding the logic to the P4 program. Jin et al. [65] present NetChain, a new approach that uses this kind of emerging, flexible and high-performance programmable switch (i.e., Barefoot Tofino switch) to provide scale-free sub-RTT (round-trip time) coordination in datacenters. As compared with current solutions like NetBricks [108] which can process tens of millions of packets per second, NetChain implemented with four Barefoot Tofino switches can process several billion packets per second, which lights the way to use Tofino switches for NFV acceleration.

Since 2014, Wind River has made a breakthrough in accelerating vSwitch optimized for NFV. Compared to standard OvS with no hardware acceleration, Wind River accelerated vSwitch delivers 20 times the performance, in terms of 12 Mpps to guest VMs with only two Intel Xeon processor cores at 2.9 GHz. As shown in Figure 5, the enhanced vSwitch resides on the Wind River carrier-grade server, which serves as an excellent platform for running software NFs [117]. The vSwitch is scalable since it supports a wide range of guest OSes (e.g., Linux distributions, Windows) running in the VMs, while supporting fast convergence during live VM migrations. In commercial use, the platform makes it possible to deliver six 9s reliability for telecom networks.

At Mobile World Congress (MWC) 2018 held in Shanghai, some companies such as Lenovo and China Mobile jointly showcase their accelerator-based cloud-RAN (radio access network) solution [85]. Cloud-RAN can accelerate packet processing with various hardware and totally decouple software from hardware. To further drive the virtualization and cloudification of mobile base stations and finally make the equipment ecosystem open, cloud-RAN is designed with programmable hardware and uniform interfaces with the help of Lenovo ThinkCloud Accelerator solution and Napatech sNICs. ThinkCloud contains carrier-grade platform software, x86 servers and switches, while Napatech sNICs offers powerful computing performance with Xilinx FPGA silicon. This solves many issues in current acceleration solutions, including customized function designs and non-uniform interfaces. With cloud-RAN, China Mobile together with many industry partners takes a valuable step towards general-purpose acceleration solution in NFV. In the near future, the five companies aim to further improve the performance with optimized solutions. Together with Lenovo and Certusnet, Intel also releases its FPGA-based solution in MWC. As early as 2015, Intel acquired Altera Corporation with 16.7 billion dollars since Intel realized the advantage of FPGA compared to NPU and GPU long before. For achieving CPU-FPGA heterogeneous computing, Intel wishes to integrate its Xeon server with FPGA, thus building the end-to-end NFV ecosystem. Based on Intel FPGA, Certusnet pushes out a series of virtual BRAS (vBRAS) solutions, including vBRAS IPv4, IPv6 and hardware acceleration solutions.

Table 6. Representative projects for data plan acceleration.

Project	Begin Time	Organization	Description
ODP	10/2013	Linaro Network Group	A collection of APIs designed for data plane on different platforms
Rocket	09/2018	OPNFV	NFV solutions for hardware acceleration and API design for accelerators
FD.io	02/2016	Linux Foundation	A collection of several sub-projects and libraries for flexible and scalable network and storage
FDS	03/2016	OPNFV	NFV solution stacks for fast and flexible networking
XDP	03/2016	Linux Foundation	A high-performance and flexible network data path

5.2 Popular Projects Established for NFV Acceleration

There are also a number of projects that have been launched in industry. A list of representative projects is presented in Table 6.

Specifically, OpenDataPlane (ODP) [49] separates its APIs from the underlying hardware architecture, and not only supports pure software implementations, but can also exploit underlying hardware features. ODP provides a set of APIs for application portability across a range of networking platforms that offer various types of hardware acceleration. Rocket [105] is proposed to provide various solutions when hardware accelerators are applied in the NFV architecture. It takes many accelerator-specific factors into account, such as price, power consumption and achievable performance of a certain accelerator type. In particular, Rocket focuses on the design of common APIs between NF and NFV infrastructure, with which NFs could employ the accelerators to satisfy the urgent needs of performance and power regardless of the NF vendors. Fast Data Stacks (FDS) [25] provides a user-space I/O framework with a modular design and high extensibility that enable developing I/O services rapidly with high throughput, low latency and resource efficiency. Both Rocket and FDS projects are within the community of Open Platform for NFV (OPNFV) [37]. FDS also creates and composes a set of scenarios which include the virtual forwarder supplied by Fast Data - Input/Output (FD.io) project [32]. Its main feature is to take advantage of CPU optimizations such as vector instructions and direct interactions between I/O and CPU, so as to deliver best-in-class packet processing performance. As part of the IO Visor Project, XDP [138] still achieves bare metal packet processing speed without bypassing the kernel. Meanwhile, the fast data path requires no modifications to the kernel and works well in cooperation with the network stack.

Note that NFV acceleration is also on the agenda in the open-source community of Open Network Automation Platform (ONAP) [104], where the initiators present the motivation of NFV acceleration, state important problems such as acceleration requirements, and call for some proposals.

6 GAP ANALYSIS AND FUTURE TRENDS

Despite the rapid development of NFV, network operators still encounter many obstacles towards large-scale NF deployments. Since the benefits of higher performance often come with considerable compromises, many approaches still face limitations when moving to real scenarios. Achieving high performance for NFV applications will continue to be a hot research topic in both academia and industry. We next identify some promising methods to improve current solutions and point out directions that have not been comprehensively explored.

6.1 Acceleration Hardware and CPU Co-design

NFV promises to reduce capital expenditures (CAPEX) by using commodity servers instead of dedicated hardware. However, the basic principle of hardware acceleration goes against this benefit commitment. Without proper acceleration, the performance gap between virtualized NF and dedicated hardware (e.g., the example of gateways in Section 1) can be up to 8 times. Despite this fact, the overall cost efficiency achieved is not very impressive since the price of acceleration hardware is much higher than the COTS servers. According to the statistics, hardware acceleration makes sense only when there is a performance improvement of more than 60%. CAPEX saves 15% when hardware acceleration increases by 100%, and the overall total cost of ownership savings is only 9%.

To solve such a contradiction, joint acceleration hardware and CPU co-design for reducing the hardware usage while achieving a similar performance is one of the appealing solutions. Instead of running NFs entirely on hardware, the co-design solution only offloads the required parts (e.g., compute-intensive) to the acceleration hardware, while the remaining parts are still running on the CPU. In this way, the hardware not only has more space for holding accelerator modules, but also makes the NFs to flexibly use the accelerator modules on demand. Meanwhile, similar performance can be achieved compared to hardware-only solutions.

The co-design of acceleration hardware and CPU is less to be explored so far since we find that only DHL and UNO are such solutions to the best of our knowledge. Moreover, there still exist many problems such as how to intelligently determine which parts to offload to the hardware for various NFs, how to efficiently schedule the hardware and CPU resources for load balancing and how to rapidly move the data between the CPU and hardware.

