Network Function Virtualization and Service Function Chaining Frameworks

Subjects: Computer Science, Information Systems | Computer Science, Hardware & Architecture

Contributor: Haruna Adoga

Network slicing has become a fundamental property for next-generation networks, especially because an inherent part of 5G standardisation is the ability for service providers to migrate some or all of their network services to a virtual network infrastructure, thereby reducing both capital and operational costs. With network function virtualisation (NFV), network functions (NFs) such as firewalls, traffic load balancers, content filters, and intrusion detection systems (IDS) are either instantiated on virtual machines (VMs) or lightweight containers, often chained together to create a service function chain (SFC).

Software-defined Networking | Network function Virtualisation | Service Function Chain | 5G | Networks

1. Introduction

With the advent of software-defined networking (SDN) and network function virtualization (NFV), middlebox functionality is increasingly being virtualized and provided in software, which can reduce power consumption, resource usage, and operational costs for service providers [1][2]. This paradigm is a departure from the use of hardware middleboxes, which are often proprietary, and thus not easily extendable by service providers. By abstracting network functionalities and implementing them in software, network operators can create network functions that suit their service level agreement (SLA) and service models [3].

NFV helps network operators by providing the right environment for the rapid deployment and scaling of virtual network functions (vNFs) [4], which are chained together in what is known as the service function chaining (SFC) architecture. Unlike the typical routing technique employed by middleboxes, where packets are simply forwarded directly from source to destination, SFC routes packets through a chain of network functions before reaching the destination (depending on the type of service and policy in use). Thus, one of the major goals of SFC is the flexibility it offers by allowing traffic to traverse diverse network functions along the service chain.

Heterogeneous network services deployed using SFC and virtual functions are installed on multiple virtual machines, sometimes combined with containers [5][6], which are chained together to provide services to users. SFC is generally considered as one of the important use cases of NFV and SDN architectures [7], which is also made possible using a centralised network controller that has a *global* logical view of the entire network infrastructure, and handles tasks such as the creation of service chains and orchestration of traffic between vNFs [8]. In terms of the location of network functions in a service chain, the virtualized infrastructure can span multiple datacentres, which calls for inter-data-centre networking or within the same data centre, which results in an intra-data-centre network.

2. Background and Motivation

There are several existing studies that focus on the general areas of SDN, NFV, and SFC. Each of the related surveys presented explores different aspects such as NFV implementation problems, traffic steering in SFC, NFV deployment acceleration, optimal NFV and SFC concepts, design, and taxonomy of NFV platforms, the placement of network functions, and resource allocation.

As one of the most recent survey works in the NFV/SFC space, Fei et al. [9] focused their work on the proposals that consider the acceleration of NFV deployments. A taxonomy of the surveyed approaches is presented and discussed, which mainly involves the hardware and software acceleration of NFV deployments.

In their work, Zhang et al. [10] presented NFV platform design choices. They presented three main open issues in NFV: (1) the use of artificial intelligence in NFV, (2) network slicing, that is, the management of network slices, the communication between slices and placement of network slices, and (3) the integration of NFV with IoT. The taxonomy of NFV platforms presented by Zhang et al. consists of prototyping, testing, deployment, management, execution, and integrated NFV platforms. researchers take a different approach by first presenting the requirements that must be met for the chaining of vNFs and some useful use-cases. This is relevant in understanding the key components involved in creating service function chains in service provider network environments.

In the survey presented by Hantouti et al. [11], they discussed the traffic steering approaches used in SFC solutions with SDN. The current traffic steering approaches are classified into three methods: header-based, tag-based, and programmable switch-based methods. They concluded by identifying QoS, scalability, security, and management as some of the challenges in SFC traffic steering. Although their survey is comprehensive in terms of SDN-based traffic steering in SFC, researchers focus the work not only on traffic steering in SFC, but also on the implementation frameworks used to achieve a scalable, resilient, and high-performance SFC.

Bonfim et al. [12] presented a review of integrated NFV/SDN architectures, where they considered implementation frameworks that combined NFV and SDN. They identified vNF scheduling and placement, improving network programmability, and the possibility of deploying multiple SDN controllers to achieve scalability, security, and standardisation of SDN/NFV solutions as some of the open challenges. researchers extend their contributions by considering proposals that implement NFV/SFC and the challenges that require further research.

As one of the early attempts in this domain, the survey by Yang et al. [13] explores the challenges faced in mobile and wireless network (MWN) environments. Their work carefully describes how software-defined wireless networks (SDWNs) and wireless network virtualization (WNV) can be used in addressing the challenges of MWN networks.

A survey on SFC was presented by Herrera et al. [1] and Xie et al. [14], where the researchers focused on resource allocation approaches in the literature. Medhat et al. [15] presented open challenges in service function chaining for next-generation networks. A taxonomy of prior work was presented, where they classified it into data -and control-plane SFC solutions.

Bhamare et al. [16] presented a detailed survey of SFC, where the researchers identified optimal resource allocation, dynamic service mapping, and policy enforcement, as some of the challenges in these environments. Li et al. [7] and Laghrissi et al. [4] focused on the placement of resources in SFC environments, where the former presented a survey of network function placement in SFC, and the latter focused on the placement of virtual resources. In their survey, Hamdan et al. [17] explored the traffic load balancing approaches used in SDN network environments.

Mirjalily et al. [18] presented a survey of SFC and NFV implementation efforts. Some of the future research directions presented by [18] include SLA and QoS approaches, online chaining of service functions, availability and resilience of chains, security, and energy efficiency.

The comprehensive survey presented by Bera et al. [19] focus on SDN technologies employed in network environments such as the data centre, edge, access, and core networks. They present their findings in relation to IoT use cases. They identified open research challenges such as platform independence, policy enforcement, mobility management, and the fully practical implementations of SDN-based solutions in IoT environments.

