electronics

Article

A Survey of NFV Network Acceleration from ETSI Perspective

Yong-Xuan Huang [1,*] and Jerry Chou [2,*]

[1] Institute of Information System and Applications, National Tsing Hua University, Hsinchu 300, Taiwan
[2] Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan
* Correspondence: yxhuang@lsalab.cs.nthu.edu.tw (Y.-X.H.); jchou@lsalab.cs.nthu.edu.tw (J.C.)

Abstract: Network function virtualization (NFV) enables network operators to save costs and flexibility by replacing dedicated hardware with software network functions running on commodity servers. There is a high need for network acceleration to achieve performance comparable to hardware, which is vital for the implementation of NFV. The necessity of NFV acceleration stems from the lengthy packet delivery path following virtualization and the unavailability of generic operating system designs to serve network-specific scenarios. Therefore, the software approach alters the operating system's processing architecture through Kernel Bypass or offload packet processing to hardware. A typical classification scheme divides it into two main categories based on technology with software and hardware. Only these two categories can be utilized to rapidly and easily establish a classification system. However, it is difficult to suggest the specifics and peculiarities of any acceleration approach during real-world operation. For a more comprehensive classification of NFV acceleration, we refer to the ETSI NFV architectural framework in this research. As the framework clearly illustrates, the technical infrastructure layer of NFV and the corresponding management roles provides a comprehensive and intuitive view of the differences between these acceleration technologies, solutions, and initiatives. Additionally, we conducted an analysis to identify opportunities for improvement in existing solutions and propose new research programs. We expect that NFV will increasingly rely on cloud services in the future. Since cloud services do not offer a choice of hardware, our acceleration method will be primarily software-based.

Keywords: network function virtualization; network acceleration; ETSI NFV framework

check for updates

Citation: Huang, Y.-X.; Chou, J. A Survey of NFV Network Acceleration from ETSI Perspective. *Electronics* **2022**, *11*, 1457. https://doi.org/10.3390/electronics11091457

Academic Editors: Carlos Senna and Arnaldo Silva Rodrigues de Oliveira

Received: 31 March 2022
Accepted: 29 April 2022
Published: 2 May 2022

1. Introduction

Network functions virtualization (NFV) enables enhanced flexibility and cost-saving by substituting software deployed on commodity servers for hardware network facilities. NFV uses virtualization technology to reconstruct network functions and deploy on standard infrastructure hardware, which allows for rapid scalability and meets carrier-grade operational specifications. Furthermore, in the place of designated hardware, NFV utilizes software that can be dynamically deployed, scaled, and migrated as an intermediary, thus providing a flexible and programmable infrastructure.

However, physical network devices have limited scalability because they only run on proprietary hardware appliances. Replacing application-specific integrated circuit (ASIC) hardware can be seriously detrimental to performance, affecting primary performance such as throughput and latency, which in turn impacts the overall end-to-end application performance [1]. Performance issues are mostly related to the packet delivery path. When moving packets from the network interface controller (NIC) to the driver, the packets need to pass through a ring buffer before moving to the poll queue to be read by the process, hence the latency in delivery. Moreover, this process requires multiple I/O operations, resulting in reduced performance. The latency in resolution is due to the CPU's time-division multiple access (TDMA) architecture, which requires the CPU to perform content-switching non-stop when processing request flows of different lengths, which causes overheads in processing.

For accelerating the network performance on NFV, recent research has focused on accelerating virtual network functions (VNF) and addressing performance issues at different levels. For example, studies [2–5] include improving software architecture, optimizing the entire data packet transmission process from the NIC to the network stack, or offloading part of the packet-processing process to programmable hardware. Typically, these solutions fall into two categories: software-based solutions such as kernel bypass, zero-copy, or hardware-based offload techniques [6–8]. Because both technologies will be used in tandem at times, it is difficult to effectively convey the distinctions between each acceleration approach. A typical example is the service chain, which was previously classed as a software acceleration technique for using algorithms to improve the path taken by user requests through websites. Therefore, we follow the NFV architectural framework from the European Telecommunication Standards Institute (*ETSI*) in this research to accommodate a broader degree of acceleration research. The ETSI NFV framework allows re-identifying whether the acceleration method is at the infrastructure or management layer. In the infrastructure layer, the acceleration optimizes the interaction of various virtual network functions (VNFs) within an element management system (EMS). However, the management layer's virtualized infrastructure manager (VIM) performs path optimization between various EMSs. The example indicates that the acceleration technology should apply to one or several EMSs, which were previously unable to detect only on the classification within hardware and software. Additionally, we also cover industrial products and solutions to demonstrate the viability of advancing the adoption of NFV as a realistic management framework.

The primary contribution of this research was to employ the ETSI Framework as the classification's major axis. The framework's operability and practicality enable more precise identification of the characteristics and applications of each accelerated research strategy. Simultaneously, we observe that cloud-based infrastructure has progressively become the major deployment environment for NFV. This implies that, while hardware acceleration approaches are successful, they may not be appropriate for cloud situations with restricted hardware alternatives. As a result, this paper's study will concentrate on the software method, which is also one of our innovations.

The rest of the paper is structured as follows. Section 2 describes the ETSI NFV framework and how the acceleration methods will be mapped to the management framework. Sections 3 and 4 will break down the acceleration technologies into subcomponents based on the ETSI NFV framework's two infrastructure and management categories. Furthermore, Section 5 extends the examination of future acceleration trends and directions and Section 6 concludes the paper.

2. Taxonomy Methods with ETSI Management Framework

With the growth of NFV technologies, network operators urgently want easy access to these solutions. However, most suppliers vigorously advertise their NFV solutions, emphasizing their own business and competitive advantages. Due to this, there is no single management standard for these solutions and no unified interface for the higher layer network element to use, which network operators dislike. ETSI is a non-profit, independent standards body focused on telecommunications in Europe. The members are equipment makers and Internet service providers. ETSI develops global standards for information and communication technologies (ICTs), such as fixed, mobile, radio, convergent, broadcast, and Internet technology. ETSI's NFV architectural framework is ETSI's recommended implementation of NFV governance, which shows in Figure 1. We take the framework as the primary cornerstone of our taxonomy investigation due to its depth of coverage and realizability.

The left part of Figure 1 illustrates how virtualization techniques abstract hardware into components capable of performing a single network function. From the bottom–up, physical resources are converted into virtual resources via virtualization techniques, and then virtual resources are logically divided into the virtualization network function (VNF).

Because network functions are regarded as elements, they will be provided in the element management system (EMS). When the network components are available, they connect to the service provider's business and operation support system (OSS/BSS) to provide services. It is worth noting that although container technology abstracts the hardware via a technique other than virtual simulation, the concept is still a form of virtualization. The right side of the Figure corresponds to the management positions associated with each of these processes. To begin, the virtualization of hardware requires a coordinator to translate resource requirements into matching hardware, which is the responsibility of the virtualization infrastructure manager (VIM). Then, the EMS responds to managing the lifetime of VNF while providing services, and requirements are passed onto the VNF manager for processing. Finally, once the service is available for OSS/BSS, the NFV orchestrator will coordinate the EMS assistance required.

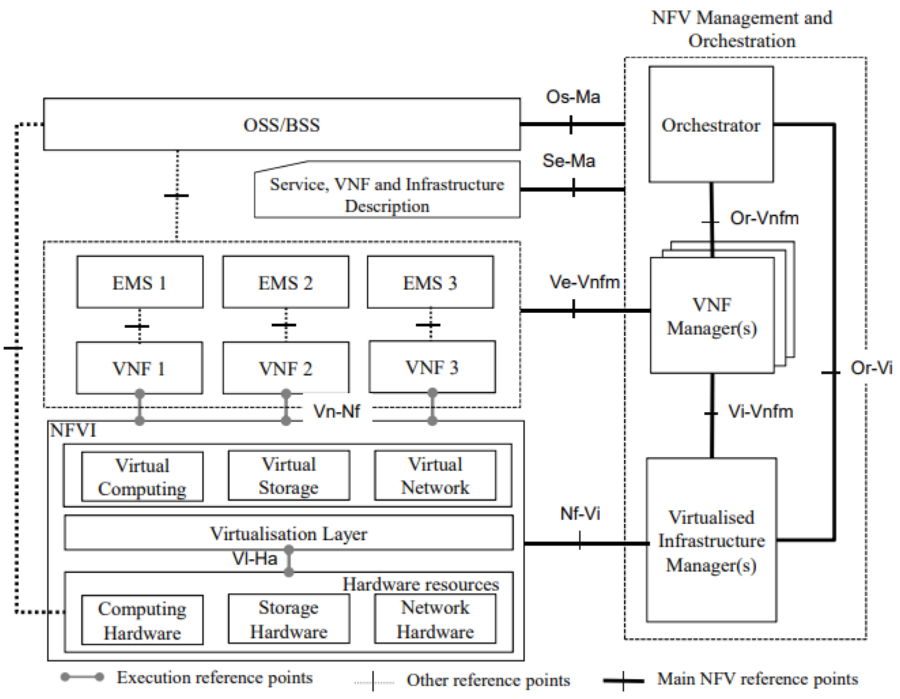

Figure 1. NFV reference architectural framework.

2.1. ETSI NFV Infrastructure Architecture

We examine the classification of acceleration concerning infrastructure components in terms of the work they perform. The infrastructure consists of EMS, VNF, and NFVI. EMS is a collection of VNFs that emulate the various network hardware functions. VNFs can run on one or multiple virtual machines (VMs). Generally, each EMS controls one or multiple corresponding VNFs at once. NFVI is the most basic infrastructure in the NFV architecture and mainly consists of three function blocks: virtualized resources, virtualization layer, and hardware resources. Hardware resources include computing power, storage, and networking (for example, routers, wired or wireless connections). The virtualization layer between the virtualized and hardware integrates software with hardware. Once hardware resources are virtualized, they become virtual resources that VNF can dynamically request.

We classified the components associated with the acceleration studies as Figure 2. The VNF is the infrastructure component between the network service and the entity and provides the most significant acceleration benefits. Since VNFs are derived from the same design as PNFs, the PNF is usually accelerated through operating system tuning, operating system network module tuning, or hardware acceleration. VNF adopts the same strategy, except the acceleration from hardware-provided offloads is replaced by virtual hardware via the hypervisor, such as a virtual switch module in the hypervisor that sends packets directly to the virtual machine's NIC.