6.2 NF Performance Benchmarking

Benchmarking NF performance plays a key role in the smooth transition to NFV in that it enables test-before-deploy opportunities before large-scale deployments. In practical NF deployment, the NFV orchestration tools need to set up and configure the virtualization environment in advance based on the available NF profiles, for NF performance optimization [19]. Traditional methods for both data plane and control plane are applicable for NF benchmarking, but are expensive and not sufficient. In the NFV paradigm, multiple NFs from different vendors can be provisioned on the same compute node, forming a shared, multi-vendor environment. As a result, identifying the performance bottlenecks of those NFs becomes a tremendous challenge since the bottlenecks may occur at any stage during the workflow of NF processing, as we have discussed in Section 2.

Recently, a framework called VBaaS [121, 122] is proposed, which indicates the need for NF benchmarking in order to support orchestration decisions. However, it is far from straightforward to provide predictable performance for NFs in the virtualized environment. Current test solutions on the market either focus on self-generated workloads or collect the NFV infrastructure resource utilization statistics with an inaccurate understanding of NF profiles. Furthermore, adversarial workloads are required by accelerated NFV platforms, which aim to test and optimize the performance of NFs, as the tool introduced in CASTAN [110]. CASTAN has already served as a good example of NF performance benchmarking using synthesized workloads, however, more studies are necessary in this regard, especially using real user workloads. Actually, there is still no unified approach since the vendors only concentrate on their respective technologies and NF implementations.

6.3 Acceleration of Service Chains at Higher Layers

Existing NFV frameworks largely focus on L2/L3 processing, which relies on a packet-centric model to steer packets through service chains. However, complex NFs are more than simply packet processing but also have additional functionalities. For instance, at the transport layer, the flows through an IDS require full bytestream reconstruction, while a web proxy needs complete TCP endpoint termination [92]. This requires higher levels of the protocol stacks to process those NFs. To accelerate the stack processing of NFs in a chain, redundant TCP processing should be avoided, since current high-performance user-space TCP stacks (e.g., mTCP [62], mOS [60]) can only be individually used by each NF in the chain. Otherwise, the high cost of the redundant stack processing would inevitably harm the processing latency for a service chain.

Although the consolidation and customization of stack processing and higher level event communication interfaces have been proposed in Microboxes (see Section 4.1.3), the acceleration of service chains at higher layers is still in the early stage due to the fact that: most of existing frameworks

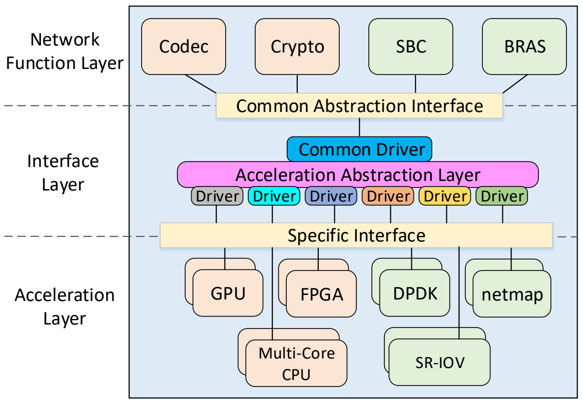


Fig. 6. An illustration of the Acceleration Abstraction Layer that enables various acceleration options used by multiple NFs, independent of the underlying accelerators.

focus on speeding up the processing of monolithic NFs rather than service chains, even in high-performance user-space protocol stacks. The modular design for the event-based communication mechanisms in Microboxes could be a good way forward, but further investigation in developing valuable high-performance frameworks is needed in order to ensure large NF deployments at L4 and the above. Furthermore, many advisable ideas in accelerating service chains at L2/L3, such as NFP and Parabox, can be borrowed in accelerating higher level service chains.

6.4 Standardization of NFV Acceleration

With the emerging of various acceleration solutions, there is an urgent demand for network operators to get easy access to these solutions. However, most of the vendors push out their respective acceleration solutions heavily based on their own business and advantages. Consequently, there is no uniform standard to manage these solutions and no unified interface for the upper layer network element to use, which are unacceptable to the network operators.

To solve the problems above, a new generation solution that exposes a common interface for the operators to exploit the acceleration capabilities may be the future evolution. This interface is generally called as an acceleration abstraction layer (AAL), which is independent of the underlying acceleration entities. Figure 6 illustrates such an interface. The AAL hides the implementation complexities that are depending on specific acceleration methods, while providing a common interface to the independent NFs. In general, the AAL is a normal feature of OSes and can be implemented with general driver models or hardware-specific drivers [44], by defining a standard set of APIs to leverage the acceleration capabilities.

Fortunately, the standardization and open source projects in the field of NFV acceleration are in good progress. European Telecommunication Standards Institute (ETSI) has released a series of IFA specifications starting in 2015, which cover many aspects for NFV acceleration, including NF interfaces specification [46, 48] and accelerator resource management specification [45, 47]. Specification [44] defines the information models and interfaces for NFV acceleration, along with some use cases. In particular, [46] defines and advocates abstracted models for different NF implementations, such as tightly-coupled software and hardware, and loosely-coupled software, in order to leverage accelerators in an implementation-independent way. Furthermore, the proposals in [46] also consider the diversity of the execution environments, acceleration frameworks or libraries and the virtualization environment. In addition, OpenStack [26] community also sets up Cyborg project [55], which intends to provide a general-purpose management platform that includes a variety of acceleration alternatives (e.g., NP, GPU, FPGA and DPDK). Cyborg keeps pace with the ETSI standards and releases its first version. The standardization of NFV needs to continue to advance in order to make NFV acceleration feasible.

6.5 Resource Scheduling and Isolation for NFV Acceleration

NFV platforms support flexible resource scheduling and scaling to satisfy the dynamically changing workload of NFs, thus how to efficiently provision resources while providing necessary performance isolation will severely affect the performance. Without proper scheduling/isolation of resources, current NFV acceleration techniques will suffer from sub-optimal resource utilization and unwanted resource contention among competing NF instances. Fortunately, some existing works have investigated this important aspect.

For example, RTNF [3] is a scalable, online resource allocation and scheduling framework, which can satisfy certain latency requirements. RTNF is designed with an efficient consolidation approach and flexible timing interfaces, which can construct SFCs from complex directed acyclic graph (DAG) based applications with optimized resource allocation at runtime. NFVnice [81] is proposed as a user-space, fair and dynamic NFV resource allocation platform, which can efficiently schedule and manage NFs for SFCs. NFVnice shows a good advantage in reducing wasted workload by using backpressure to effectively reduce previous NF's load in an SFC and carefully adjusting scheduler weights. In this way, NFVnice can largely reduce packet loss rate and improve throughput. NFVnice also provides performance isolation, which can improve the throughput of TCP flows from 30 Mbps to 4 Gbps without decreasing the throughput of UDP flows. However, NFVnice adopts poll-mode NF scheduling, which requires modifications to NFs by making them compatible with specific schedulers. To deal with the issue of assigning poll-mode NFs to shared CPU cores, UN*i*S [6] is proposed as a non-intrusive and user-space NF scheduler with workflow awareness, which can assign the maximal number of NFs to the same CPU core, without degrading the high performance for processing packets. Specifically, UN*i*S works in user space with no need to modify the kernel or implement NFs with specific scheduling logic. Thus, poll-mode NFs are regarded as black boxes in UN*i*S, while the processing order of an SFC is considered when scheduling NFs.