In their comprehensive survey, Kaur et al. [20] classified SDN/NFV approaches into availability, placement, and load-balancing solutions. They identified the ordering of SFCs, resiliency, security, topology configuration, and service placement, as some of the challenges related to current SFC implementations.

The survey carried out by Hantouti et al. [21] on SFC challenges covered frameworks that proposed to solve challenges such as path selection, orchestration, security, SFC path composition, QoS, and traffic steering. They concluded by acknowledging that more work needs to be done in developing related technologies (SDN and NFV). Researchers also consider frameworks that focus on areas such as resilience, fault recovery, performance tuning, and resource allocation to extend their contributions.

Researchers have summarised the key contributions of notable related surveys and how the work adds to the NFV/SFC domain. Unlike most existing surveys, the research also presents the detailed requirements for chaining virtual network functions in service provider environments, including some important use cases. The taxonomy researchers created in the work presents the design choices and technologies used for the implementation of different equivalence classes of NFV/SFC frameworks. This focuses on the existing problems in the NFV/SFC domain, that is, researchers categorise the surveyed frameworks based on their proposed solutions and technological design choices.

Most of the notable related surveys researchers have presented only discuss the technologies and open challenges in the NFV or SFC domain. Some exceptions to this are the surveys by Bonfim et al., which present a taxonomy of NFV/SDN architectures by categorising them into NFV-side and SDN-side designs, while Zhang et al. presented a taxonomy based on the life cycle of NFV platforms. Kaur et al. presented a taxonomy based on the optimisation approaches used in SFC, such as availability, placement, and load balancing. In contrast to previous surveys, researchers categorise the surveyed frameworks based on their proposed solutions and technological design choices.

Researchers carefully describe what each framework has been designed to achieve, what technology and approach have been used, and what performance (or other quality) benchmarks have been performed. Researchers have also highlighted the different methods and technologies used for implementing each framework. In addition to presenting an extensive list of open research challenges in NFV/SFC, researchers also discuss some notable early attempts in the literature aimed at addressing the research challenges identified, providing the reader with knowledge of some existing efforts in this direction.

2.1. SFC Chaining Requirements

When it comes to the chaining of network functions in an SFC environment, the IETF SFC draft [22] provides an architectural framework that captures all the components that are required for SFC implementation in service provider networks.

There are some useful assumptions that need to be considered when creating a chain of network functions: (1) different network functions present their own configuration and description challenges, thus, creating a generalised description for all service functions is not trivial; (2) the implementation environment of the network function affects the list of functions that can exist in a particular domain; (3) the logic employed for the chaining of service functions is not fixed, that is, it is peculiar to any given administrative domain and the requirements of service(s) to be delivered to end-users; and (4) the invocation of any service chaining criteria depends on the administrative domain in which the service functions are deployed [22].

In terms of chaining requirements, although there are domain-specific requirements that need to be in place when deploying SFCs (based on the network administrative domain), there are also general requirements for the components that are found in most SFC deployments [23]. Irrespective of the network environment(s), *global* components must be in place. These components are the service classifier, which is placed at the entry point of the network (to classify ingress flows). Flow classification helps with the decision-making process of the orchestrator in terms of traffic steering across the service chain.

The service function forwarder (SFF) is another component that forwards received traffic to the right service function (SF), and can be embedded on a physical network component or deployed as a virtual component along the service function path (SFP), which is based on the classification of ingress traffic performed by the SC [24]. The

SFF is also responsible for handling any return traffic that needs to be forwarded back to a specific service function or service classifier in the service chain [22].

Another component is the SFC proxy, which is often optional in SFC implementations and used in scenarios where other components (SF and SFF) are unable to communicate in the chain [21]. SFC deployments can be fully deployed without the use of any form of proxy component, that is, when SFC-unaware service functions are not deployed in the network infrastructure. In situations where SFC-unaware service functions are part of the service chain, an SFC proxy is used to add or remove encapsulation information; thus, these are considered as *logical* components of the SFC architecture [22].

2.2. Typical SFC Environments

As more network operators continue to adopt network slicing, which serves as the enabler for next-generation networks, the chaining of virtual network functions for efficient service delivery has become commonplace. Some common use cases can be found in today’s service provider networks. Some available common environments are the Gi-LAN network used by mobile network operators, residential/consumer services, and inter/intra-data-centre networks. Mobile network operators deploy functions such as traffic optimizers, firewalls, carrier-grade network address translation (NAT), load balancers, and DPI, at the core of the network, which is designed for subscribers that access Internet-based services [25]. Here, researchers briefly explore these environments.

2.2.1. The Gi-LAN Mobile Core Network

A typical environment in which service function chaining is deployed is the Gi-LAN network, which is a component of mobile networks used by operators to provide fine-grained user-specific services such as traffic optimisation, DPI, and firewalls [26]. Gi-LAN implementation by mobile network operators is an emerging use case for SFC architecture [27]. These services are often chained together and are provided by multiple vendors, and the traffic is steered to the right service functions, which is aimed at meeting service level agreements and policy enforcement [28].

The ability to add or remove a service becomes easier along the processing pipeline, as per-user services can be created for mobile data monetisation [29]. In terms of implementation requirements, implementing a service chain that contains a network function such as NAT, for example, requires that the function is placed on the edge sites, which is closer to the users requesting the service [30]. Another useful requirement when creating a service chain in a mobile core environment is the consideration of the SLA agreement between multiple vendors. For packet classification requirements, the classifier should be located at the packet gateway.

2.2.2. Residential and Consumer Services

Because NFV/SDN allows for the provisioning of highly specialised solutions to meet customers’ quality of service requirements, using the concept of service function chaining, service providers can steer residential traffic such as parental control and VoIP-related services. The idea of follow-the-user service deployment in residential environments is an important use case that is achieved using SFC implementations.

Network operators make use of vCPEs to easily create a chain of services that meet user requirements [31]. Users in these environments are more likely to make use of web-based applications that use HTTP as the de facto protocol [30]. One of the requirements for such deployments is to create a service chain that prioritises security [32]. A typical example is a service chain which follows the order: firewall > IDS > proxy. Security of users is a priority and an important requirement, especially in this scenario.