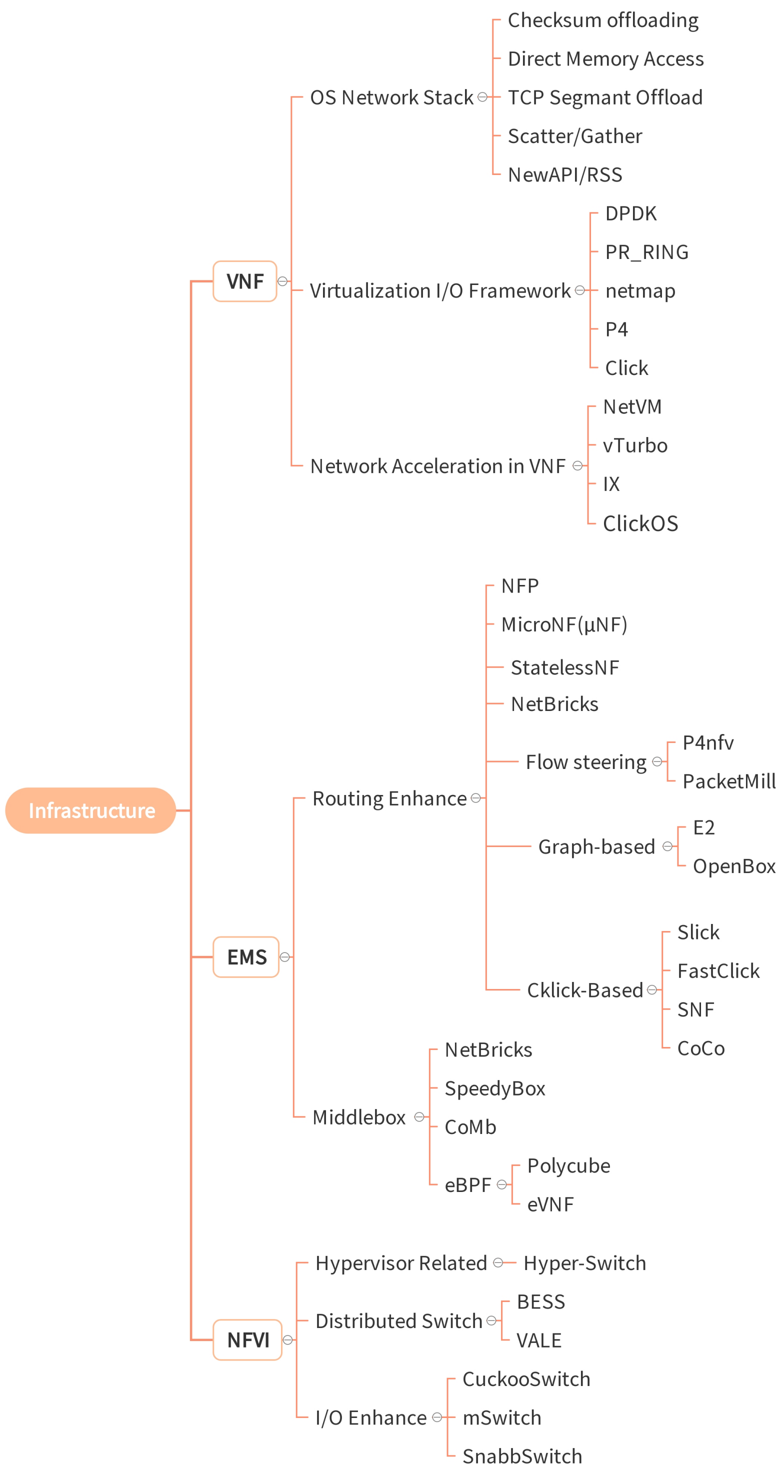

Figure 2. The accelerating solutions based on NFV infrastructure.

As the core of the whole NFV framework, VNFs are the most studied for acceleration since they are the central object of the overall NFV. It is because acceleration on VNF delivers the most value to the NFV as a whole. VNF can be thought of as network functions implemented in a virtual machine or container. We can abstract that the packet's process flow components are the virtualized hardware resources, the virtual machine's operating system, and the network stack. Therefore, in VNFs, we classify the related functions into network acceleration in VNFs, virtualization I/O framework, and OS network stack. The network acceleration in VNFs is a technique for increasing network performance by utilizing virtualized hardware resources. The virtualization I/O framework is a classification of how to tune the operating system to support the network, whereas the OS network stack studies how to tune the operating system's network modules.

The acceleration mechanisms of EMS are dependent on their architecture, which is typically composed of one or more VNFs. EMS must manage VNF rapidly and efficiently in this deployment architecture. The majority of service chain acceleration algorithms can resolve crosstalk between several VNFs. Remember that this concept will be presented again in the VNFM. The contrast is that in the VNFM, wherein the optimization logic for the service chain is resolved after external processing. The EMS is optimized internally via automated optimization, so accelerated research in EMS is concerned with optimizing the service chain. The optimization approach can be classified into conventional routing enhance approaches, EMS with a middlebox for control optimization, and the novel kernel technique, eBPF. Routing technology facilitates seamless communication between the VNF's several subnetwork components. Middlebox employs a distinct control plane and data plane architecture to create a micro-SDN architecture within a single EMS. The eBPF is a novel approach for accelerating the development of kernel modules, which can significantly reduce the time required to meet EMS criteria.

NFVI can be directly interpreted as a hypervisor because NFVI is in charge of packet delivery to the virtual machine. The common acceleration occurs in a virtual switch. We are going to organize the studies to improve the intelligence of the software Switch, including enhancing the packet delivery path, directly integrating with the hypervisor, and enhancing the Switch-related functions. The acceleration associated with the infrastructure is detailed in the following section.

2.2. ETSI NFV Management Architecture

The NFV management and orchestration function blocks are critical for the overall control and coordination of the NFV technology architecture. It is primarily composed of three components: the NFV orchestrator (NFVO), the VNF manager (VNFM), and the virtualized infrastructure manager (VIM). We organize the NFV acceleration to correspond to the ETSI NFV framework as Figure 3.

The role of VIM is to communicate and coordinate between NFVI as at the bottom components. A typical strategy is to accelerate network capability through cross-node work. SDN gives an excellent example of separating the control and data planes to achieve flex-scale network capacity in response to network loads dynamically. VIM's administration designs for managing virtual switches distributed across NFVIs include packet flow acceleration methods. Traffic steering acceleration is a noteworthy strategy [8]. By applying efficient steering models in the virtual switch, it is possible to reduce inter-core communications and deal with more complex scenarios in traffic steering. Furthermore, given the statefulness of a large number of NFs, this acceleration is critical for attaining high-performance service chains.

When software and hardware resources are required, the NFV MANO coordinates, validates, and authorizes the request for resources and manages the lifecycle of EMS, which includes operations such as the instantiation, scaling, updating, querying, and termination of VNFs. In addition, the NFV MANO is responsible for network policy management, performance measurements, the collection and transfer of relevant events, and the allocation of infrastructure-related resources, for instance, adding resources to virtual machines, improving energy efficiency, and reclaiming resources. The development of multi-vendor and multi-service NFV systems entails that relevant services and overall operations will require large amounts of data processing and operations. Therefore, NFV MANO must identify and reference relevant data when managing process coordination.

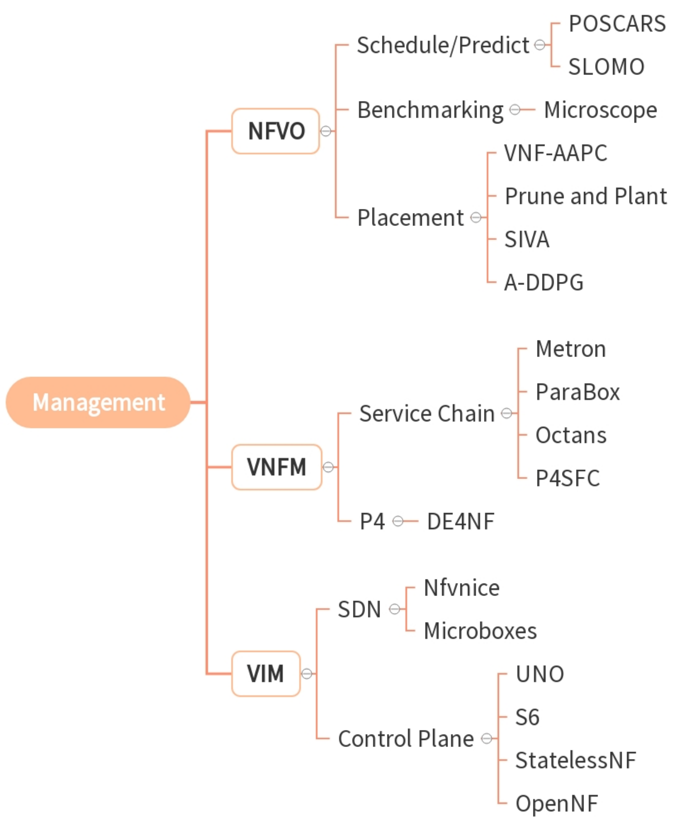

Figure 3. The accelerating solutions based on NFV management framework.

The middle tier, VNFM, as the name implies, manages the life cycle of the VNF. It includes VNF deployment templates, VNF service cascade forwarding diagrams, NFVI information models, and various service-related information. The standard VNFM management tools nowadays are relatively simple in function. Therefore, many studies have tried to incorporate intelligent processing into their designs, such as placement problems. We will also introduce more studies in subsequent sections.

3. Accelerations in NFV Infrastructure

3.1. Virtualized Network Function (VNF)

Understanding the packet delivery process within the OS network stack helps with a more fundamental understanding of network stack issues. In Appendix A, we succinctly organize the OS packet processing procedure. The general I/O architecture of the operating system is the root cause of network latency. Therefore, adapting a general-purpose architecture to meet the needs of a network with particular features is a common optimization strategy. As mentioned in the previous section, we subdivided the acceleration methods of VNF into OS network stack, virtualization I/O framework, and network acceleration in VNF. We discuss the related solutions in each of these categories in the rest of the section.

3.1.1. Os Network Stack

Direct Memory Access

Direct memory access (DMA) is mainly adequate for accelerating the packet processing of a single machine. However, NFV, through hardware abstraction, requires two independent hosts to exchange packet processing between them. Therefore, the packet exchange between two OSs has to go through the OS network stack, which requires substantial server resources and bus bandwidth. Furthermore, the data are copied and moved back and forth between memory, processor cache, and network controller cache, which burdens the CPU and the server's memory.