As we stated in Section 2.2, container-based NFV platforms have emerged to provide more efficient NF management and resource allocation (e.g., NFVnice [81] allows NFs to be deployed in Docker containers). However, providing a high level of isolation is more difficult in a container-based NFV platform than in a VM-based one, because the underlying operating system is shared by all containers built on it. To ensure high-level performance isolation, Iron [75] is proposed to monitor, charge and enforce CPU usage for processing network traffic while ensuring this processing cannot adversely interfere with co-located containers. Specifically, containers in Iron can be appropriately credited or charged for network packet processing because of its efficient charging strategy, which is integrated with the Linux cgroup scheduler. Besides, Iron's dropping mechanism can cost-efficiently eliminate the effect of a noisy neighbor that has fully consumed its resources. Moreover, Zeng et al. [148] have disclosed that NF consolidation on a single server will cause performance degradation due to the hardware resource competition. They call it NF interference, which will degrade the throughput from 12.36% to 50.30% as more NFs are consolidated on the same server. Thus, when making NF scheduling and resource allocation decisions, it is also worth considering this important characteristic to ensure high performance. Zhang et al. [150] apply a supply-demand model to capture the NF interference when making NF placement decisions in service-customized 5G network slices. The joint NF placement and resource scheduling problem has also been studied in some other state-of-the-art works. For example, NFVdeep [142] utilizes deep reinforcement learning (DRL) approach to achieve online SFC deployment and Zhang et al. [151] jointly optimize the NF chain placement and request scheduling by modeling the NFV network as an open Jackson network. Besides, there are still some valuable issues requiring constant concern and further study when designing and implementing resource scheduling and isolation for NFV acceleration, such as how to dynamically scale NFs and SFCs to cope with traffic fluctuations (e.g., Fei et al. [34] investigate the adaptive NF scaling and flow routing with proactive demand prediction and FlexNFV [33] is proposed for timely and flexible SFC scaling), how to design an effective load balancing mechanism (e.g., Wang et al. [139] propose a multi-resource load balancing mechanism with special consideration on the most stressed resource and Fei et al. [143] present a framework for joint cost-efficient SFC deployment and load balancing of multiple resources in geo-distributed central offices), how to improve the resource utilization efficiency (e.g., Jin et al. [64] study the efficient resource reuse at network edge and Finedge [89] design a dynamic, fine-grained and cost-efficient resource management for NFV network) and even how to consider the energy efficiency of NF when scheduling resources (e.g., Xu et al. [145] provide a measurement study on the power efficiency of software data planes, virtual I/O and NFs).

6.6 Scalability of NFV Acceleration Solution

Scalability is a key factor that an acceleration solution needs to consider. The scalability is reflected in many aspects. First, the used acceleration techniques require minor or even no modifications to OSes, virtual switches, hypervisors and NICs. Similarly, rewriting NFs in order to adapt to a certain high-performance framework is not allowed, and there is no necessity to implement new modules when introducing new NF functionality. The accelerated NFV system had better run in a standard development environment (e.g., Linux and FreeBSD). Second, the acceleration solution must be designed to support processors, NICs and hardware platforms from different vendors (e.g., Intel x86, IBM POWER). This makes it possible for the solution to be easily integrated into different platforms. Third, the solution is expected to offer the capabilities to launch and migrate NFs across multiple servers. This requires higher scalability requirements.

To achieve this, significant efforts have to be put into developing efficient tools for building and running NFs. These tools should allow NFV developers to use high-level abstractions, while supporting both rapid development and high performance. Besides, proof of concept associated with a new acceleration method should be conducted before large-scale NF developments, in order to validate its scalability. Many existing projects described in Section 5.2 can be used for reference in developing a scalable NFV acceleration solution.

6.7 Fault Tolerance vs. High-performance NFV Platforms

While exploring efficient NFV acceleration techniques, effective fault tolerance schemes should also be employed in high-performance NFV platforms. For example, REINFORCE [80] addresses NF chain resiliency and supports efficient fault detection and NF recovery upon link and server failures. By separating deterministic and non-deterministic NF processing, REINFORCE can minimize the performance overhead by adopting a lazy checkpointing mechanism for deterministic packet processing and a lightweight checkpointing mechanism for non-deterministic processing. Kanizo et al. [70] also design an optimal resource-constrained NF recovery strategy with performance guarantees, which mainly deals with the case that only a few NFs malfunction at the same time.

Sherry et al. [127] state that existing NFV platforms implement careful engineering to avoid failures and backup deployment for quick recovery. However, these mechanisms are either not quick enough or can not provide effective recovery for stateful NFs. Thus, they propose FTMB [127] (i.e., Fault-Tolerant MiddleBox) for rollback recovery, which employs ordered logging with restored state of NF replica during replay and parallel release for low overhead fault tolerance. To break the tight couple of state and processing, StatelessNF [68] emerges as a new NFV architecture to decouple NF into stateless processing components while maintaining the state of NF in in-memory databases (e.g., in RAMCloud). To structure the stateless NFs, StatelessNF uses high-performance DPDK-enabled pipelines for efficient network I/O, an optimized data store interface for quick access and lightweight Docker containers for flexible service provision. In order to achieve greater elasticity and failure resiliency, StatelessNF also supports instantaneous development of a new NF instance upon failure with zero-cost failure detection and flow re-direction. However, database can be a bottleneck for some NFs in StatelessNF.

In contrast to previous NFV recovery strategies, which store the NF state externally or snapshot the NF state at specific checkpoints, FTvNF [53] can effectively amortize the cost of state tracking for multiple NFs, especially for SFCs. Specifically, FTvNF can largely reduce the cost in failure-free operations where a master instance and a slave instance of a protected NF are deployed. When a failure happens, the master instance is recovering while the slave handles the traffic. Nevertheless, how to balance the recovery efficiency (i.e., maintain high performance) and the overhead of fault tolerance, and how to rapidly detect the failures, by means of active probing or passive observation, are still issues worthy of further study.

6.8 5G and NFV Acceleration

5G is the next generation of mobile network beyond the 4G long term evolution mobile networks today. Compared to 4G, 5G has more stringent performance requirements. That is, the primary goals for 5G are to improve the system capacity by 1,00 times, increase the throughput by 100 times, support 100 times more devices for connectivity, and reduce latency from 5 ms to 1 ms. As such, NFV acceleration is the foundation to deliver 5G in enhancing the viability of 5G radio access networks' functionality and architecture.

NFV overcomes some challenges of 5G when NFs are implemented by software components, but faces new problems. Researchers have put some efforts into investigating the problems after the integration of NFV and 5G. While Abdelwahab et al. [2] focus on addressing the networking problems by using a network overlay approach, and Agarwal et al. [4] seek to make optimal decisions when concerning the placement of NFs and the CPU allocation in a host, few researchers have studied how to jointly consider the challenges in NFV-enabled 5G networks with acceleration techniques introduced. In the near future, we will surely witness the involvement of NFV acceleration in the next-generation mobile network or even being a part of it, to jointly offer high-performance mobile services to end users.