2.2.3. Inter-and Intra-Data-Centre Networks

SFC in the inter/intra DC environment allows for the chaining of virtualized enterprise network applications (in the case of intra-data-centre) and chaining across multiple locations, or inter-cloud, in the case of inter-data-centre networking. The ability of service functions to be instantiated across multiple datacenters (inter-data-centre networking) is a key requirement for live VM migration. The SFC architecture to be implemented should be designed to dynamically migrate service functions from one VM/container to another without disrupting user service requests [33].

Deploying and managing service function chains in an inter-data-centre setting incurs inter-data-centre bandwidth cost, deployment cost, intra-data-centre cost, and vNF costs [34]. In these environments, NFs are also used for policy-based routing of cloud services and enterprise applications [25], which means that service providers can up-sell their services easily using SFC at the enterprise. This can be achieved by making network services user-programmable.

3. SDN, NFV and SFC

In service provider networks, the process of creating, deleting, modifying, and steering traffic in service function chain (SFC) is carried out efficiently by using software-defined networking (SDN) and network function virtualisation (NFV) technologies [35]. These technologies are the key networking paradigms that are at the core of the frameworks surveyed. Researchers describe these technologies as they relate to network function virtualisation (NFV)/SFC implementation frameworks in service provider network environments, thus showing their interrelation in the operations of next-generation networks. Even though the chaining of hardware middle-boxes is possible, the use of NFV makes it much easier and cheaper [36]. Thus, SDN is employed for orchestrating virtual network functions by providing centralised logical control and the creation of service chains.

3.1. Software-Defined Networking

Software-defined networking decouples the control plane from the data plane in the networking devices. Traditional non-SDN networks often have control and data planes integrated on a single device, which brings about challenges such as management complexity and scalability issues. Implementing centralised network control using

SDN controllers results in easier service deployment and management [36]. This helps service providers to easily steer traffic between NFs by scaling across multiple physical machines.

The functional separation of the network infrastructure into control and data planes, is the core concept behind SDN. The application layer consists of various network applications, providing network services that use the Northbound Interfaces for sending requests to the control plane (centralised logical control). A global view of the network infrastructure is maintained by an SDN controller such as OpenDaylight [37], POX [38], RYU https://ryu-sdn.org/ (accessed on 12 December 2021), or a custom-built controller can be used to manage network functions, which handles requests coming from the network applications, and sends instructions to the data plane of the network for packet processing [39].

Using the southbound application programming interfaces (APIs) and the OpenFlow protocol, rules are sent down to devices in the data plane of the network, which is responsible for packet processing and forwarding. Although traditional SDN networks use the OpenFlow protocol to communicate with the network data plane by inserting flow rules on devices, the network has become more programmable over the years. Programmability allows network operators to define the processing pipeline and how packets are processed using high-level languages such as P4 [40].

3.2. Network Function Virtualization

The use of proprietary network hardware is expensive for service providers in terms of procurement, security, configuration, scalability, and maintenance costs. The European Telecommunications Standards Institute (ETSI) [41] introduced a high-level NFV architectural framework, envisaging the deployment of network functions as software, running on the network function virtual infrastructure (NFVI), which could be a general-purpose server. This proposal was introduced to take advantage of hardware virtualisation [42][43]. The deployment of network services has been greatly simplified by NFV, because the cost of acquiring new hardware middle-boxes is reduced, and several middleboxes can be virtualized and deployed on single or multiple general-purpose servers.

The ETSI architectural framework for NFV depicted shows all the important components that are necessary for deploying NFV. The operations support system (OSS) and business support system (BSS) directly interact with the vNFs. The vNF component is the network functionality, for example, a traffic load balancer, a WAN optimizer, and a firewall, etc. The hardware infrastructure consists of a virtual infrastructure with virtual computing, storage, and network components. This infrastructure is managed by the virtual infrastructure manager (VIM), which is responsible for resource allocation and embedding of virtual network functions on the virtual infrastructure. NFV Orchestrator (NFVO), which is an integral part of the ETSI NFV framework [41], is responsible for service orchestration and management [15]. One of the functions of the orchestration layer is the mapping of virtual network functions in a service chain to available physical resources. The management and network orchestration (MANO) component is responsible for the orchestration of vNFs and the chaining of services in a scenario where service function chaining is used. As shown in the SFC scenario, general-purpose hardware could be a typical high-performance commercial off-the-shelf (COTS) hardware [41].

3.3. Service Function Chaining

The vNFs implemented in NFV constitute the NF forwarding graph, which consists of network functions connected via logical network links to achieve the goal of packet processing by the vNFs. SFC is made up of NFs connected in a chain (based on service requirements and specifications) to deliver end-to-end services to end-users [41][44]. A typical service chain consists of NFs such as a NAT function, a firewall, and a traffic load balancer. In order for an SFC deployment to be complete, components such as the service classifier, SFF, Service Function Path, SFC proxy, and service function need to be in place.

The IETF SFC architecture presented by Halpern et al. [24] shows that the SFCs are either bidirectional or unidirectional, where packet processing is performed through an ordered list of service functions in a unidirectional scenario [24]. A bidirectional SFC scenario requires packet processing elements (SFs) to be placed in both directions of the service chain. SFCs are deployed as network service graphs with SFs placed carefully at different parts of the service chain. The ability to add and remove SFs dynamically along the service path is essential for the design of any SFC framework. NFV and SDN are integrated to achieve instantiation, management, and orchestration of service chains [45].

Intelligent service orchestration is important when handling various service functions, and this can be achieved when the NFV is properly integrated with SDN [46]. A classification of the user traffic is carried out by the flow classifier, which helps in deciding what network function(s) need to be traversed by the traffic before reaching its destination. In a scenario where the service requests traverse more than one NF, a service orchestrator is used to create a chain of NFs that forms the final processing pipeline toward the destination (requested service).