Checksum Offloading

Network transmission protocols such as IP, TCP, and UDP need checksum to verify packets. The CPU does checksum calculation (sending) and verification (receiving), which does affect the CPU, as producing a checksum involves every byte of data in the calculation. For a 100 Gbps network, the CPU needs to calculate approximately 12 GB of data. In order to reduce this burden on the CPU, modern NICs support checksum calculation and verification. Kernel packets can skip the checksum by waiting until the NIC receives the packet, computing it according to the network protocol rules, and then filling in the checksum in the relevant place.

New API/Receive Side Scaling

When the DMA is loaded into the packet, the NIC issues an interrupt request (IRQ) to the CPU, allowing the CPU to continue. The CPU must process the interrupt handler for each IRQ that is triggered. Suppose that the NIC generates an IRQ each time it gets a packet. In that case, the CPU will spend considerable time executing the interrupt handler and will be able to recover only one packet from the ring buffer after processing. Even though the interruption interval is brief, it significantly impairs performance. Newer kernels employ a new API (NAPI) method to identify whether to use a pure interrupt or a poll query based on the number of packets to be handled to combine IRQs and reduce call count. A similar approach is called receive side scaling (RSS), which effectively reduces interrupts with a multi-core processor setup. NICs possessing RSS compatibility have multiple reception queues. Thus, NICs can use different queues to receive network flows. These queues are allocated to different CPUs to divide the load and improve network transmission efficiency.

Scatter/Gather

Scatter/gather is a commonly used acceleration method known as vector addressing, which mainly accelerates the packet sent. In simple terms, this means that a reader can read data from multiple separated memory addresses during data transfer, rather than continuously reading data from one buffer. For example, a kernel retains the original data after receiving it from the application. Then, it computes the protocol header of each layer in another memory address and notifies the NIC driver to copy the data from these two memory addresses to reduce unnecessary copy processes.

TCP Segmentation Offload

TCP segmentation offload (TSO) is a method for transferring data over a network that is similar to TCP. It lets apps deliver data of any length to TCP. TCP is a transport layer protocol that does not deliver accurate user data to the lower protocols. Instead, data can be segmented and sent in segments to ensure reliability, efficiency, and optimal transmission. TCP (L2) and IP (L3) data segmentation is data fragmentation. TCP breaks massive data into smaller portions based on the maximum segment size (MSS) before sending a packet to the IP layer. Due to the MTU limitation, the IP protocol separates the data from the higher layer into numerous portions.

3.1.2. Virtualization I/O Framework

Along with OS network stack advancements, network acceleration via virtualization is an active research field. However, because virtualization lacks the operating system's priority execution, it is easy to incur inefficient I/O due to the long path to convey data from the operating system to the hypervisor. Because of this, research in virtualization technology, I/O efficiency, and processing has historically been a critical field of study. The following sections outline the data and network I/O difficulties and the accompanying acceleration techniques.

DPDK

Numerous endeavors alleviate the time of context switching using techniques with kernel bypasses. DPDK [9] is a collection of data plane libraries and network interface controller (NIC) drivers for high-speed packet processing. Using DPDK libraries and application program interfaces (APIs) in userspace, the program can constantly poll instead of managing packet arrivals via interrupts. For minimizing locking overhead, DPDK's lock-free rings are based on the Linux kernel's lock-free ring buffers, which support both one-to-one and many-to-many producer/consumer models. Thread affinity refers to DPDK associating threads with logical cores to minimize the context switching and boost CPU cache hit rate. DPDK employs per-core memory in NUMA to ensure cache consistency and uses 2 MB and 1 GB big pages to minimize the possibility of TLB misses. Finally, DPDK is hardware-independent, as it provides a programming framework that works with any processor. Moreover, DPDK can achieve tenfold the performance of packet processing, with a throughput of more than 80 Mpps on a single processor.

PF_RING (DNA)

The primary goal of PF_RING is to reduce packet copy time during transmission. PF_RING polls the NIC for packets and stores them in ring buffers to accomplish this. Userspace applications then read packets directly from the rings using the Linux New API (NAPI). This technique, however, requires two polling times for NAPI and the application, which uses additional CPU cycles. To reduce the CPU consumption for polling packets, PF_RING DNA copies packets from the NIC to ring buffers by NIC NPU instead of NAPI. As a result, it performs better, but the disadvantage is that only one program may access the ring at a time, and apps must swap packets to spread them.

Netmap

Netmap [10] significantly reduces or eliminates overheads associated with per-packet memory allocations, system calls, and packet copying by adding three optimization techniques: buffer preallocation, big batch processing, and shared buffers. These are used to accelerate packet transmission across the NIC to userspace applications. Notably, netmap offers two modes that enable users to specify whether packets should be routed through the host stack [11]. In contrast, the netmap mode enables packets to be delivered directly to an application using the netmap API. Netmap's implementation requires only minor modifications, does not require any specialized hardware, and can easily hit the line rate on a 10G NIC with a 64B frame size [10].

P4

P4 [12] is a high-level programming language designed to develop protocol-independent packet processors that act as a general-purpose, flexible interface between switches and controllers for matching header fields and parsing packets. In addition, P4 provides a straightforward API for configuring the switch's physical implementation. In summary, P4 can accomplish three objectives:

- Protocol independence: switches should be capable of handling packets of varying formats.
- Reconfigurability: the controller can parse and process packets with programming.
- Target independence: programmers can describe the functions that process packets without knowing the hardware implementation details.

In comparison to more widely used programming languages (such as C or Python), P4 is a domain-specific language that provides high-level abstraction for network programming and improves network data forwarding via a collection of carefully built features.

Click

Click is a widely used software router that supports flexible and modular configuration. A click router is composed of numerous parts, each of which processes packets in order to carry out a certain router function [13]. Due to Click's modular nature, it is easy to extend it. Apart from routing, Click can be used for rapid prototyping and new protocols. Researchers have made significant attempts to enhance the performance of Click with the development of RouteBricks [14], DoubleClick [15], Click-on-osv [16], and FastClick [17]

3.1.3. Network Acceleration in VNFs

We give a comparison at Table 1. To improve the efficiency of packet delivery, VNFs use the same strategies as PNFs, such as zero-copy, NUMA, and other architectures. Drivers such as virtIO and pnet provide a standard equipment model. To good effect, they use I/O acceleration mechanisms commonly found on PNFs, such as zero-copy or NUMA. ClickOS [18] and NetVM [19] are all examples of research that optimizes packet transfer between the NIC and the VM, as well as between VMs. Due to the length of time required for development, simulation technology used to produce VNF technology is still the preferred alternative.

IX [20] relies on multi-queue NICs to securely hash incoming packets, and operating a large number of NFs in the data plane eliminates synchronization overhead. Offloading is also well-known for significantly reducing CPU utilization in VNFs. vTurbo offloads tasks to a turbo core, a time-sliced slice segregated from the CPU core. Offloading workloads to a dedicated turbo core decreases the latency associated with VM core access. Therefore, vTurbo significantly increases the network throughput at the application level.

Table 1. Network acceleration in VNF.

Name	Strategies	Effectiveness	Advantage	Disadvantage
IX	Separates control plane and data plane	On a 10GbE experiment using short messages, IX outperforms Linux and mTCP by up to 10× and 1.9× respectively for throughput	High I/O performance, while maintaining the key advantage of strong protection offered by existing kernels	Only support hardware-based hypervisor
vTurbo	Implement CPU scheduler to enhance vCPU	Increase of 63–200% for TCP, 300% for UDP throughput and 75% to 82% for disk write and 26% for disk read	The improvement supports all I/O behavior	The implementation needs evolved hypervisor
NetVM	Using NUMA-aware and DPDK design to improve network performance	Inter-VM communication using NetVM can achieve a line-rate speed	Only requires a simple descriptor to be copied via shared memory, which then gives the VM direct access to packet data stored in huge pages	Cannot share CPU loading in multicore platform
ClickOS	A Xen-based software platform optimized for middlebox processing	Linux throughput increases from 6.46 Gb/s to 9.68 Gb/s for 1500B packets and from 0.42 Gb/s to 5.73 Gb/s for minimum-sized packets	Resource using optimized with Small (5MB), boot quickly (approximately 30 milliseconds) and add little delay (45 microseconds)	Only supports Xen-based hypervisor

3.2. Element Management System (EMS)

The basic unit of a network service is the element management system (EMS). Compared to the accelerated notion of VNFs, it can perform network processing at a higher level, which means that several VNFs can operate in the same EMS. Linking VNFs to analytical services is the most straightforward approach to deploying an EMS. However, EMS may need to pass user network flow via multiple VNFs. As previously stated in our classification scheme, EMS places a premium on interoperability and control amongst VNFs. The Middlebox provides a more detailed view of network traffic than the VNF, which will behave differently in practice. In more detail, route enhancing refers to optimizing the flow or forwarding components. Thus, by evaluating the service chain of user requests, the EMS may determine the flow of VNF. For a more clear comparison, we organized the solutions as Table 2.

Table 2. The acceleration solutions in EMS by routing enhance.

Name	Strategies	Effectiveness	Advantage	Disadvantage
NFP	Allows parallelized NF to boost the performance of service chains	Decreases latency with 35.9%	Enables network function parallelism	Lack of policy specification scheme to represent more complex NF composition rules
MicroNF	A disaggregated packet processing architecture facilitating the deployment of VNFs and SFCs	Reaches a throughput of 2.08 Mpps or 3.67 Gbps	Supports existing solutions including batched I/O and zero-copy I/O	Does not support complex VNFs
Slick	Designs and implements various VNFs based on Click	10–15% of optimal placement and outperforms random placement for varying number of flow sizes	Proposes a secure middleware framework	Does not support multi application
StatelessNF	Using DPDK and OpenFlow-based network	Reach a throughput of 10 Gbit/sec, with an added latency overhead of between 100 µs and 500 µs	Supports failover which does not disrupt ongoing traffic	Only supports limited scenario
NetBricks	Enhances I/O efficiency by zero-copying	64B packets copying can result in a performance drop of up to 3 times	Provides the same memory isolation as containers and VMs	The supporting scenario is simple
P4nfv	P4 for programming protocol-independent packet processors	On average, 108.43 packets are lost during the migration of the NAT which corresponds to a service disruption of 0.217 s	Ensures the liveness of functions and acceptable performance degradation when functions are migrated	Only supports P4 based
PacketMill	Efficiently manages packet metadata	Increases throughput (up to 36.4 Gbps–70%)	Reduces latency up to 101 us	Needs hardware support
E2	Allows developers to rely on external framework-based mechanisms for common tasks	Offers a 25–41% reduction in CPU and a 1.5–4.5× improvement in overall system throughput	A single controller handles both the management and interconnection of NFs	Not a standard NF management framework
OpenBox	Decoupling the control plane of a NF	Increase from 86 90% performance and reduce 35 50% overhead	Effectively decouples the control plane of NFs from their data plane	Lack of security support
FastClick	Integrating a faster Click with both Netmap and DPDK	Exhibits up-to about 2.3× speed-up	Boasts improved abstractions for packet processing	Does not support middlebox
SNF	Synthesizes NF service chains by removing the redundant operations (i.e., packet and I/O operations)	Achieves line-rate 40 Gbps throughput (up to 8.5× greater)	Eliminating redundant I/O and repeated elements	Only applicable to fairly limited service chains
CoCo	Provides a performance-aware approach for deploying modularized SFCs	Reduces the total DB by 2.46× compared to random strategy and by 1.64× compared to greedy strategy	Consolidate and provides a performance-aware placement algorithm to place MSFCs	Only scales out the element that is overloaded and to reduce the scaling overhead

3.2.1. Routing Enhance

Routing enhance provides many advantages, including the reuse of applications and software and geographic dispersion. However, developing a new EMS continues to be a painful process that requires developers to continually rediscover and reapply the same set of optimizations. At the same time, existing approaches for providing isolation across NFVs (through VMs or containers) incur significant performance costs. NetBricks [21] takes inspiration from data analytics frameworks when developing NFVs and creating a modest collection of customizable network processing pieces. Likewise, NetBricks adopts type checking and safe runtimes to offer isolation in software rather than relying on hardware isolation. Simultaneously, the processing of this flow is abstracted as an np-problem, laying the groundwork for future study in this area.