7 SUMMARY AND CONCLUSION

The main obstacle for boosting NF deployments on a large scale is that the performance provided by traditional approaches is a poor match for carrier networks. To remedy this mismatch, considerable efforts have been paid to improve the performance of NFs or service chains. Due to numerous performance bottlenecks inherent in a virtualized system, designing and implementing such a high-performance platform faces many challenges. Many research works have resorted to various acceleration approaches and targeted on many different acceleration purposes.

In this article, we gave a close examination on how the performance bottlenecks occur in NFV from a systematic view and identified the key challenges in achieving high performance. We surveyed the development of acceleration techniques in the context of NFV, including common software frameworks, switches/routers and popular hardware accelerators. We presented a new taxonomy by classifying the surveyed works into four categories according to the acceleration approaches. Within the taxonomy, we reviewed all the surveyed works, and then compared these solutions to identify their respective advantages and disadvantages. We also discussed the emerging products, solutions and popular projects in industry. Finally, we analyzed the gap that exists in current approaches and highlighted some promising research trends in the future.

ACKNOWLEDGMENTS

This work was supported in part by the NSFC under Grant 61722206 and 61761136014 (and 392046569 of NSFC-DFG) and 61520106005, in part by National Key Research & Development (R&D) Plan under grant 2017YFB1001703, in part by the Fundamental Research Funds for the Central Universities under Grant 2017KFKJXX009 and 3004210116, in part by the National Program for Support of Top-notch Young Professionals in National Program for Special Support of Eminent Professionals. Corresponding author: Fangming Liu (fmliu@hust.edu.cn).

REFERENCES

[1] 6WIND. 2019. Data Plane Acceleration. Retrieved from https://www.6wind.com/products/solutions/data-plane-acceleration/

[2] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati. 2016. Network function virtualization in 5G. *IEEE Communications Magazine* 54, 4 (April 2016), 84–91. https://doi.org/10.1109/MCOM.2016.7452271

[3] S. Abedi, N. Gandhi, H. M. Demoulin, Y. Li, Y. Wu, and L. T. X. Phan. 2019. RTNF: Predictable Latency for Network Function Virtualization. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 368–379. https://doi.org/10.1109/RTAS.2019.00038

[4] S. Agarwal, F. Malandrino, C. Chiasserini, and S. De. 2018. Joint VNF Placement and CPU Allocation in 5G. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 1943–1951. https://doi.org/10.1109/INFOCOM.2018.8485943

[5] A. C. Ajayan, P. Prabaharan, M. R. Krishnan, and S. Pal. 2016. Hiper-ping: Data plane based high performance packet generation bypassing kernel on ×86 based commodity systems. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 478–483. https://doi.org/10.1109/ICACCI.2016.7732091

[6] Anthony, S. R. Chowdhury, T. Bai, R. Boutaba, and J. François. 2018. UNiS: A User-space Non-intrusive Workflow-aware Virtual Network Function Scheduler. In *2018 14th International Conference on Network and Service Management (CNSM)*. 152–160. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8584987&isnumber=8584918

[7] Bilal Anwer, Theophilus Benson, Nick Feamster, and Dave Levin. 2015. Programming Slick Network Functions. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*. ACM, New York, NY, USA, Article 14, 13 pages. https://doi.org/10.1145/2774993.2774998

[8] David Bacon, Rodric Rabbah, and Sunil Shukla. 2013. FPGA Programming for the Masses. *Queue* 11, 2, Article 40 (Feb. 2013), 13 pages. https://doi.org/10.1145/2436696.2443836

[9] Tom Barbette. 2018. *Architecture for programmable network infrastructure.* Doctoral thesis. University of Liege, Faculty of Applied Sciences, Department of Electricity, Electronics and Informatics, Liege, Belgium. http://hdl.handle.net/2268/226257

[10] T. Barbette, C. Soldani, and L. Mathy. 2015. Fast userspace packet processing. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 5–16. https://doi.org/10.1109/ANCS.2015.7110116

[11] Barefoot. 2020. Barefoot Tofino. Retrieved from https://barefootnetworks.com/products/brief-tofino/

[12] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. ACM, New York, NY, USA, 164–177. https://doi.org/10.1145/945445.945462

[13] Arsany Basta, Wolfgang Kellerer, Marco Hoffmann, Hans Jochen Morper, and Klaus Hoffmann. 2014. Applying NFV and SDN to LTE Mobile Core Gateways, the Functions Placement Problem. In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges (AllThingsCellular'14)*. Association for Computing Machinery, New York, NY, USA, 33–38. https://doi.org/10.1145/2627585.2627592

[14] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 49–65. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/belay

[15] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '05)*. USENIX Association, Berkeley, CA, USA, 41–41. http://dl.acm.org/citation.cfm?id=1247360.1247401

[16] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and et al. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[17] Anat Bremler-Barr, Yotam Harchol, and David Hay. 2016. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 511–524. https://doi.org/10.1145/2934872.2934875

[18] Z. Bronstein, E. Roch, J. Xia, and A. Molkho. 2015. Uniform handling and abstraction of NFV hardware accelerators. *IEEE Network* 29, 3 (May 2015), 22–29. https://doi.org/10.1109/MNET.2015.7113221

[19] L. Cao, P. Sharma, S. Fahmy, and V. Saxena. 2015. NFV-VITAL: A framework for characterizing the performance of virtual network functions. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 93–99. https://doi.org/10.1109/NFV-SDN.2015.7387412

[20] D. Cerović, V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle. 2018. Fast Packet Processing: A Survey. *IEEE Communications Surveys Tutorials* (2018), 1–1. https://doi.org/10.1109/COMST.2018.2851072

[21] B. Chatras and F. F. Ozog. 2016. Network functions virtualization: the portability challenge. *IEEE Network* 30, 4 (July 2016), 4–8. https://doi.org/10.1109/MNET.2016.7513857

[22] S. R. Chowdhury, Anthony, H. Bian, T. Bai, and R. Boutaba. 2019. μNF: A Disaggregated Packet Processing Architecture. In *2019 IEEE Conference on Network Softwarization (NetSoft)*. 342–350. https://doi.org/10.1109/NETSOFT.2019.8806657

[23] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba. 2019. Re-Architecting NFV Ecosystem with Microservices: State of the Art and Research Challenges. *IEEE Network* 33, 3 (May 2019), 168–176. https://doi.org/10.1109/MNET.2019.1800082

[24] Cisco. 2001. Quality of Service for Voice over IP. Retrieved from https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/qos_solutions/QoSVoIP/QoSVoIP.pdf

[25] Cisco, Intel, Redhat, Yunify, and Cengn. 2016. FastDataStacks (FDS). Retrieved from https://wiki.opnfv.org/display/fds/

[26] Rackspace Cloud Computing. 2018. OpenStack. Retrieved from https://www.openstack.org/

[27] Intel Corporation. 2007. Improving Network Performance in Multi-Core Systems. Retrieved from https://www.intel.com/content/dam/support/us/en/documents/network/sb/318483001us2.pdf

[28] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. 2009. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, New York, NY,