3.4. NFV/SFC and 5G Networks

The chaining of virtual network functions for effective end-to-end service delivery is a key enabler of Beyond 5G networks [47][48]. Since 5G-enabled networks are characterized by low latency, programmability, and the support for diverse use-cases of the future, technologies such as NFV can allow providers to deploy services that are suitable for radio access networks (RANs) and mobile core networks [47].

By implication, using NFV, SDN, and SFC, service providers can easily provide tailored solutions that meet customer demands, by carefully orchestrating user-generated traffic between an ordered list of network functions. The chaining of virtual network functions in SFC enables use cases such as the Gi-LAN mobile core network, residential and customer services, and inter and intra-datacentre networks. Other important use cases such as self-driving cars, e-healthcare [49], and mixed reality (MR) and 5G-enabled IoT [49][50] are also possible due to the flexibility offered by 5G network slicing.

Efforts such as the work by Morocho et al. [51] focus on showcasing how machine learning (ML) can be used to leverage the benefits provided by Beyond 5G networks. ML can be used with enhanced mobile broadband (eMBB) and support future Beyond 5G applications, that are envisaged to have high data rate requirements. Massive machine-type communications (mMTC) and ultra-reliable low-latency communications (URLLC) are also required to provide support for future use cases for Beyond 5G networks [52][53].

Abdelwahab et al. [54] explored how the 5G RAN can be enhanced using NFV, which could also lead to a reduction in the overall capital expenditure for telecommunications service providers (TSPs). As detailed in [54], some challenges that are related to 5G networks such as efficient scalability of vNFs between physical networks, vNF performance guarantees, and simultaneously supporting the deployment of hardware and virtualized network functions, can be overcome with the flexibility offered by NFV implementations.

4. NFV and SFC Frameworks Taxonomy

Researchers present the implementation frameworks proposed for NFV/SFC deployments. Some of these implementations are set out to solve specific problems in the SFC domain, such as resource allocation and service orchestration, performance tuning, resilience, and fault recovery.

Figure 1 depicts a summary of the taxonomy of the frameworks presented. Resource allocation and service orchestration frameworks deal with the efficient utilization of available resources, by employing techniques such as synthesizing packet processing graphs and offloading packet processing tasks onto smart NICs, while ensuring that traffic is steered to the right network functions in a service chain, and efficiently managing the life-cycle of network functions.

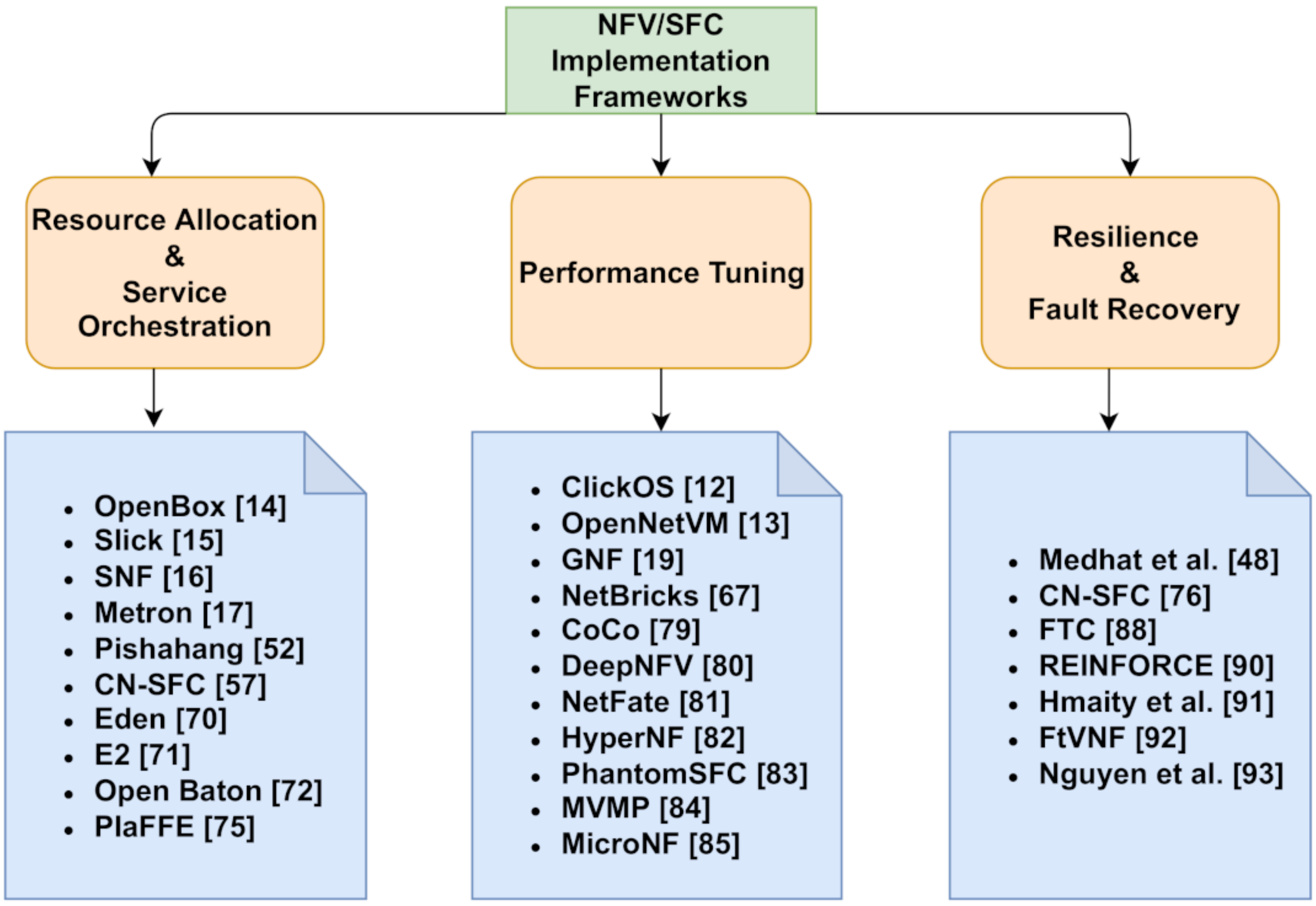

Figure 1. NFV/SFC frameworks taxonomy.