The study in Ref. [22] examined how to identify the next user flow item at each network stage. These pathways in the table will be replaced when they discover a more advantageous site in the process beyond. The table where these paths are recorded is called the matching action table. Since the table can track the operational status of each NF record in real-time, adjusting the order of NFVs and eliminating repetitive tasks can resolve different NFVs, resulting in reduced operational delays. PacketMill [23] demonstrates the use of specialized DPDK buffers and optimized code to minimize unnecessary memory access. Therefore, user stream metadata are more efficiently managed due to better buffer-local performance. This adds to implementing network services at 100 Gbps and is more significant in software on commercial hardware. As previously indicated, there is forwarding component optimization in addition to flow optimization, in which the forwarding components influence the flow design to some extent. The NFP [24] of the flow process duplicates data elimination processing. After identifying the critical performance factors, instruction-level parallelism can process the user flow in parallel. Using the basic NFP infrastructure to consolidate packets for processing avoids consuming additional network bandwidth resources. It also supports NFP with a zero-copy of packet delivery mechanisms.

Click is a novel way to provide configurable routing. Because of its scalability, many studies were conducted around Click. Slick [25] also sees value in constructing custom, fine-grained flow processing blocks that can be reused across NFVs. It also advocates the development of high-level control software that determines who conducts the processing and the traffic routes across processing blocks. Fastclick [26] is capable of processing network traffic at up to 40Gbps and uses dynamic programming to accelerate the suggested network function parallelism. Coco [27] is a lightweight and optimized flow consolidation framework designed to optimize the performance and resource consumption efficiency of SFC flows in NFV. CoCo solves the challenge of integrating parts by implementing a performance-aware placement algorithm based on 0–1 secondary planning. Furthermore, CoCo proposes a novel push-aside scaling method to avoid performance degradation when scaling. In contrast, StatelessNF [28] makes significant performance compromises due to all state accesses being remote.

Graph algorithms help solve large-scale and complex flow topologies, and the same approach is also practical for packet acceleration processing. Ref. [29] provides a straightforward graph-merging algorithm that treats the initial request flow as a pGraph, and a directed acyclic graph with a single path. The graph indicates the path length; the path is then integrated into the flow volume of a single strategy map, and the flow and passing nodes are changed accordingly. Ref. [30] extends the study of graph-merging approaches by developing an algorithm for segmenting the flow into terminals, classifiers, modifiers, shapers, and statics, as well as for minimizing the path length between the graph's input and output terminals.

3.2.2. Middlebox

When a user's service request runs through many VNFs in the back end, it is evident that efficiency will suffer if each user flow must transit through all the VNFs. Therefore, it is common to adjust the path of user service requests to accelerate the network services. For instance, the same paths can be merged into optimized ones by analyzing the user service flow with algorithms. This component, which is placed between the user's service requirements and the actual provision of network services, is called a middlebox. The main idea is disaggregating packet processing. Ref. [31] extends middlebox disaggregation by decomposing VNF into independently deployable, loosely linked, lightweight, and reusable packet processors. As shown in Table 3, VNFs are then created by combining these deployable NFVs into a packet processing pipeline.

Table 3. The acceleration solutions in EMS by middlebox.

Name	Strategies	Effectiveness	Advantage	Disadvantage
SpeedyBox	Optimizations in a service chain to eliminate processing redundancy.	Reduces the latency by 59 % and achieves 2.1× processing rate	Optimizations in a service chain to eliminate processing redundancy.	Only support NFs can decouple in service chain.
CoMb	Implements low-level processing components for capturing packets, parsing packet headers and reconstructing TCP sessions	CoMb reduces the network provisioning cost 1.8–2.5× and reduces the load imbalance by 2–25×.	Built and managed as standalone devices	Mainly focuses on consolidating NFs
Polycube	In-kernel packet processing applications	31.8×, 7.5× and 1.4× factor, respectively, for nftables with tables and OVs	A very small overhead compared to vanilla eBPF applications	Performance is not as good as DPDK
eVNF	Leaves the simple but critical tasks inside the kernel with XDP and lets complex tasks be processed outside XDP	Improves the number of completed requests per second by 1.4× times	Allows building VNFs with both speed and flexibility	Only considers one VNF per VM.

CoMb [32] is designed and managed as separate devices squander infrastructure hardware and network management resources. On the other hand, consolidated middleboxes share a common hardware platform, necessitating a re-architecture of CoMb software-centric middleboxes at the device and network layers. The CoMb implementation supports low-level processing components such as packet capture, header parsing, and TCP session reconstruction. However, establishing performance isolation, security, and fault tolerance when running many reusable NF modules on the same hardware platform is challenging due to CoMb's failure to solve implementation concerns (software design and performance optimization). In practice, an abstract controller called Speedybox [33] is constructed at the top level, followed by the creation and improvement of an allocation algorithm with detection capabilities via delay detection processing.

eBPF is a ground-breaking technique originating in the Linux kernel that enables the execution of sandboxed programs within an operating system kernel. It is used to safely and efficiently enhance the capabilities of the kernel without modifying the kernel source code or loading kernel modules. Polycube [34] is a software framework designed to bring NFV to in-kernel packet processing applications with eBPF. Polycube offers unprecedented flexibility and customization. It allows the building of arbitrary and complicated network function chains with efficient in-kernel data planes and flexible userspace control planes that are isolated, persistent, and composable. Furthermore, cubes are polycube network functions that may be dynamically produced and injected into the kernel networking stack, easing debugging and introspection, two essential aspects in contemporary cloud settings. eVNF [35] takes XDP to build the firewall (eFW), deep packet inspection (eDPI), and load balancer (eLB), demonstrating that eVNF may dramatically improve service throughput while reducing latency and CPU utilization.

3.3. Network Functions Virtualization Infrastructure (NFVI)

As mentioned previously, the acceleration with NFVI is mainly related to the network. We give a comparison in Table 4.

Table 4. The acceleration solutions in NFVI.

Name	Strategies	Effectiveness	Advantage	Disadvantage
Hyper-Switch	Removes the costs of hypervisor entries and guest announcements by using state-aware batching of packets	Achieves a peak of 81 Gbps as compared to only 31 Gbps under Xen and 47 Gbps under KVM	Combines the advantages of both driver domains and hypervisors while eliminating the software overheads for achievable I/O performance inherent in hypervisors	Not supported in every OS
BESS	A modular framework for software switches to overcome the severe contradiction between limited NIC abilities	No experiment	Makes it possible for a user to customize the packet processing by using a number of modules	Does not support multi-core architecture
VALE	A virtual local Ethernet software	This can achieve very high-speed communication between different types of VMs (up to 20 Mpps within a short frame)	Both QEMU and KVM hypervisors have to be modified to accommodate the new network backend.	Not supported with Xen
CuckooSwitch	Compacts the forwarding table lookup based on the design of cuckoo hashing	Can process 92.22 Mpps for a packet size of 64B when using eight 10 Gbps Ethernet interfaces	Employs DPDK to achieve high throughput for packet processing when handling a great deal of L2 rules	Losing flexibility in switching logic
mSwitch	Extends the VALE switch to provide a logically separated switching fabric and the switching logic	A modified OvS with minor code changes results in an up to 3 times speedup	High-performance packet I/O and secure data path using memory buffers between virtual ports	Does not receive entire batches of packets instead of having per-packet semantics
SnabbSwitch	The implementation of vhost-user as well as the use of a trace compiler	Throughput with 13.78 Mpps while the theoretical maximum for a 10G Ethernet device is 14.8	Proven to outperform OvS-DPDK and can yield almost the same performance as hardware-based solutions such as SR-IOV	Does not support multicore CPU platform

3.3.1. Hypervisor Related

Given that NFVI serves as the primary infrastructure for the functioning of VNF, the primary bottleneck is determining how to expedite communication between VNFs. For instance, NFVI will build the fundamental components of virtual switches to manage virtual switches across several hypervisors. Hyper-switch [36] is a prototype for Xen that compares its performance with the default network I/O architecture used by Xen and the vhost-net architecture used by KVM. The hyper-switch prototype outperformed both, especially where network connectivity between VMs was required. ClickOS [18] is another Xen-based software middlebox architecture and is also a software-based Ethernet switch with built-in efficient, high-performance, and highly concurrent hash tables to create compact and swift FIB searches.

3.3.2. Distributed Switch

Another option to accelerate the distributed switch is to optimize the single switch's I/O processing. The acceleration technology can implement by kernel or kernel-bypass. For accelerating in the kernel, the VALE switch [37] is an in-kernel virtual switch using the netmap pipe to provide a direct connection between two virtual machines. The virtual machine gains exclusive access to the host's physical devices. The most used technique with kernel-bypass is related to DPDK but implemented on an open switch. OvS-DPDK is a kind of accelerated OvS with DPDK datapath that bypasses the host kernel by utilizing the DPDK libraries. It appears as though the DPDK PMD driver is creating a userspace vSwitch on the host. Therefore, OvS-DPDK can deliver up to ten-fold the native OvS. BESS [38] is a modular framework for software switches that address the severe incompatibility between limited NIC capabilities and changing user needs. This enables developers to include NIC functionality into their products with minimal performance overhead.