USA, 15–28. https://doi.org/10.1145/1629575.1629578
[29] DPDK. 2019. Data Plane Development Kit (DPDK). Retrieved from https://dpdk.org
[30] Marcel Enguehard. 2016. *Hyper-NF: synthesizing chains of virtualized network functions*. Master's thesis. KTH Royal Institute of Technology, School of Information and Communication Technology, Kista, Sweden. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-180397 TRITA-ICT-EX, 2016:2.
[31] Bin Fan, David G. Andersen, and Michael Kaminsky. 2013. MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX, Lombard, IL, 371–384. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/fan
[32] FD.io. 2016. FD.io - The Fast Data Project. Retrieved from https://fd.io/
[33] Xincai Fei, Fangming Liu, Hai Jin, and Bo Li. 2020. FlexNFV: Flexible Network Service Chaining with Dynamic Scaling. (February 2020), 1–7. https://doi.org/10.1109/MNET.001.1900483
[34] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. 2018. Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 486–494. https://doi.org/10.1109/INFOCOM.2018.8486320
[35] O. E. Ferkouss, I. Snaiki, O. Mounaouar, H. Dahmouni, R. Ben Ali, Y. Lemieux, and C. Omar. 2011. A 100Gig network processor platform for openflow. In *2011 7th International Conference on Network and Service Management*. 1–4.
[36] Linux Foundation. 2016. Open vSwitch. Retrieved from http://www.openvswitch.org
[37] The Linux Foundation. 2018. Open Platform for NFV (OPNFV). Retrieved from https://www.opnfv.org/
[38] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. 2015. Comparison of Frameworks for High-Performance Packet IO. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '15)*. IEEE Computer Society, Washington, DC, USA, 29–38. http://dl.acm.org/citation.cfm?id=2772722.2772729
[39] S. Garzarella, G. Lettieri, and L. Rizzo. 2015. Virtual device passthrough for high speed VM networking. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 99–110. https://doi.org/10.1109/ANCS.2015.7110124
[40] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling Innovation in Network Function Control. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, New York, NY, USA, 163–174. https://doi.org/10.1145/2619239.2626313
[41] Robin Giller. 2016. Open vSwitch* with the Data Plane Development Kit (OvS-DPDK). Retrieved from https://software.intel.com/en-us/articles/open-vswitch-with-dpdk-overview
[42] Inc Global Market Insights. 2018. Network Function Virtualization (NFV) Market worth $70bn by 2024: Global Market Insights, Inc. Retrieved from https://globenewswire.com/news-release/2018/07/31/1544465/0/en/Network-Function-Virtualization-NFV-Market-worth-70bn-by-2024-Global-Market-Insights-Inc.html
[43] Younghwan Go, Muhammad Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. 2017. APUNet: Revitalizing GPU As Packet Processing Accelerator. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, Berkeley, CA, USA, 83–96. http://dl.acm.org/citation.cfm?id=3154630.3154638
[44] ETSI Industry Specification Group. 2015. Network Functions Virtualisation (NFV); Acceleration Technologies; Report on Acceleration Technologies & Use Cases. Retrieved from https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/001/01.01.01_60/gs_nfv-ifa001v010101p.pdf
[45] ETSI Industry Specification Group. 2016. Network Functions Virtualisation (NFV); Acceleration Technologies; Management Aspects Specification. Retrieved from https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/004/02.01.01_60/gs_NFV-IFA004v020101p.pdf
[46] ETSI Industry Specification Group. 2016. Network Functions Virtualisation (NFV); Acceleration Technologies; VNF Interfaces Specification. Retrieved from https://www.etsi.org/deliver/etsi_gs/nfv-ifa/001_099/002/02.01.01_60/gs_nfv-ifa002v020101p.pdf
[47] ETSI Industry Specification Group. 2017. Network Functions Virtualisation (NFV); Acceleration Technologies; Acceleration Resource Management Interface Specification; Release 3. Retrieved from https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/019/03.01.01_60/gs_NFV-IFA019v030101p.pdf
[48] ETSI Industry Specification Group. 2017. Network Functions Virtualisation (NFV); Acceleration Technologies; Network Acceleration Interface Specification; Release 3. Retrieved from https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-IFA%20018v3.1.1%20-%20GS%20-%20Acceleration%20Intface%20Spec.pdf
[49] Linaro Network Group. 2013. The OpenDataPlane Project. Retrieved from https://www.opendataplane.org/
[50] Sangjin Han. 2015. Berkeley Extensible Software Switch (BESS). Retrieved from https://github.com/NetSys/bess
[51] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. *SoftNIC: A Software NIC to Augment Hardware*. Technical Report UCB/EECS-2015-155. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-155.html
[52] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2010. PacketShader: A GPU-accelerated Software Router. In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. ACM, New York, NY, USA, 195–206. https://doi.org/10.1145/1851182.1851207

[53] Yotam Harchol, David Hay, and Tal Orenstein. 2018. FTvNF: Fault Tolerant Virtual Network Functions. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems (ANCS'18)*. Association for Computing Machinery, New York, NY, USA, 141–147. https://doi.org/10.1145/3230718.3230731

[54] Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. 2015. mSwitch: A Highly-scalable, Modular Software Switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)*. ACM, New York, NY, USA, Article 1, 13 pages. https://doi.org/10.1145/2774993.2775065

[55] Yumeng Bao Howard Huang and Shaohe Feng. 2017. The Cyborg project. Retrieved from https://wiki.openstack.org/wiki/Cyborg

[56] J. Hwang, K. K. Ramakrishnan, and T. Wood. 2015. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. *IEEE Transactions on Network and Service Management* 12, 1 (March 2015), 34–47. https://doi.org/10.1109/TNSM.2015.2401568

[57] Intel. 2014. Building Enterprise-level Cloud Solutions with Outscale. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/case-studies/xeon-e5-2660-family-ssd-s3700-series-dpdk-case-study.pdf

[58] Intel. 2014. Introduction to Intel Ethernet Flow Director and Memcached Performance. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/intel-ethernet-flow-director.pdf

[59] Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. 2012. Kargus: A Highly-scalable Software-based Intrusion Detection System. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. ACM, New York, NY, USA, 317–328. https://doi.org/10.1145/2382196.2382232

[60] Muhammad Asim Jamshed, YoungGyoun Moon, Donghwi Kim, Dongsu Han, and KyoungSoo Park. 2017. mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 113–129. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/jamshed

[61] Keon Jang, Sangjin Han, Seungyeop Han, Sue Moon, and KyoungSoo Park. 2011. SSLShader: Cheap SSL Acceleration with Commodity Processors. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. USENIX Association, Berkeley, CA, USA, 1–14. http://dl.acm.org/citation.cfm?id=1972457.1972459

[62] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 489–502. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/jeong

[63] Ashton Metzler Jim Metzler. 2015. The 2015 Guide to SDN and NFV. Retrieved from https://www.emc.com/collateral/analyst-reports/2015ebook_sdn_nfv_ch2.pdf

[64] Panpan Jin, Xincai Fei, Qixia Zhang, Fangming Liu, and Bo Li. 2020. Latency-aware VNF Chain Deployment with Efficient Resource Reuse at Network Edge. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*.