The frameworks presented under the performance-tuning category are concerned with improving the overall performance of SFCs by employing techniques such as modular SFC deployments, the use of lightweight packet processing elements, the use of acceleration frameworks for packet processing, and deep learning techniques to improve the overall chain-wide performance.

The third category in the taxonomy, resilience, and fault recovery, consists of frameworks that handle the problem of fault tolerance in SFC. These frameworks employ techniques such as network function replication and piggybacking of NF state changes across service chains to achieve resilience in SFCs.

Each of the presented frameworks strive to achieve diverse objectives and are hence very often evaluated against different and incompatible baselines. researchers have therefore chosen to create a taxonomy of the surveyed frameworks to be able to compare, quantitatively and qualitatively, the different works in their own contexts. Rather than forcing a comparison of potentially disjoint performance characteristics between frameworks that could have led to superficial superiority claims, researchers have carefully described what each framework has been designed to achieve, what technology and approach has been used, and what performance (or other quality) benchmarks have been performed. Frameworks under each category are discussed next.

4.1. Resource Allocation and Service Orchestration Frameworks

The ability to efficiently allocate network, storage, and processing resources in virtualized network environments is key in service provider networks. This is even more so in NFV/SFC environments, where user traffic type and frequency can either be deterministic or nondeterministic, bringing the need for the underlying system to provide and allocate resources efficiently. The chaining of network functions to provide end-to-end services cannot be achieved without an efficient service orchestration scheme in place [55]. Efforts such as the work by Sun et al. [56] propose algorithms to handle SFC orchestration, resource utilization, and optimization. The algorithm proposed by [56] for example, is based on the breadth-first search (BFS) algorithm, which reduces overall chain-wide latency and bandwidth consumption. Researchers presents frameworks that have been designed to achieve the goal of efficiently allocating resources in NFV/SFC and the orchestration of network services. researchers categorically focus on frameworks that handle resource allocation and service orchestration as the core contributions of the work by presenting the technologies used and the implementation approach employed by the researchers.

4.1.1. Eden Framework

The Eden framework was proposed by Ballani et al. [57] as a framework suitable for virtual network function provisioning on end-user devices. Eden leverages the concept of data-plane programmability by implementing NFs on end-user devices written in the F# language, which is a high-level programming language. Eden comprises three functional components that work together to handle packet processing tasks: the centralised logical controller, which is used for service orchestration and for providing a global view of the infrastructure. The second component is the *Stage*, which is simply a name for kernel modules, libraries, or applications on hosts that are used for the classification of packets before being sent to the Enclave (the third component in Eden).

The Enclave is responsible for handling the functionalities of a programmable data plane, which can be implemented on NICs, FPGAs, hypervisors, or operating systems. Eden maintains match-action rules in the Enclave, with traffic association carried out on the host device by the stage component of the framework. The Enclave is also responsible for interpreting the bytecode, which is obtained from compiling action functions. Eden still leaves the open question about where to best to deploy the network functions, that is, either on the user OS or on the programmable NIC. Researchers shed some light on this open challenge by arguing for a hybrid implementation framework for fast packet processing and efficient service chain creation in next-generation networks.

4.1.2. E2 Framework

The E2 framework presented by Palkar et al. [58] is designed for the management of NFV applications and resource allocation, which is achieved without necessarily knowing the low-level implementation of the applications. The target environments for the E2 system are hardware commodity servers and switches in high-performance network environments, which are typically found in today's central office locations. E2 implements a manager, which is responsible for orchestrating communication between the SDN controller and a cluster of servers.

The E2 framework also manages the placement of NFs using the proposed algorithm on available servers by monitoring the available resources and efficiently placing NFs to avoid unnecessary system overheads. Network operators can define their policies using *pipelets*, which state the steps involved in processing traffic from a specific class. A directed acyclic graph (DAG) is a key component of the E2 *pipelet*, which defines how a class of traffic is processed by the E2 NFs, with nodes representing physical switch ports or NFs.

The configuration of network functions is done by providing the following inputs: (i) an API exported by E2 for leveraging optimisation options, (ii) a method for attribute association, that is, for per-packet and port metadata; (iii) information on the scalability of the application, that is, whether it can scale across multiple cores or multiple servers, which gives the E2 framework an idea of how to handle situations of traffic overload, (iv) a method of splitting traffic across multiple NF instances by considering the constraints of the target environment; and (v) information on the NF processing capacity in terms of traffic rate, which helps with placement decisions. E2 provides service providers with the flexibility of declaring their policies without prior knowledge of the underlying network function or infrastructure.

4.1.3. Pishahang Framework

Pishahang is a multi-domain service orchestration framework for SFCs, proposed by Kouchaksaraei et al. [59][60] (with the introduction of dynamic service chaining), which combines container-based and VM-based vNFs to create a chain of network services. The framework was implemented across the OpenStack and Kurbenetes domains. In terms of service description, Pishahang uses two descriptors: the first for describing information considered to be *high level*, such as service chaining, microservices, and vNFs, while the second descriptor focuses on a more fine-grained description of vNFs, such as the required resources for running the vNFs. Pishahang is a framework that has been built to chain services across heterogeneous domains.

Pishahang used the SONATA MANO https://www.sonata-nfv.eu/ (accessed on 15 December 2021) framework, which supports the addition of new functionalities following a microservice-based architecture. Service graphs are translated by the SDN adaptor, which is sent to the controller, converted to forwarding rules, and installed on switches. To validate the features of Pishahang in chaining services across multiple domains, VM-based and container-based forwarders were chained to create an SFC with ICMP packets sent end-to-end. The use of containers for the deployment of vNFs is yet to be fully developed because of reasons such as lack of functional isolation, and security.