3.3.3. I/O Enhance

When performing a software-based switch, the additional capacity is required to minimize collisions in hash tables and avoid locking costs while allowing several threads to read from the forwarding table simultaneously. I/O enhancements are helping in dispatching the packets to the virtual NIC. CuckooSwitch [39] is a software-based Ethernet switch that is built around a memory-efficient, high-performance, and highly concurrent hash table that enables compact and quick forwarding information base (FIB) lookup. CuckooSwitch combines a novel hash table design to develop a best-in-class software switch. SnabbSwitch [40] is a virtual switch designed to run in user space and achieve carrier-grade performance. It is built on an efficient packet-switching algorithm architecture, which sparked the invention of the vhost-user. mSwitch [41] is an accelerated Open vSwitch module with minor code changes that boosts performance by 2.6–3 times; and a filtering module that can direct packets to virtualized middleboxes. mSwitch is also a learning bridge with 45 line codes that outperform FreeBSD's bridge by up to 8 times.

4. Accelerations in NFV Management

This section discusses how to accelerate from a managerial standpoint. As previously stated, the ETSI NFV framework attributes a management responsibility to each infrastructure piece. We classified and structured the corresponding acceleration methods as in Figure 3 according to the framework's design.

4.1. Virtualization Infrastructure Manager (VIM)

Since the solutions are all related to management, we combine the solutions in Table 5.

4.1.1. Software Defined Network (SDN)

The distinction between VIM and NFVI is in the network's control ability. VIM is in charge of distributing network control between NFVIs, which is an ideal scenario for SDN. SDN's primary goal is to abstract the component responsible for forwarding decisions (control plane) from switches and routers and place it in software operating on general-purpose hardware. Nfvnice [42] discovered that by integrating SDN and NFV, SLAs could be successfully met, network monitoring and use could be more thorough, and the NF's network overhead could be globally reduced. The study employs an NF state manager and a flow manager to maintain real-time control over the status of the two resources.

Microboxes [43] enable the decomposition of complex NFVs into parts and the construction of dynamically and effectively coupled chains of functions. Microboxes provide modular protocol processing engines that can only be configured to handle the functionality required by a particular flow. A middlebox service chaining perspective that is not packet-centric is required to ensure that individual flows acquire appropriate protocol functionality. Individual packet arrivals, protocol events, and application-level actions produce and consume events generated by microboxes. Additionally, this establishes a publish–subscribe architecture that enables the development of convenient higher-level interfaces that are event-driven rather than packet-driven. Both studies, as previously indicated, aim to synchronize the status of NFVs to lessen their reliance on one another.

Table 5. The acceleration solutions in VIM.

Name	**Strategies**	**Effectiveness**	**Advantage**	**Disadvantage**
Nfvnice	Complementing the capabilities of the OS scheduler but without requiring changes to the OS's scheduling mechanisms	Appropriate rate–cost proportional fair share of CPU to NFs and significantly improves NF performance (throughput and latency)	Reducing wasted work across an NF chain	Not support in every OS
Microboxes	Support transport- and application-layer middleboxes	This can provide between 31% and 57% improvement in throughput and a 32% to 47% reduction in latency while only using one core.	Eliminates redundant processing across a chain and enables a modular design	Does not support Layer2
UNO	A generalized SDN-controlled NF offload architecture	Reduces power usage by up to 2×, and reduce the control plane overhead by more than 50%.	Makes optimal use of the host's and sNIC's combined packet processing capabilities	Needs hardware support
StatelessNF	Implemented three example network functions (network address translator, firewall, and load balancer)	Able to reach a throughput of 10 Gbit/s	Architected around efficient pipelines utilizing DPDK	Can support a limited scenario
OpenNF	Combination of events and forwarding updates to address race conditions	State can be compressed by 38%, improving the execution latency from 110 ms to 70 ms	Ensure lock-step coordination of updates to NF and network state	Not supported by complex NFV

4.1.2. Control Plane

A control plane design [44] offers the coordinated control of both internal and network forwarding states, enabling quick, safe, and fine-grained flow reallocation between NF instances. UNO [45] develop an offloading architecture that dynamically optimizes by making the best use of sNICs and host packet processing capabilities without requiring changes to the data center's administration and orchestration. Building a rule translation algorithm that maps NF traversal rules from an external controller to the component host/sNIC switches ensures that the controller's packet routing semantics are appropriately implemented. Meanwhile, the controller's NF configuration is presented, which formulates and executes an NF placement algorithm that dynamically determines the ideal position for an NF. Thus, sNIC and interconnection resources are efficiently utilized, migration techniques that effectively minimize packet loss during NF relocation are avoided, and the NF's internal state is maintained during the relocation process.

Finally, Ref. [28] introduced stateless network functions, a ground-breaking design approach that decouples the state that network functions must retain from the activities that they must perform. This simplifies state management and addresses several concerns that existing solutions have highlighted. Beyond the above studies, Ref. [46] offered a machine learning-based strategy that begins with the first packet of a flow and makes its decision from there. A fundamentally different technique is to make the offloading decision based on packet sampling. This compares the two ways in terms of complexity, offloaded traffic share, and table occupancy. The results indicate that the first packet's offloading decision based on machine learning is doable. The sampling strategy achieves equivalent performance only at extremely high sampling rates.

4.2. Virtual Network Functions Manager (VNFM)

The solutions in VNFM are organized as shown in Table 6.

Table 6. The acceleration solutions in VNFM.

Name	Strategies	Effectiveness	Advantage	Disadvantage
Metron	Realizes stateful network functions at the speed of 100 GbE network cards on a single server	2.75–8× better efficiency, up to 4.7× lower latency, and 7.8× higher throughput than OpenBox	Zero inter-core communication	Needs hardware support
ParaBox	Distributes packets to VNFs in parallel	Reduces the service chain latency by up to 37.7% and increases the downloading throughput by up to 10.8%	Not only significantly reduces the service chaining latency, but also improves throughput	The complex SFC may increase the latency
Octans	An NFV orchestrator to achieve the maximum aggregate throughput of all SFCs in many-core systems	Improves the aggregate throughput comparing to two state-of-the-art placement mechanisms by 26.7% 51.8%	First formulates the optimization problem as a non-linear integer programming (NLIP) model	Needs a very long calculation of the NLIP model
P4SFC	Leverages P4-capable switches to accelerate packet processing by offloading proper NFs to the switches	Increases 20% performance	Fully offloadable, partially offloadable and non-offloadable according to the limitations of P4	Only supports P4-capable switches to accelerate SFCs

4.2.1. Service Chain

The administration of the VNFM layer is intended to be similar to that of the VNF life cycle. It will undertake actions such as adding, removing and upgrading the services supplied by the VNF. Meanwhile, it may be tasked with assessing the requirements of the service chain. Along with defining the service orchestration method, the service chain describes the resource requirements for the service, such as the number of required VNFs and the network interface to use. The Open Network Automation Platform (ONAP) and Open Source MANO (OSM) are well-known orchestration solutions for VNFM. These systems exchange essential service specifications via descriptive languages such as Yet Another Next Generation (YANG) or TOSCA to reduce the service function chain (SFC) latency by leveraging parallel packet processing capabilities across NFVs. However, since the VNFM's process description is pre-written, the service description cannot parse the service independently. Thus, the main issue in VNFM is how to disassemble the user process correctly and generate a description spec.

Metron [47] sought to mitigate the state's impact on NFV distribution by including an early statement in the packet flow. The resulting packet flow is then partitioned between stateless and stateful activities. Metron instructs all programmable hardware (i.e., switches and network interface cards) to perform stateless operations while routing incoming packets to CPU cores performing stateful operations. ParaBox [48] is a hybrid packet flow processing architecture that dynamically distributes packets to VNF in parallel and intelligently blends their outputs to preserve accurate sequential processing semantics where possible. Octan [49] is an VNF orchestrator that facilitates the optimal arrangement of SFCs within a server. It begins by deriving a non-linear Integer Programming (NLIP) model for identifying critical optimization parameters. The critical element for problem resolution is defined as how NFVs can impact an NF's throughput incomparable or dissimilar SFCs due to cross-node memory access and intra-node resource congestion.

4.2.2. P4

P4 is an open source domain-specific programming language for network devices that specifies how data plane devices (switches, routers, NICs, and filters, for example) process packets. The P4 ecosystem is comprised of a diverse set of products, initiatives, and services. To discover more about P4 and to join the community, please visit the P4 website. DE4NF [50] presents P4-based software switches that do high-speed flow table lookups and packet header inspection, resulting in a lighter manager. DE4NF's NF manager receives processed packets from software switches, extracts the result of the flow table, and routes packets to the appropriate service chains. In addition, DE4NF builds an efficient event management system in which the transport layer or lower events are created in switches and kept in a tunnel header that is put into each packet upon arrival.

The P4 and service chain work very well together. P4SFC [51] is a high-performance SFC system that takes advantage of P4-capable switches to accelerate packet processing by offloading the appropriate NFVs to the switches. To steer the packet flow of new SFCs at runtime, P4SFC creates a dynamic P4 data plane with reconfigurable execution logic that can be modified without interfering with the current execution logic. Furthermore, P4SFC provides state consistency between the server and switch for partially offloaded NFVs, which creates a state library that automatically synchronizes the server and switch states. The experimental results demonstrate that P4SFC significantly improves the performance of real-world SFCs.

4.3. Network Function Virtualization Orchestrator (NFVO)

The solutions in NFVO are organized as shown in Table 7.

Table 7. The acceleration solutions in NFVO.