[65] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. 2018. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 35–49. https://www.usenix.org/conference/nsdi18/presentation/jin

[66] Carlos Pignataro Joel Halpern. 2016. Service Function Chaining (SFC) Architecture. Retrieved from https://tools.ietf.org/html/draft-ietf-sfc-architecture-11

[67] Lavanya Jose, Lisa Yan, George Varghese, and Nick McKeown. 2015. Compiling Packet Programs to Reconfigurable Switches. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 103–115. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/jose

[68] Murad Kablan, Azzam Alsudais, Eric Keller, and Franck Le. 2017. Stateless Network Functions: Breaking the Tight Coupling of State and Processing. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 97–112. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kablan

[69] Anuj Kalia, Dong Zhou, Michael Kaminsky, and David G. Andersen. 2015. Raising the Bar for Using GPUs in Software Packet Processing. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, Berkeley, CA, USA, 409–423. http://dl.acm.org/citation.cfm?id=2789770.2789799

[70] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz. 2018. Designing Optimal Middlebox Recovery Schemes With Performance Guarantees. *IEEE Journal on Selected Areas in Communications* 36, 10 (Oct 2018), 2373–2383. https://doi.org/10.1109/JSAC.2018.2869956

[71] Georgios P. Katsikas. 2018. *NFV Service Chains at the Speed of the Underlying Commodity Hardware*. Doctoral thesis. KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, Kista, Sweden. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-233629 TRITA-EECS-AVL-2018:50.

[72] Georgios P. Katsikas, Tom Barbette, Dejan Kostić, Rebecca Steinert, and Gerald Q. Maguire, Jr. 2018. Metron: NFV Service Chains at the True Speed of the Underlying Hardware. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, Berkeley, CA, USA, 171–186. http://dl.acm.org/citation.cfm?id=3307441.3307457

[73] Georgios P. Katsikas, Marcel Enguehard, Maciej Kuźniar, Gerald Q. Maguire Jr., and Dejan Kostić. 2016. SNF: synthesizing high performance NFV service chains. *PeerJ Computer Science* 2 (Nov. 2016), e98. https://doi.org/10.7717/peerj-cs.98

[74] Georgios P. Katsikas, Gerald Q. Maguire Jr., and Dejan Kostić. 2017. Profiling and accelerating commodity NFV service chains with SCC. *Journal of Systems and Software* 127C (Feb. 2017), 12–27. https://doi.org/10.1016/j.jss.2017.01.005

[75] Junaid Khalid, Eric Rozner, Wesley Felter, Cong Xu, Karthick Rajamani, Alexandre Ferreira, and Aditya Akella. 2018. Iron: Isolating Network-based CPU in Container Environments. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 313–328. https://www.usenix.org/conference/nsdi18/presentation/khalid

[76] Joongi Kim, Seonggu Huh, Keon Jang, KyoungSoo Park, and Sue Moon. 2012. The Power of Batching in the Click Modular Router. In *Proceedings of the Asia-Pacific Workshop on Systems (APSYS '12)*. ACM, New York, NY, USA, Article 14, 6 pages. https://doi.org/10.1145/2349896.2349910

[77] Joongi Kim, Keon Jang, Keunhong Lee, Sangwook Ma, Junhyun Shim, and Sue Moon. 2015. NBA (Network Balancing Act): A High-performance Packet Processing Framework for Heterogeneous Processors. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)*. ACM, New York, NY, USA, Article 22, 14 pages. https://doi.org/10.1145/2741948.2741969

[78] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. 2000. The Click Modular Router. *ACM Trans. Comput. Syst.* 18, 3 (Aug. 2000), 263–297. https://doi.org/10.1145/354871.354874

[79] M. Kourtis, G. Xilouris, V. Riccobene, M. J. McGrath, G. Petralia, H. Koumaras, G. Gardikis, and F. Liberal. 2015. Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 74–78. https://doi.org/10.1109/NFV-SDN.2015.7387409

[80] Sameer G Kulkarni, Guyue Liu, K. K. Ramakrishnan, Mayutan Arumaithurai, Timothy Wood, and Xiaoming Fu. 2018. REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization Based Services. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT'18)*. Association for Computing Machinery, New York, NY, USA, 41–53. https://doi.org/10.1145/3281411.3281441

[81] Sameer G. Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K. K. Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu. 2017. NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'17)*. Association for Computing Machinery, New York, NY, USA, 71–84. https://doi.org/10.1145/3098822.3098828

[82] Patrick Kutch and Brian Johnson. 2017. SR-IOV for NFV Solutions. Retrieved from https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/sr-iov-nfv-tech-brief.pdf

[83] KVM. 2016. Main Page — KVM,. https://www.linux-kvm.org/index.php?title=Main_Page&oldid=173792 [Online; accessed 23-April-2019].

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

[85] Lenovo, China Mobile, Xilinx, Napatech, Radisys Demo Hardware Acceleration, and Decoupling Technologies. 2018. Industry Leaders Showcase Accelerator-based Cloud-RAN Solution. Retrieved from https://news.lenovo.com/pressroom/press-releases/innovation/industry-leaders-showcase-accelerator-based-cloud-ran-solution.htm

[86] Giuseppe Lettieri, Vincenzo Maffione, and Luigi Rizzo. 2017. A Survey of Fast Packet I/O Technologies for Network Function Virtualization. In *High Performance Computing*, Julian M. Kunkel, Rio Yokota, Michela Taufer, and John Shalf (Eds.). Springer International Publishing, Cham, 579–590. https://link.springer.com/chapter/10.1007/978-3-319-67630-2_40

[87] David Levinthal. 2008-2009. Performance Analysis Guide for Intel i7 Processor and Intel Xeon 5500 Processors. Retrieved from https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

[88] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/2934872.2934897

[89] Miao Li, Qixia Zhang, and Fangming Liu. 2020. Finedge: A Dynamic Cost-efficient Edge Resource Management Platform for NFV Network. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. 1–10.

[90] P. Li, X. Wu, Y. Ran, and Y. Luo. 2017. Designing Virtual Network Functions for 100 GbE Network Using Multicore Processors. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 49–59. https://doi.org/10.1109/ANCS.2017.15

[91] Xiaoyao Li, Xiuxiu Wang, Fangming Liu, and Hong Xu. 2018. DHL: Enabling Flexible Software Network Functions with FPGA Acceleration. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 1–11. https://doi.org/10.1109/ICDCS.2018.00011

[92] Guyue Liu, Yuxin Ren, Mykola Yurchenko, K. K. Ramakrishnan, and Timothy Wood. 2018. Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 504–517. https://doi.org/10.1145/3230543.3230563

[93] IHS Markit. 2016. Network Functions Virtualization Market Worth Over $15 Billion by 2020, Says IHS Markit. Retrieved from https://news.ihsmarkit.com/press-release/technology/network-functions-virtualization-market-worth-over-15-billion-2020-says-ihs

[94] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the Art of Network Function Virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, USA, 459–473. http://dl.acm.org/citation.cfm?id=2616448.2616491

[95] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. https://doi.org/10.1145/1355734.1355746