4.1.4. SNF Framework

SNF is a SFC framework proposed by Katsikas et al. [61], which synthesises SFCs with the main goal of performance optimisation by eliminating redundancy in packet processing across the service chain. Typical causes of redundancy are eliminated by the SNF framework by (i) creating a logical processing entity to handle all chain-wide operations on received packets, rather than handling network functions as separate processes, (ii) discarding packets that need to be discarded very early in the service chain; (iii) reducing multiple read operations by collecting read operations and constructing classes of traffic as a directed acyclic graph, which is synthesised into a classifier, and (iv) reducing the number of write operations by modifying traffic classes in a single operation.

The concept of set theory and graphs is employed for traffic classification to achieve the synthesis of similar network functions to improve the overall performance of the service chain. Another key feature of the SNF framework is the management of states across multiple network functions in a service chain, which enables the synthesis of stateful service chains. At any given time, there is a processing core that actively classifies all the received frames into the required traffic class units. The ingress traffic is hashed using RSS, which helps to serve bi-directional flows by the same processor and re-writer [61].

4.1.5. Open Baton Framework

The Open Baton framework was proposed by Carella et al. [62] as a framework for NFV service orchestration and management. The framework has the OpenStack cloud infrastructure as its underlying platform, which is compliant with the ETSI MANO architectural framework. The key component implemented by Open Baton is the multi-site service orchestration feature in heterogeneous network environments. Open Baton provides a management component for handling the life cycle of network functions, including the use of the JUJU http://openbaton.github.io/documentation/vnfm-juju/ (accessed on 20 December 2021) virtual network manager for interoperability between diverse network functions.

The framework provides support for diverse VIMS, which means there is no need to rewrite the components of the logic that is responsible for service orchestration. To speed up the NF instantiation time, drivers are provided for VIMs and VNFMs that support the deployment of containerized network functions. Scaling of network functions can be handled at runtime using the auto-scaling component https://github.com/openbaton/autoscaling-engine (accessed on 1 January 2022).

Open Baton uses Zabbix http://openbaton.github.io/documentation/zabbix-plugin/ (accessed on 1 January 2022) to monitor network activities and network function status. The framework handles resource allocation by leveraging the concept of network slicing using SDN; thus, the extensibility and interoperability of Open Baton makes it ideal as an orchestration framework for heterogeneous network functions. A fault management module, together with a management dashboard, makes Open Baton a complete solution for heterogeneous NFV orchestration.

4.1.6. Metron Framework

Metron is an NFV framework proposed by Katsikas et al. [63], which achieves high utilisation of commodity servers and underlying network resources. Metrons can offload some packet processing tasks to the underlying network infrastructure and achieve low inter-core communication using tag-based hardware dispatching for processing packets. The reduction of inter-core transfers implemented in Metron gives it the capability to process packets at the speed of the L1 cache. Metron performs stateless packet processing and classification by leveraging the OpenFlow and P4 protocols.

The problem of having a mismatch between the server and network architecture is also addressed by tagging packets to be dispatched and switched in the service chain, which is controlled by the implementation of the ONOS SDN controller. Placement decisions of synthesised packet processing graphs are carried out accurately and at a low cost by obtaining the network state. A load-balancing scheme was introduced for servers and CPU cores [63].

4.1.7. CN-SFC Framework

Dab et al. [64] presented an SDNLess SFC microservice architecture for Cloud-Native NFV, a framework for cloud-native SFC creation which uses an advanced version of network service mesh (NSM) https://networkservicemesh.io/ (accessed on 27 December 2021) and Kubernetes (https://kubernetes.io/) (accessed on 2 January 2022) for chaining cloud-native elements to form a service chain. They considered the use of SFCs in micro-service architectures and addressed the shortcomings of the NSM architecture, such as its inability to handle L2 or L3 traffic and the lack of support for advanced routing algorithms.

The cloud-native framework, CN-SFC, was also used to achieve traffic steering by efficiently load-balancing traffic across the Cloud-native functions (CNFs). An approach was introduced, which uses the weighted round robin algorithm by maintaining the weights of Kubernetes pods and distributing traffic based on pods with high weight values. TS-CNF, which is a traffic steering problem, was modelled and solved as an integer programming (IP) problem.

In terms of the evaluation of CN-SFC, a Kubernetes cluster was used, which consists of an NSM control plane, a master node, and two worker nodes. The Docker and the Kind https://kind.sigs.k8s.io/ (accessed on 28 December 2021) framework were used to run the cluster. The proposed network-aware traffic steering scheme (NA-TS) was evaluated by varying the number of flows that need access to the network service in the cluster. The number of replicas that were used for deploying the VPN and firewall services were also varied, while UDP and ICMP (to evaluate reachability) packets were generated and evaluated (jitter and packet loss in the service chain) between the two assigned pods and those deployed in the cluster. Additional metrics such as the latency of the network and service instantiation time were also evaluated.

4.1.8. Slick Framework

The Slick framework was proposed by Anwer et al. [65], which allows for the programming of network functions with a control program that describes how packets can be processed in a specific traffic set. Traffic flow is specified by applications that provide information on the elements that need to be traversed by the packets in the network. The controller deploys the selected processing elements and deploys them on the machine. The task of resource management in the framework is handled by the slick-run-time, which ensures that the links and processing elements are not overloaded with packets beyond capacity.

Network functionalities are placed as software elements, which can be installed dynamically or at the initialisation time. Events are sent to the Slick controller by the modular processing elements; thus, Slick takes care of the placement of network functions and the steering of traffic between packet processing elements. In terms of evaluation, the proposed framework was evaluated using a Mininet SDN network emulator with various network topologies. The controller was run on a separate VM, while the emulator was run on another VM with 60 SDN switches. Both VMs had eight CPU cores. One of the goals of Slick, which was achieved, is to maximise bandwidth utilisation between various network functions.