Name	Strategies	Effectiveness	Advantage	Disadvantage
POSCARS	An efficient predictive and online service chaining and resource scheduling scheme	Randomly incurs the highest response time (47 ms), as it disregards information about workloads or communication costs when scheduling requests	Achieves tunable trade-offs among various system metrics with stability guarantee	Does not support service chaining and scheduling scheme with multi-resource fairness consideration among VNF instances
SLOMO	Ensures the performance of each VNF while minimizing the total energy consumption of the data center	Reduces energy consumption by 7.6% and the running time cost by 63.2% on average compared to state-of-the-art methods.	On deep reinforcement learning (DRL) to process complex large-scale network state spaces in real-time	Need booting data for the reinforcement learning
VNF-AAPC	Two methods to tackle the VNF AAPC problem with integer linear programming (ILP) and heuristic-based method	(No specific experiment results)	Incorporating accelerator-awareness in VNF-PC strategies can help operators achieve additional cost-savings	The problem has been separated into two pieces
Microscope	A performance diagnosis tool for network functions that leverage queuing information at NFs to identify the root causes	Correctly captures root causes behind 89.7% of performance impairments, up to 2.5 times more than the state-of-the-art tools with low overhead	A performance diagnosis tool that identifies the causal relations in a directed acyclic graph (DAG) of NFs without any knowledge of their implementation	Only supports with interrupts, software bugs, traffic bursts, resource exhaustion
Prune and plant	Reduces the overall delay while limiting the number of duplicated packets	Decreases 20% delay	The dependency of the NFs is characterized by a directed acyclic graph (DAG)	Does not support to handle parallelizable NFs of multiple network services
SIVA	Mathematically for both unprotected and protected SFCs	(No specific experiment results)	Achieves a close to optimal performance with a much quicker running time	Does not support the DAVS problem and DSVS problem when traffic demands vary in different time intervals

Table 7. *Cont.*

Name	Strategies	Effectiveness	Advantage	Disadvantage
A-DDPG	Formulates the VNF placement and traffic routing problem as a Markov decision process model to capture the dynamic network state transitions	(No specific experiment results)	Uses the attention mechanism to ascertain smooth network behavior within the general framework of network utility maximization (NUM)	Only cares for the observation process, training process, and online running process

4.3.1. Schedule/Predit

NFVO is responsible for service scheduling and administration and is primarily responsible for NFV configuration planning, schedule management, and service runtime scheduling. The majority of contemporary methods and techniques are centered on algorithm processing. Nowadays, the bulk of methods and procedures are based on algorithm processing. To begin, VNF-AAPC [52] uses integer linear programming (ILP) to optimize VNF placement, chaining, and accelerator assignment simultaneously while adhering to all NFVI requirements and efficiently allocating regular NFVI resources as well as hardware accelerators to VNF chains.

4.3.2. Placement

Meanwhile, simulate the VNF placement and traffic routing problem using a Markov decision [53] process to account for dynamic network state changes. For scheduling, Ref. [54] presents a scalable, distributed, and online method for configuring a trade-off between a large number of system parameters while maintaining stability, all while exploiting predictive scheduling power. Ref. [55] suggested that earlier performance prediction frameworks perform poorly on contemporary architectures and NFVs because they consider memory as a monolithic unit and ignores the fact that the memory subsystem has several components that might individually create congestion. An adjustable trade-off between numerous system metrics is achieved with POSCARS [56], an efficient, distributed, and online method that uses predictive scheduling to maintain stability. POSCARS presents three variations that employed randomized load balancing to reduce the sampling overhead. It shows that POSCARS and its derivatives can achieve near-optimal average system costs while reducing average request response times. Maintaining resource counters at individual NFVs is insufficient, as resource contention can spread between NFVs and over time.

A directed acyclic graph represents the VNF's dependency (DAG), deploying VNFs in the optimal locations and processing them concurrently without breaking the DAG, reducing overall time. However, tackling the delay minimization problem is NP-hard and may send a significant number of duplicated packets into the system, burdening it. To address these difficulties, the Prune and Plant [57] has a polynomial computing cost and reduces the overall time while minimizing packet duplication. Prune and Plant consists of two stages: in the Prune step, we convert the original DAG into a series-parallel graph (SP-graph), which reduces NP-hardness while preserving VNF parallelism.

4.3.3. Benchmarking

A benchmark was added to NFVO because valid measurements are required to support sound decision making. In later chapters, we will talk about the impact of undermeasurement on the acceleration of NFV. Microscope [58] is a network function performance diagnosis tool that takes advantage of queuing information at NFVs to pinpoint underlying causes (i.e., resources, NFVs, traffic patterns of flows). The evaluation of realistic NF chains and traffic demonstrates that they can correctly identify the underlying causes of 89.7% of performance degradations, up to 2.5 times higher than state-of-the-art low-overhead solutions.

5. The Significance of Cloud Infrastructure for NFV

Mobile networks will drive the majority of current NFV demands. Telecommunications standards groups have made mobile network components natively NFV-compatible in recent years. To the point that components such as those included in 5G's Core Network already support the NFV architecture natively, making the development of the Core Network more adaptable. As cloud service providers demonstrate an increasing ability to deliver reliable cloud infrastructure, many successful cases are being put on cloud environments. We anticipate that the cloud data center will be the principal setting in which NFV will be implemented in the future. This section will address the demand for and impact of NFV on cloud infrastructure. Additionally, we also analyze that each ETSI NFV framework's components can be accelerated for its use in cloud infrastructure.

5.1. 5G NFV Platform Requirements

5G is the next generation of mobile networks. In comparison to 4G, 5G requires a higher level of performance. NFV acceleration is critical for delivering 5G by improving the functionality and architecture of 5G radio access networks. NFVs are implemented as software components. NFV overcomes some of the 5G difficulties but introduces new ones. Researchers have made some efforts to investigate the issues that have arisen due to the integration of NFV with 5G. However, Ref. [59] focuses on networking problems and [60] on making optimal decisions regarding the placement of NFVs and CPU allocation in a host, few researchers have examined how to consider the challenges in NFV-enabled 5G networks with concurrently introduced acceleration techniques. Soon, we will undoubtedly see NFV acceleration integrated into or even as a component of next-generation mobile networks to offer high-performance mobile services to end customers jointly.

Scalability is critical while developing an acceleration solution to support 5G. To begin with, the employed acceleration techniques require minimal, if any, changes to operating systems, virtual switches, hypervisors, and network interface cards. Similarly, modifying NFVs to conform to a particular high-performance framework is not permitted, and new modules are not required to deliver new NF capabilities. The fast NFV system would have performed better if it had been developed in a more conventional development environment (e.g., Linux and FreeBSD). Second, the acceleration solution must be built to work with various processors, network interface cards, and hardware platforms (e.g., Intel ×86, IBM POWER). This allows for the easy integration of the solution into a variety of platforms. Thirdly, the solution should enable the deployment and migration of NFVs across various servers. These tools should enable NFV developers to work with high-level abstractions while focusing on rapid development and speed. Additionally, proofs of concept for new acceleration methods should be carried out prior to large-scale NF development to ensure their scalability.

5.2. Cloud Infrastructure: Of NFV

Through software technology, NFV implements network functions in regular hardware, and throughout this process, the requirements of traditional CT are implemented through IT technology. Originally, CT had a single functional specification. It demands an exceptionally high level of performance and stability, which is difficult to achieve with current information technology. However, following the growing demand and the evolution of technology, the standard specification of CT has become increasingly common and modular, and IT technology can sustain more extreme performance and stability requirements. On top of that, the catalyst for integrating these two factors is cloud service. The reasons for this integration can be analyzed from three angles:

1. **CT-compliant service requirements**: Cloud services already host over 92% of global services, and as the infrastructure provider for these services, cloud services can provide 99.9999% of service-level requirements. This allows us to have a very high confidence level in using cloud services as our infrastructure.

2. **Telecom-center level bandwidth**: In 2021, Amazon launched Amazon services that support direct 100Gbps connections for enterprises. Therefore, it can be said that, regardless of whether it is a pure cloud model with only cloud services or a hybrid cloud model with both public and hybrid clouds, each model can support 100 Gbps of services.
3. **Diversified service models and ecosystem toolchains**: Stability and high bandwidth indicate that the hybrid cloud model can be further adapted and embraced. According to statistics, more than 80% of enterprises use hybrid cloud architectures. The simultaneous use of public and private cloud resources can deliver higher service requirements. For telecom operators who need to invest in server room construction early, it is possible to integrate their existing server room with infinitely scalable services seamlessly. The rich ecosystem of toolchains of cloud services can also reduce the cost of OSS/BSS for telecom operators.

Therefore, we can be confident in our belief that, in the actual deployment of NFVs in the future, cloud services will be used extensively or even wholly to provide carrier-grade services. Amazon Web Services (AWSs) has already successfully implemented a 5G server room core network combined with AWS outpost services and direct access to operate in the cloud service environment.

5.3. The Strategies of Cloud Infrastructure Acceleration

Although cloud infrastructure is generally a virtualized execution environment, it is delivered as a service which means that the uniqueness of its hardware and the unmanageability of its software must be taken into account. Additionally, because cloud service providers offer the hardware for cloud services, the option is restricted, and there are even specific standards for the hardware environment. This is why we do not prioritize hardware solutions. A similar issue exists with acceleration technologies enabled by virtualization, which vary according to the cloud provider. Additionally, we adhere to the ETSI structure framework and categorize the acceleration techniques by block as shown in Table 8.

Table 8. An ETSI framework perspective on acceleration strategies that can be used in cloud infrastructure.

NFV Function Blocks	Support Acceleration in Cloud	Strategies
EMS	Yes	VNF routing and orchestration
VNF	Yes	Operation system tuning
NFVI	No	N/A
VIM	No	N/A
VNFM	Yes	Service function chain/dispatch Optimization
NFVO	Yes	Scheduling and placement

The cloud execution environment has a negligible impact on EMS. The architecture is implemented as a logical service delivery layer between the infrastructure's VNFs. Thus, current acceleration techniques such as middleBox and routing enhancement can be used indefinitely. VNF acceleration, both in terms of OS network tuning and I/O access path improvement, should continue to significantly impact network performance. One thing to note is that some acceleration techniques integrate virtualization technology, which depends on whether the cloud environment in which it is running supports the corresponding virtualization technology.

Acceleration via the NFVI and VIM blocks is nearly brutal. Since cloud service providers frequently utilize customized virtualization technologies to properly manage the virtual machines that they deliver to their clients, even if the technology is not altered, user autonomy is limited. Therefore, NFV customers cannot increase the transport link between physical hosts or the capabilities of their underlying software network switches. Additionally, while cloud infrastructure is often a virtualized execution environment, it is supplied as a service, implying that the hardware's uniqueness and the software's unmanageability must be considered.

Moreover, because cloud service providers provide the hardware for cloud services, the selection is limited, and the physical environment must adhere to particular requirements. That is why we do not place a premium on hardware. A similar issue applies with virtualization-enabled acceleration solutions, which differ according to the cloud provider. Additionally, we follow the ETSI structure framework and classify acceleration algorithms according to their block type.

VNFM and NFVO both provide abstract management flexibility as necessary. In addition, through algorithmic and resource allocation optimization in a cloud context, the VNFM and NFVO can boost the performance of NFV. It is worth noting that cloud infrastructures frequently cannot accelerate NFV via algorithms if resource use is inefficient. Again, this is because the NFV's performance is difficult to quantify. Occasionally, it is difficult to adequately comprehend the monitoring figures when no access to the underlying devices or virtualized hypervisor layer of the virtualized environment is available. The following section will address this subject in further depth.