[96] Zili Meng, Jun Bi, Chen Sun, Shuhe Wang, Minhu Wang, and Hongxin Hu. 2018. PAM: When Overloaded, Push Your Neighbor Aside!. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos (SIGCOMM '18)*. ACM, New York, NY, USA, 63–65. https://doi.org/10.1145/3234200.3234215

[97] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu. 2018. CoCo: Compact and Optimized Consolidation of Modularized Service Function Chains in NFV. In *2018 IEEE International Conference on Communications (ICC)*. 1–7. https://doi.org/10.1109/ICC.2018.8422641

[98] J. Nam, M. Jamshed, B. Choi, D. Han, and K. Park. 2015. Scaling the performance of network intrusion detection with many-core processors. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 191–192. https://doi.org/10.1109/ANCS.2015.7110135

[99] Napatech. 2018. Napatech SmartNIC for Virtualization Solutions. Retrieved from https://www.napatech.com/products/napatech-smartnic-virtualization/

[100] Netronome. 2018. Agilio CX SmartNICs. Retrieved from https://www.netronome.com/products/agilio-cx/

[101] Ethernity Networks. 2018. ENET ACE-NIC 100 Carrier Grade SmartNIC for Virtual Network and Security Acceleration. Retrieved from http://www.ethernitynet.com/wp-content/uploads/2018/07/PB_ACE_NIC_100-July2018.pdf

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

[103] ntop. 1998-2018. PF_RING. Retrieved from https://www.ntop.org/products/packet-capture/pf_ring/

[104] ONAP. 2018. Acceleration Management (BoF). Retrieved from https://wiki.onap.org/pages/viewpage.action?pageId=36963051

[105] OPNFV. 2018. Rocket. Retrieved from https://wiki.opnfv.org/display/PROJ/Rocket

[106] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. E2: A Framework for NFV Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. ACM, New York, NY, USA, 121–136. https://doi.org/10.1145/2815400.2815423

[107] Mike Pall and Luke Gorrie et al. 2012. Snabb: Simple and fast packet networking. Retrieved from https://github.com/snabbco/snabb

[108] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. 2016. NetBricks: Taking the V out of NFV. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 203–216. http://dl.acm.org/citation.cfm?id=3026877.3026894

[109] M. Paolino, N. Nikolaev, J. Fanguede, and D. Raho. 2015. SnabbSwitch user space virtual switch benchmark and performance optimization for NFV. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 86–92. https://doi.org/10.1109/NFV-SDN.2015.7387411

[110] Luis Pedrosa, Rishabh Iyer, Arseniy Zaostrovnykh, Jonas Fietz, and Katerina Argyraki. 2018. Automated Synthesis of Adversarial Workloads for Network Functions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 372–385. https://doi.org/10.1145/3230543.3230573

[111] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2015. Arrakis: The Operating System Is the Control Plane. *ACM Trans. Comput. Syst.* 33, 4, Article 11 (Nov. 2015), 30 pages. https://doi.org/10.1145/2812806

[112] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. 2015. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 117–130. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[113] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. Floem: A Programming System for NIC-accelerated Network Applications. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, Berkeley, CA, USA, 663–679. http://dl.acm.org/citation.cfm?id=3291168.3291217

[114] Paul Quinn, Uri Elzur, and Carlos Pignataro. 2017. Network Service Header (NSH). Retrieved from https://tools.ietf.org/id/draft-ietf-sfc-nsh-17.html

[115] Kaushik Kumar Ram, Alan L. Cox, Mehul Chadha, and Scott Rixner. 2013. Hyper-switch: A Scalable Software Virtual Switching Architecture. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC'13)*. USENIX Association, Berkeley, CA, USA, 13–24. http://dl.acm.org/citation.cfm?id=2535461.2535464

[116] Muhammad Siraj Rathore. 2017. *Performance, Isolation and Service Guarantees in Virtualized Network Functions*. Doctoral thesis. KTH Royal Institute of Technology, School of Information and Communication Technology, Kista, Sweden. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-206830 TRITA-ICT-2017:11.

[117] Wind River. 2014. Wind River Delivers Breakthrough Performance for Accelerated vSwitch Optimized for NFV. Retrieved from https://www.windriver.com/news/press/pr.html?ID=12801
[118] Luigi Rizzo. 2012. Netmap: A Novel Framework for Fast Packet I/O. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*. USENIX Association, Berkeley, CA, USA, 9–9. http://dl.acm.org/citation.cfm?id=2342821.2342830
[119] Luigi Rizzo, Luca Deri, and Alfredo Cardigliano. 2012. 10 Gbit/s line rate packet processing using commodity hardware: Survey and new proposals. https://www.semanticscholar.org/paper/10-Gbit-%2F-s-Line-Rate-Packet-Processing-Using-%3A-and-Rizzo-Deri/faac6e22fde5d600034e2e99cd5673c14728fbbd
[120] Luigi Rizzo and Giuseppe Lettieri. 2012. VALE, a Switched Ethernet for Virtual Machines. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*. ACM, New York, NY, USA, 61–72. https://doi.org/10.1145/2413176.2413185
[121] R. V. Rosa, C. E. Rothenberg, and R. Szabo. 2015. VBaaS: VNF Benchmark-as-a-Service. In *2015 Fourth European Workshop on Software Defined Networks*. 79–84. https://doi.org/10.1109/EWSDN.2015.65
[122] Raphael Vicente Rosa, Christian Esteve Rothenberg, and Robert Szabo. 2016. VNF Benchmark-as-a-Service. Retrieved from https://tools.ietf.org/html/draft-rorosz-nfvrg-vbaas-00
[123] Rusty Russell. 2008. Virtio: Towards a De-facto Standard for Virtual I/O Devices. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 95–103. https://doi.org/10.1145/1400097.1400108
[124] G. Sabin and M. Rashti. 2015. Security offload using the SmartNIC, A programmable 10 Gbps ethernet NIC. In *2015 National Aerospace and Electronics Conference (NAECON)*. 273–276. https://doi.org/10.1109/NAECON.2015.7443082
[125] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. 2012. Design and Implementation of a Consolidated Middlebox Architecture. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX, San Jose, CA, 323–336. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/sekar
[126] N. Shah, K. Ravindran, W. Plishker, and K. Keutzer. 2004. NP-Click: A Productive Software Development Approach for Network Processors. *IEEE Micro* 24 (09 2004), 45–54. https://doi.org/10.1109/MM.2004.53
[127] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, and et al. 2015. Rollback-Recovery for Middleboxes. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*. Association for Computing Machinery, New York, NY, USA, 227–240. https://doi.org/10.1145/2785956.2787501
[128] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. 2012. Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service. *SIGCOMM Comput. Commun. Rev.* 42, 4 (Aug. 2012), 13–24. https://doi.org/10.1145/2377677.2377680
[129] Solarflare. 2018. OpenOnload. Retrieved from http://www.openonload.org/
[130] Haoyu Song. 2013. Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*. ACM, New York, NY, USA, 127–132. https://doi.org/10.1145/2491185.2491190
[131] Nik Sultana, Salvator Galea, David Greaves, Marcin Wójcik, Jonny Shipton, Richard G. Clegg, Luo Mai, Pietro Bressana, Robert Soulé, Richard Mortier, Paolo Costa, Peter Pietzuch, Jon Crowcroft, Andrew W. Moore, and Noa Zilberman. 2017. Emu: Rapid Prototyping of Networking Services. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17)*. USENIX Association, Berkeley, CA, USA, 459–471. http://dl.acm.org/citation.cfm?id=3154690.3154734
[132] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. 2017. NFP: Enabling Network Function Parallelism in NFV. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, New York, NY, USA, 43–56. https://doi.org/10.1145/3098822.3098826
[133] W. Sun and R. Ricci. 2013. Fast and flexible: Parallel packet processing with GPUs and click. In *Proceedings of Architectures for Networking and Communications Systems*. 25–35. https://doi.org/10.1109/ANCS.2013.6665173
[134] Netcope Technologies. 2018. MEET NFB-200G2QL, NEW 200G PROGRAMMABLE SMART NIC. Retrieved from https://www.netcope.com/en/company/press-center/press-releases/meet-nfb-200g2ql,-new-200g-programmable-smart-nic
[135] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P. Markatos, and Sotiris Ioannidis. 2008. Gnort: High Performance Network Intrusion Detection Using Graphics Processors. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID '08)*. Springer-Verlag, Berlin, Heidelberg, 116–134. https://doi.org/10.1007/978-3-540-87403-4_7
[136] Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, and Sotiris Ioannidis. 2014. GASPP: A GPU-Accelerated Stateful Packet Processing Framework. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association, Philadelphia, PA, 321–332. https://www.usenix.org/conference/atc14/technical-sessions/presentation/vasiliadis
[137] Giorgos Vasiliadis, Michalis Polychronakis, and Sotiris Ioannidis. 2011. MIDeA: A Multi-parallel Intrusion Detection Architecture. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. ACM, New York, NY, USA, 297–308. https://doi.org/10.1145/2046707.2046741
[138] IO Visor. 2016. eXpress Data Path (XDP). Retrieved from https://www.iovisor.org/technology/xdp
[139] Tao Wang, Hong Xu, and Fei Liu. 2017. Multi-resource Load Balancing for Virtual Network Functions. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 1322–1332. https://doi.org/10.1109/ICDCS.