Slick allows the network programmer to use a high-level language to describe a module that handles the steering of traffic, as well as the placement of lightweight functions in arbitrary locations along the service chain. Slick does not consider other environments, such as the edge of the network; thus, their implementation is restricted to data-center environments.

4.1.9. Openbox Framework

The OpenBox framework proposed by Bremler et al. [66] is a framework for the management, development, and deployment of vNFs. OpenBox abstracts network functions as packet processing graphs, which represent the behaviour of typical network functions, such as a firewall, DPI, and NAT. Processing graphs are implemented using the elements of the click router framework [67], specifically for firewall, web cache, load balancer, and IPS.

To improve performance, OpenBox introduced a graph merging algorithm to merge the abstracted NF processing graphs, which reduces per-packet latency by minimising the number of processing blocks that need to be traversed by packets. The graph merging process starts by normalising the graphs into trees to avoid the convergence of paths, and the resulting trees are concatenated in the right order of packet processing by the network functions (to ensure chain-wide correctness of the processing pipeline).

There are three major components that make up the OpenBox framework: OpenBox Applications, OpenBox Controller (OBC), OpenBox Service Instances, and OBIs, which constitute the OpenBox data-plane. Packet processing graphs are sent down to the OBI data plane (which can be implemented in hardware or software) from the controller, which in turn receives information about the packet processing capabilities of the OBIs. The communication between the OpenBox controller and the data plane is handled by the OpenBox protocol, which defines packet processing blocks for building vNFs.

The controller is used by OpenBox to achieve multi-tenancy, smart NF placement, and NF scaling, in addition to steering traffic to the right vNF. This component of the OpenBox framework provides network application developers with a layer of abstraction for creating applications that have a specific packet processing graph and logic. The OpenBox framework was evaluated using service chain configurations and pipelined network function scenarios, with both scenarios yielding much higher throughput when compared with scenarios without OpenBox. To cater to the resilience of network functions, the researchers merged multiple NFs together to create a single processing pipeline, which provides higher throughput to one of the network functions at off-peak times of the second NF in the merged processing graph, yielding a throughput that is 20% better than the naive merge approach.

4.1.10. Piaffe Framework

PlaFFE, a Place-as-you-go in-network framework for the flexible embedding of vNFs, is a placement framework proposed by Mafioletti et al. [68], which achieves multilevel chaining and placement of vNFs implemented on SmartNICs. The overall aim of the PlaFFE framework is to maximise throughput and achieve minimum latency by embedding network functions (eNFs) on in-network processors. These embedded network function implementations reduce server CPU utilisation on end-hosts, increase throughput to line-rate speeds, and reduce latency.

A PoC which was implemented on two physical servers was achieved by chaining three network functions for authentication, IDS, and firewall functionality. PlaFFE reduces host load by performing full or partial vNF offloading while consolidating multilevel chaining. PlaFFE made use of the P4 programming language to steer traffic to vNFs or eNFs, depending on what function has been embedded as an eNF. This decision is determined by the P4 hash table or bloom filter, which has been implemented using a P4 Data Structure (P4DS).

4.2. Performance Tuning Frameworks

Improving the performance of virtual network functions, either as standalone functions or as part of a service function chain, helps network operators reduce service instantiation costs [69], and the optimal use of available resources. Researchers present frameworks that focus on optimising the performance of network functions in a service chain. Most frameworks that are designed to optimise performance devise mechanisms that can achieve goals such as reducing network function deployment and provisioning time, maximising the throughput of network applications and reducing latency.

4.2.1. Coco Framework

CoCo is an NFV framework proposed by Meng et al. [70], which was designed for the deployment of modularized service function chains (MSFCs). One of the major goals of the proposal is the consolidation of processing elements collocated on a VM and handling the placement of the modularized SFC to minimise packet transfer overhead between VMs. The researchers designed a placement scheme that selects the right SFC elements for consolidation using performance and resource-aware placement. Fairness is achieved between several NFs tied to a single CPU core using a run-time scheduler implemented in CoCo.

In terms of scalability, CoCo can utilise a push-aside scheme specifically designed to handle reduction in performance, which might arise due to scaling elements. Unlike most existing NF scalability approaches that start up a VM when there is a need to scale, which leads to more overhead in terms of latency, the push aside algorithm reduces the need for inter-VM hop creation. Rather than creating a new replica (as used by traditional vNF scalability solutions), the CoCo framework adds more resources to elements that are overloaded.

In terms of performance evaluation, the efficiency of resource utilisation and the reduction in the cost of transferring packets across the service chain are the two major benefits of the CoCo framework. Throughput and CPU utilisation were measured with CoCo implemented using Docker containers to allow for the consolidation of processing elements; thus, Open vSwitch was employed as the virtual switch for VM-VM communication. The results show that CoCo can improve performance by approximately 45.5% and a 2.46X reduction in packet transfer overhead.

4.2.2. Deepnfv Framework

DeepNFV is a lightweight NFV framework proposed by Li et al. [71], which was designed specifically for edge network deployments, with the aim of minimising the packet processing tasks at the core of the network by offloading to edge network functions. DeepNFV is built on the GNF framework [72], which uses lightweight docker containers to build network functions for the edge. The key components of the proposed framework are the deep learning models employed and the infrastructure layer, which handles the interaction between network links and devices.

DeepNFV uses deep learning to enhance tasks such as the optimisation of QoS parameters, classification of traffic, and analysis of network links. Similar to the GNF framework, DeepNFV was built to support the idea of moving network processing elements as close to the data source as possible (edge computing). As a use case for the DeepNFV framework, network traffic analysis functionality was considered by the researchers by generating basic images from network traffic. A traffic analysis-containerized network function was used for the analysis and classification of images using deep learning models.