5.4. Benchmarking NFV Performance

Benchmarking network performance is critical for a successful transition to NFV since it allows experimentation options prior to large-scale installations. Ref. [61] suggested that NFV orchestration tools must pre-configure the virtualization environment based on available NF profiles for NF performance optimization. Multiple NFVs from disparate manufacturers can be provisioned on the same computed node under the NFV paradigm, resulting in a shared multi-vendor environment. Although cloud providers supply runtime infrastructure, they have restricted NFV benchmarking capabilities due to inefficiency and high cost.

As a result, identifying performance bottlenecks for those NFVs becomes particularly challenging, as bottlenecks might emerge at any level of the NF processing cycle. Recently, a paradigm dubbed VBaaS was introduced [62], emphasizing the critical role of NF benchmarking in orchestration decision support. On the other hand, predicting NFV performance in a virtualized environment is far from straightforward. Currently, test solutions either focus exclusively on self-generated workloads or collect NFV infrastructure resource utilization statistics based on a misunderstanding of NF profiles. Likewise, adversarial workloads are required by accelerated NFV platforms such as the CASTAN tool [63] that are used to analyze and optimize the performance of NFVs. CASTAN has previously demonstrated an exceptional ability to test the NF performance using synthesized workloads; nevertheless, additional research, particularly utilizing real user workloads, is required in this area. Indeed, there is currently no unified approach as providers continue to prioritize their technologies and NF implementations.

6. Conclusions

The primary impediment to large-scale NFV implementation is that the performance delivered by previous techniques is insufficient for carrier networks. To solve this gap, considerable effort has been expended on improving the performance of NFVs or service chains. However, due to the various bottlenecks associated with a virtualized system, creating and deploying a high-performance platform presents significant obstacles. Numerous research studies have used different acceleration techniques and have focused on a variety of different acceleration purposes.

In this research, we examined the evolution of acceleration approaches in the context of NFV and developed a taxonomy by categorizing the examined works with the ETSI NFV framework based on their acceleration methodologies. We analyzed all the surveyed works inside the taxonomy and then compared these solutions to ascertain their relative merits and demerits. Additionally, we discussed developing goods, solutions, and widespread industry projects. Finally, we examined the gap between current techniques and identified some possible future research directions.

Author Contributions: Supervision, J.C.; Writing—original draft, Y.-X.H.; Writing—review and editing, J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Flow of Processing of Packets

In the network architecture, the logic of L1–L2 layers, such as the packaging and unpackaging of L2 data packets, is executed by the network interface controller (NIC). At the same time, the CPU processes the corresponding protocols according to the specifications of the L3–L7 layers. When high-speed networks are used, some of the processing logic is offloaded to the NIC to alleviate the CPU's stress, and some NICs can even offload the entire L4 layer entirely to the hardware. Due to the hardware offloading capabilities, the internet protocol stack processing of a host OS can match the existing high-speed network. To gain a better understanding of the latency issues associated with network function virtualization (NFV), it is vital to understand the network packet processing process as Figure A1:

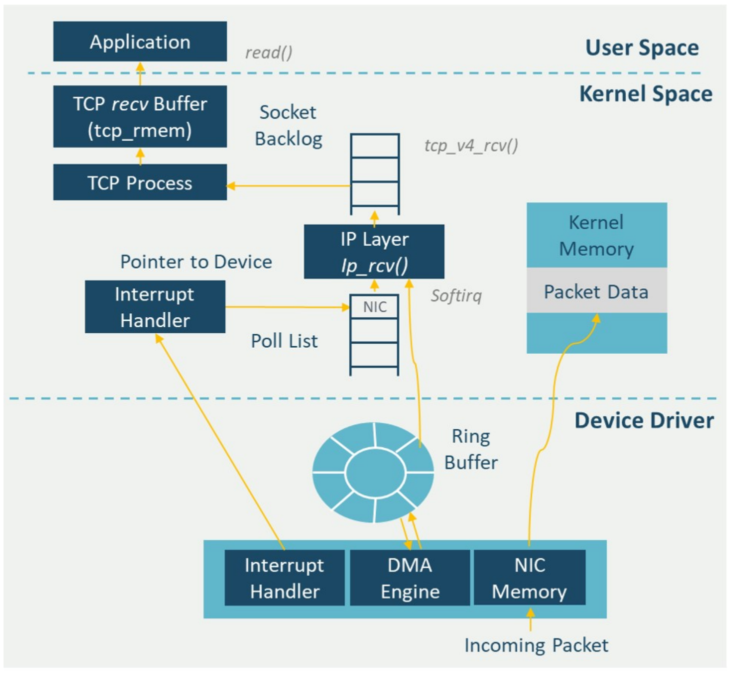

Figure A1. Linux packet processing flow.

1. When the NIC receives a packet, it uses direct memory access (DMA) to place the packet in the ring buffer (Tx), after which it uses a hard interrupt to notify the NIC it has received the packet.
2. The NIC interrupt handler allocates the kernel data structure sk_buff (socket buffer) for the packet and notifies the kernel when it has received the packet through a soft interrupt.
3. At this point, the CPU removes the packet from the sk_buff buffer for processing, which includes checking the legitimacy and identifying the type of upper layer protocol (e.g., IPv4 or IPv6), after which it removes the packet's header and footer and transfers it to the network layer.

4. The network layer determines whether the packet should be transferred to the next layer for processing or forwarded. When the network layer confirms that the packet will be sent to the local machine, it captures the protocol type of the upper layer (e.g., TCP or UDP), removes the header, and transfers it to the transport layer for processing.
5. After retrieving the TCP or UDP header, the transport layer locates the corresponding socket and copies the data to the socket's recipient cache, which is the TCP receive window.
6. Finally, the application in user space can use the socket read interface to read the data, and the program switches to kernel space and copies the data in the socket receive buffer to the user space, after which it is removed from the socket buffer.

References

1. Bronstein, Z.; Roch, E.; Xia, J.; Molkho, A. Uniform handling and abstraction of NFV hardware accelerators. *IEEE Netw.* **2015**, *29*, 22–29. doi: 10.1109/mnet.2015.7113221. [CrossRef]
2. Nascimento, M.; Primini, T.; Baum, E.; Martucci, P.; Cabelo, F.; Mariote, L. Acceleration mechanism for high throughput and low latency in NFV environments. In Proceedings of the 2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR), Campinas, Brazil, 18–21 June 2017. doi: 10.1109/hpsr.2017.7968692. [CrossRef]
3. Su, J.; Han, B.; Lv, G.; Li, T.; Sun, Z. A Heterogeneous Parallel Packet Processing Architecture for NFV Acceleration. In Proceedings of the 2019 IEEE 27th International Conference on Network Protocols (ICNP), Chicago, IL, USA, 8–10 October 2019. doi: 10.1109/icnp.2019.8888106. [CrossRef]
4. Park, Y.k.; Yang, H.s.; Kim, Y.h. Application and Comparison of Data Plane Acceleration Technologies for NFV. *J. Korean Inst. Commun. Inf. Sci.* **2017**, *42*, 1636–1646. doi: 10.7840/kics.2017.42.8.1636. [CrossRef]
5. Bonelli, N.; Procissi, G.; Sanvito, D.; Bifulco, R. The acceleration of OfSoftSwitch. In Proceedings of the 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, Germany, 6–8 November 2017. doi: 10.1109/nfv-sdn.2017.8169842. [CrossRef]
6. Cerovic, D.; Piccolo, V.D.; Amamou, A.; Haddadou, K.; Pujolle, G. Fast Packet Processing: A Survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 3645–3676. doi: 10.1109/comst.2018.2851072. [CrossRef]
7. Shantharama, P.; Thyagaturu, A.S.; Reisslein, M. Hardware-Accelerated Platforms and Infrastructures for Network Functions: A Survey of Enabling Technologies and Research Studies. *IEEE Access* **2020**, *8*, 132021–132085. [CrossRef]
8. Fei, X.; Liu, F.; Zhang, Q.; Jin, H.; Hu, H. Paving the Way for NFV Acceleration. *ACM Comput. Surv.* **2020**, *53*, 1–42. doi: 10.1145/3397022. [CrossRef]
9. Kourtis, M.A.; Xilouris, G.; Riccobene, V.; McGrath, M.J.; Petralia, G.; Koumaras, H.; Gardikis, G.; Liberal, F. Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration. In Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, USA, 18–21 November 2015; pp. 74–78.
10. Rizzo, L. Anovel framework for fast packet i/o. In Proceedings of the Usenix Conference on Technical Conference, Boston, MA, USA, 13–15 June 2012; p. 9.
11. Vieira, M.A.; Castanho, M.S.; Pacífico, R.D.; Santos, E.R.; Júnior, E.P.C.; Vieira, L.F. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Comput. Surv. (CSUR)* **2020**, *53*, 1–36. [CrossRef]
12. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *ACM Sigcomm Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
13. Wang, Y.; Esposito, F.; Matta, I.; Day, J. *Recursive InterNetworking Architecture (RINA) Boston University Prototype Programming Manual*; Technical report, Technical Report BUCS-TR-2013-013; Boston University: Boston, MA, USA, 2013.
14. Dobrescu, M.; Egi, N.; Argyraki, K.; Chun, B.G.; Fall, K.; Iannaccone, G.; Knies, A.; Manesh, M.; Ratnasamy, S. RouteBricks: Exploiting parallelism to scale software routers. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, New York, NY, USA, 11–14 October 2009; pp. 15–28.
15. Kim, J.; Huh, S.; Jang, K.; Park, K.; Moon, S. The power of batching in the click modular router. In Proceedings of the Asia-Pacific Workshop on Systems, Seoul, Korea, 23–24 July 2012; pp. 1–6.
16. da Cruz Marcuzzo, L.; Garcia, V.F.; Cunha, V.; Corujo, D.; Barraca, J.P.; Aguiar, R.L.; Schaeffer-Filho, A.E.; Granville, L.Z.; dos Santos, C.R. Click-on-osv: A platform for running click-based middleboxes. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; pp. 885–886.
17. Barbette, T.; Soldani, C.; Mathy, L. Fast userspace packet processing. In Proceedings of the 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Oakland, CA, USA, 7–8 May 2015; pp. 5–16.
18. Martins, J.; Ahmed, M.; Raiciu, C.; Olteanu, V.; Honda, M.; Bifulco, R.; Huici, F. ClickOS and the art of network function virtualization. In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), Seattle, WA, USA, 2–4 April 2014; pp. 459–473.
19. Hwang, J.; Ramakrishnan, K.K.; Wood, T. NetVM: High performance and flexible networking using virtualization on commodity platforms. *IEEE Trans. Netw. Serv. Manag.* **2015**, *12*, 34–47. [CrossRef]