2017.233

[140] Yuefeng Wang, Flavio Esposito, Ibrahim Matta, and John Day. 2013. Recursive InterNetworking Architecture (RINA) Boston University prototype programming manual (version 1.0). https://open.bu.edu/handle/2144/11421

[141] Shinae Woo, Justine Sherry, Sangjin Han, Sue Moon, Sylvia Ratnasamy, and Scott Shenker. 2018. Elastic Scaling of Stateful Network Functions. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 299–312. https://www.usenix.org/conference/nsdi18/presentation/woo

[142] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. 2019. NFVdeep: Adaptive Online Service Function Chain Deployment with Deep Reinforcement Learning. In *Proceedings of the International Symposium on Quality of Service (IWQoS '19)*. Association for Computing Machinery, New York, NY, USA, Article Article 21, 10 pages. https://doi.org/10.1145/3326285.3329056

[143] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. 2017. Towards load-balanced VNF assignment in geo-distributed NFV Infrastructure. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS.2017.7969166

[144] Cong Xu, Sahan Gamage, Hui Lu, Ramana Kompella, and Dongyan Xu. 2013. vTurbo: Accelerating Virtual Machine I/O Processing Using Designated Turbo-Sliced Core. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (USENIX ATC'13)*. USENIX Association, USA, 243–254. https://dl.acm.org/doi/10.5555/2535461.2535491

[145] Zhifeng Xu, Fangming Liu, Tao Wang, and Hong Xu. 2016. Demystifying the energy efficiency of Network Function Virtualization. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS.2016.7590429

[146] Xiaodong Yi, Jingpu Duan, and Chuan Wu. 2017. GPUNFV: A GPU-Accelerated NFV System. In *Proceedings of the First Asia-Pacific Workshop on Networking (APNet'17)*. ACM, New York, NY, USA, 85–91. https://doi.org/10.1145/3106989.3106990

[147] Pamela Zave, Ronaldo A. Ferreira, Xuan Kelvin Zou, Masaharu Morimoto, and Jennifer Rexford. 2017. Dynamic Service Chaining with Dysco. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, New York, NY, USA, 57–70. https://doi.org/10.1145/3098822.3098827

[148] Chaobing Zeng, Fangming Liu, Shutong Chen, Weixiang Jiang, and Miao Li. 2018. Demystifying the Performance Interference of Co-Located Virtual Network Functions. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 765–773. https://doi.org/10.1109/INFOCOM.2018.8486246

[149] Kai Zhang, Bingsheng He, Jiayu Hu, Zeke Wang, Bei Hua, Jiayi Meng, and Lishan Yang. 2018. G-NET: Effective GPU Sharing in NFV Systems. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 187–200. https://www.usenix.org/conference/nsdi18/presentation/zhang-kai

[150] Qixia Zhang, Fangming Liu, and Chaobing Zeng. 2019. Adaptive Interference-Aware VNF Placement for Service-Customized 5G Network Slices. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2449–2457. https://doi.org/10.1109/INFOCOM.2019.8737660

[151] Qixia Zhang, Yikai Xiao, Fangming Liu, John C S Lui, Jian Guo, and Tao Wang. 2017. Joint Optimization of Chain Placement and Request Scheduling for Network Function Virtualization. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 731–741. https://doi.org/10.1109/ICDCS.2017.232

[152] Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K.K. Ramakrishnan, and Timothy Wood. 2016. Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 3–17. https://doi.org/10.1145/2999572.2999602

[153] W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, and T. Wood. 2016. Performance management challenges for virtual network functions. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. 20–23. https://doi.org/10.1109/NETSOFT.2016.7502435

[154] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K.K. Ramakrishnan, and Timothy Wood. 2016. OpenNetVM: A Platform for High Performance Network Service Chains. In *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization (HotMIddlebox '16)*. ACM, New York, NY, USA, 26–31. https://doi.org/2940147.2940155

[155] Wei Zhang, Abhigyan Sharma, Kaustubh Joshi, and Timothy Wood. 2018. Hardware-assisted Isolation in a Multi-tenant Function-based Dataplane. In *Proceedings of the Symposium on SDN Research (SOSR '18)*. ACM, New York, NY, USA, Article 15, 7 pages. https://doi.org/10.1145/3185467.3185493

[156] Yang Zhang, Bilal Anwer, Vijay Gopalakrishnan, Bo Han, Joshua Reich, Aman Shaikh, and Zhi-Li Zhang. 2017. ParaBox: Exploiting Parallelism for Virtual Network Functions in Service Chaining. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. ACM, New York, NY, USA, 143–149. https://doi.org/10.1145/3050220.3050236

[157] Dong Zhou, Bin Fan, Hyeontaek Lim, Michael Kaminsky, and David G. Andersen. 2013. Scalable, High Performance Ethernet Forwarding with CuckooSwitch. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*. ACM, New York, NY, USA, 97–108. https://doi.org/10.1145/2535372.2535379