20. Belay, A.; Prekas, G.; Klimovic, A.; Grossman, S.; Kozyrakis, C.; Bugnion, E. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), Broomfield, CO, USA, 6–8 October 2014; pp. 49–65.
21. Panda, A.; Han, S.; Jang, K.; Walls, M.; Ratnasamy, S.; Shenker, S. NetBricks: Taking the V out of NFV. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 203–216.
22. He, M.; Basta, A.; Blenk, A.; Deric, N.; Kellerer, W. P4nfv: An nfv architecture with flexible data plane reconfiguration. In Proceedings of the 2018 14th International Conference on Network and Service Management (CNSM), Rome, Italy, 5–9 November 2018; pp. 90–98.
23. Farshin, A.; Barbette, T.; Roozbeh, A.; Maguire, G.Q., Jr.; Kostić, D. PacketMill: Toward per-Core 100-Gbps networking. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, New York, NY, USA, 19–23 April 2021. doi: 10.1145/3445814.3446724. [CrossRef]
24. Sun, C.; Bi, J.; Zheng, Z.; Yu, H.; Hu, H. NFP: Enabling network function parallelism in NFV. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 21–25 August 2017; pp. 43–56.
25. Anwer, B.; Benson, T.; Feamster, N.; Levin, D. Programming slick network functions. In Proceedings of the 1st ACM Sigcomm Symposium on Software Defined Networking Research, Santa Clara, CA, USA, 17–18 June 2015; pp. 1–13.
26. Katsikas, G.P.; Maguire, G.Q., Jr.; Kostić, D. Profiling and accelerating commodity NFV service chains with SCC. *J. Syst. Softw.* **2017**, *127*, 12–27. [CrossRef]
27. Meng, Z.; Bi, J.; Wang, H.; Sun, C.; Hu, H. CoCo: Compact and optimized consolidation of modularized service function chains in NFV. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–7.
28. Kablan, M.; Alsudais, A.; Keller, E.; Le, F. 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), Renton, WA, USA, 9–11 April 2017; pp. 97–112.
29. Palkar, S.; Lan, C.; Han, S.; Jang, K.; Panda, A.; Ratnasamy, S.; Rizzo, L.; Shenker, S. E2: A framework for NFV applications. In Proceedings of the 25th Symposium on Operating Systems Principles, New York, NY, USA, 4–7 October 2015; pp. 121–136.
30. Bremler-Barr, A.; Harchol, Y.; Hay, D. OpenBox: A software-defined framework for developing, deploying, and managing network functions. In Proceedings of the 2016 ACM SIGCOMM Conference, New York, NY, USA, 22–26 August 2016; pp. 511–524.
31. Chowdhury, S.R.; Bian, H.; Bai, T.; Boutaba, R. A disaggregated packet processing architecture for network function virtualization. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1075–1088. [CrossRef]
32. Sekar, V.; Egi, N.; Ratnasamy, S.; Reiter, M.K.; Shi, G. Design and implementation of a consolidated middlebox architecture. In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), San Jose, CA, USA, 25–27 April 2012; pp. 323–336.
33. Jiang, Y.; Cui, Y.; Wu, W.; Xu, Z.; Gu, J.; Ramakrishnan, K.; He, Y.; Qian, X. Speedybox: Low-latency nfv service chains with cross-nf runtime consolidation. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–10 July 2019; pp. 68–79.
34. Miano, S.; Risso, F.; Bernal, M.V.; Bertrone, M.; Lu, Y. A framework for eBPF-based network functions in an era of microservices. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 133–151. [CrossRef]
35. Van Tu, N.; Yoo, J.H.; Hong, J.W.K. eVNF-Hybrid Virtual Network Functions with Linux eXpress Data Path. In Proceedings of the 2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS), Matsue, Japan, 18–20 September 2019; pp. 1–6.
36. Ram, K.K.; Cox, A.L.; Chadha, M.; Rixner, S. Hyper-switch: A scalable software virtual switching architecture. In Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), San Jose, CA, USA, 26–28 June 2013; pp. 13–24.
37. Rizzo, L.; Lettieri, G. Vale, a switched ethernet for virtual machines. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, Nice, France, 10–13 December 2012; pp. 61–72.
38. NetSys. Netsys/Bess: Bess: Berkeley Extensible Software Switch. Available online: https://span.cs.berkeley.edu/bess.html (accessed on 30 March 2022).
39. Zhou, D.; Fan, B.; Lim, H.; Kaminsky, M.; Andersen, D.G. Scalable, high performance ethernet forwarding with cuckooswitch. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, Santa Barbara, CA, USA, 9–12 December 2013; pp. 97–108.
40. Paolino, M.; Nikolaev, N.; Fanguede, J.; Raho, D. SnabbSwitch user space virtual switch benchmark and performance optimization for NFV. In Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, USA, 18–21 November 2015; pp. 86–92.
41. Honda, M.; Huici, F.; Lettieri, G.; Rizzo, L. mSwitch: A highly-scalable, modular software switch. In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Santa Clara, CA, USA, 17–18 June 2015; pp. 1–13.
42. Kulkarni, S.G.; Zhang, W.; Hwang, J.; Rajagopalan, S.; Ramakrishnan, K.; Wood, T.; Arumaithurai, M.; Fu, X. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. *IEEE/ACM Trans. Netw.* **2020**, *28*, 639–652. [CrossRef]

43. Liu, G.; Ren, Y.; Yurchenko, M.; Ramakrishnan, K.; Wood, T. 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, Budapest, Hungary, 20–25 August 2018; pp. 504–517.
44. Gember-Jacobson, A.; Viswanathan, R.; Prakash, C.; Grandl, R.; Khalid, J.; Das, S.; Akella, A. OpenNF: Enabling innovation in network function control. *ACM Sigcomm Comput. Commun. Rev.* **2014**, *44*, 163–174. [CrossRef]
45. Le, Y.; Chang, H.; Mukherjee, S.; Wang, L.; Akella, A.; Swift, M.M.; Lakshman, T. UNO: Uniflying host and smart NIC offload for flexible packet processing. In Proceedings of the 2017 Symposium on Cloud Computing, Santa Clara, CA, USA, 24–27 September 2017; pp. 506–519.
46. Durner, R.; Kellerer, W. Network function offloading through classification of elephant flows. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 807–820. [CrossRef]
47. Katsikas, G.P.; Barbette, T.; Kostic, D.; Steinert, R.; Maguire, G.Q., Jr. Metron: NFV service chains at the true speed of the underlying hardware. In Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), Renton, WA, USA, 9–11 April 2018; pp. 171–186.
48. Zhang, Y.; Anwer, B.; Gopalakrishnan, V.; Han, B.; Reich, J.; Shaikh, A.; Zhang, Z.L. Parabox: Exploiting parallelism for virtual network functions in service chaining. In Proceedings of the Symposium on SDN Research, Santa Clara, CA, USA, 3–4 April 2017; pp. 143–149.
49. Zheng, Z.; Bi, J.; Yu, H.; Wang, H.; Sun, C.; Hu, H.; Wu, J. Octans: Optimal placement of service function chains in many-core systems. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 307–315.
50. Ji, S. DE4NF: High Performance Nfv Frameworkwith P4-Based Event System. Ph.D. Thesis, Case Western Reserve University, Cleveland, OH, USA, 2020.
51. Ma, J.; Xie, S.; Zhao, J. P4SFC: Service Function Chain Offloading with Programmable Switches. In Proceedings of the 2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC), Austin, TX, USA, 6–8 November 2020; pp. 1–6.
52. Sharma, G.P.; Tavernier, W.; Colle, D.; Pickavet, M. VNF-AAPC: Accelerator-aware VNF placement and chaining. *Comput. Netw.* **2020**, *177*, 107329. [CrossRef]
53. He, N.; Yang, S.; Li, F.; Trajanovski, S.; Kuipers, F.A.; Fu, X. A-DDPG: Attention Mechanism-based Deep Reinforcement Learning for NFV. In Proceedings of the IWQoS 2021-IEEE/ACM International Symposium on Quality of Service, Tokyo, Japan, 25–28 June 2021.
54. Huang, X.; Bian, S.; Gao, X.; Wu, W.; Shao, Z.; Yang, Y.; Lui, J.C. Online VNF Chaining and Predictive Scheduling: Optimality and Trade-Offs. *IEEE/ACM Trans. Netw.* **2021**, *29*, 1867–1880. [CrossRef]
55. Manousis, A.; Sharma, R.A.; Sekar, V.; Sherry, J. Contention-aware performance prediction for virtualized network functions. In Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, Virtual Event, 10–14 August 2020; pp. 270–282.
56. Huang, X.; Bian, S.; Gao, X.; Wu, W.; Shao, Z.; Yang, Y. Online VNF chaining and scheduling with prediction: Optimality and trade-offs. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–6.
57. Bao, W.; Yuan, D.; Zhou, B.B.; Zomaya, A.Y. Prune and plant: Efficient placement and parallelism of virtual network functions. *IEEE Trans. Comput.* **2020**, *69*, 800–811. [CrossRef]
58. Gong, J.; Li, Y.; Anwer, B.; Shaikh, A.; Yu, M. Microscope: Queue-based Performance Diagnosis for Network Functions. In Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, Virtual Event, 10–14 August 2020; pp. 390–403.
59. Abdelwahab, S.; Hamdaoui, B.; Guizani, M.; Znati, T. Network function virtualization in 5G. *IEEE Commun. Mag.* **2016**, *54*, 84–91. [CrossRef]
60. Agarwal, S.; Malandrino, F.; Chiasserini, C.F.; De, S. Joint VNF placement and CPU allocation in 5G. In Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 1943–1951.
61. Cao, L.; Sharma, P.; Fahmy, S.; Saxena, V. NFV-VITAL: A framework for characterizing the performance of virtual network functions. In Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, USA, 18–21 November 2015; pp. 93–99.
62. Rosa, R.V.; Rothenberg, C.E.; Szabo, R. VNF Benchmark-as-a-Service. In Proceedings of the 2015 European Workshop on Software Defined Networks (EWSDN), Bilbao, Spain, 30 September– 2 October 2015; pp. 93–99.
63. Pedrosa, L.; Iyer, R.; Zaostrovnykh, A.; Fietz, J.; Argyraki, K. Automated synthesis of adversarial workloads for network functions. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, Budapest, Hungary, 20–25 August 2018; pp. 372–385.