To demonstrate the traffic analysis use case, the DeepNFV framework starts by splitting the received traffic into discrete components, which are stored as PCAP files, and the second step involves the modification of the packet headers to trim the header length or remove unimportant fields from the packets. The modified PCAP files are *cleaned* to remove duplicates before being converted into image data. The resulting images are processed by the CNN model and sent to the next network function in the chain for further action(s). The ability of the framework to classify images and the performance of the network functions at the edge of the network were evaluated, and improved performance in terms of precision and efficiency was recorded.

4.2.3. Micronf Framework

The MicroNF framework was proposed by Meng et al. [73] as a framework for the deployment of modularized service chains, using a centralised controller for service chain graph reconstruction and redundant NF reuse. Service providers describe the MSFC to be deployed by clearly defining how the elements are interconnected. This is followed by processing the MSFC using graph reconstruction and the identification of any dependencies by elements, with the aim of reusing elements where possible. The reordered MSFC is optimally placed with the goal of reducing the latency between processing elements.

The major goals of the framework are to (i) efficiently reuse elements that have similar configurations in the processing pipeline by first addressing the problem of dependency between different elements, (ii) solve the problem of VM to VM connection using a virtual switch in an optimal fashion, (iii) shorten the service chain length, and reduce packet processing costs, where necessary.

In terms of the scalability of NFs, MicroNF implements run-time scaling algorithms, which ensure minimal inter-NF latency along the service chain. The problem of selecting processing elements that are ideal for consolidation is also handled by the MicroNF framework, in addition to a placement algorithm that prioritises high performance. The speed of packet processing between diverse elements is also considered by the proposed resource scheduler, which ensures that the workload is efficiently shared among available processing elements [73].

4.2.4. Netbricks Framework

NetBricks is an NFV framework proposed by Panda et al. [74], which offers a platform for building and running virtual network functions that provide software isolation between NFs. The NetBricks framework differs from other approaches by (1) limiting the set of processing modules to core functionalities, which helps to reduce the number of modules that network application developers must deal with, and (2) allowing the customization of modules using user-defined functions, which makes the modules more flexible and optimised for better network function(s) performance.

NetBricks eliminates overheads resulting from context-switching by enforcing memory-level isolation in software and reducing I/O related overheads by introducing zero-copy software isolation [74]. Using zero-copy isolation, the cost of packet I/O is greatly reduced by NetBricks, which means that chains of network functions can be run as a single process.

NetBricks provides a major distinction in providing fault and memory isolation for NF implementations by utilising operators designed for parsing, de-parsing, transforming, and filtering packets. In addition to packet operators, the framework also provides abstractions for processing byte-streams, abstractions for control flow, and for state and scheduled events. To evaluate the performance of the NetBricks framework, two example network functions were used: the first is a simple network function that decrements the TTL of a packet and discards any packet that has a TTL of 0, and the second is a stripped down implementation of the Maglev load balancer [75], which splits ingress traffic among servers and also provides failure recovery for back-end servers.

The measurements evaluated include simple NF overheads, array bound overheads, and how *general* the NetBricks programming abstractions can be. For the latter part of the evaluations, that is, the programming abstractions, five network functions were implemented: NAT, firewall, Maglev load balancer, and a Snort-like NF that performs signature matching on ingress packets. Improved performance was observed for scenarios where (1) CPU cores and chain lengths were varied, (2) the load was varied with respect to CPU cycles and chain length, and (3) throughput measurements for single network functions with a variable number of CPU cycles for multiple isolation approaches.

4.2.5. Hypernf Framework

HyperNF is a high-performance NFV platform proposed by Yasukata et al. [76], which aims to properly utilise commodity server resources while scaling the number of network functions hosted by servers. The problem space addressed by HyperNF includes resource allocation, efficient utilisation, and high throughput when using everyday commodity servers to deploy virtual network functions.

The proposed framework is aimed at large NF deployments, where utilisation is maximised for better throughput. The use of hypervisor-based I/O is employed, which helps reduce synchronisation overhead. HyperNF was designed using three core design objectives: (i) CPU cores are not reserved entirely for virtual I/O operations, thus providing high flexibility in terms of utilisation, (ii) proper accountability for virtual I/O tasks on respective VMs, thus offering a cohesive resource allocation strategy, and (iii) VM switches should not be used for packet switching; instead, the data path of software switches is exported to the hypervisor for the purpose of forwarding and switching of packets.

HyperNF was evaluated for scenarios involving a baseline setup using the VALE [75] switch for inter-VM communication, with each VM tied to a single CPU core, and second, a scenario that consolidates network functions in a shared CPU environment by varying the number of VMs (CPU cores are shared among the firewall VMs deployed using a round-robin scheme). Both scenarios outperformed the split and merge schemes compared with HyperNF. Other tests carried out include resource allocation, NFV throughput, and SFC chain composition. A chain of 50 NFs can achieve a delay as low as 2 ms, which makes the framework ideal for SFC deployments.

4.2.6. Netfate Framework

NetFate was proposed by Lombardo et al. [77] as a framework that supports the deployment of network functions at the network edge and data centre infrastructure. The main elements of the NetFate framework are simply the clients, which receive or generate packets, and the CPE nodes, which hosts the network functions for clients to connect to the infrastructure.

The orchestrator contains an SDN controller for handling communication with OpenFlow switches, an NFV coordinator for handling VM life-cycle and hypervisor-VM communication, and an orchestration engine which collects statistics about available devices, connected clients, and network services. Each time ingress packets are received, the orchestrator (i) takes a decision on which NFVI can host the NF based on defined SLA, (ii) carries out the migration or instantiation of VMs for hosting the NFs in (i), (iii) creates a virtual service path for connecting VMs that host the NFs, (iv) forward ingress flows based on defined routing policies, and (v) terminating unused VMs, thus making resources available.

The Proof of Concept employed for the evaluation of NetFate comprises client devices and nodes that represent network access points, a controller, and an orchestrator which also authenticates and authorises users. This was made possible by the implementation of two firewall network functions at CPE nodes, where migration efficiency is measured while moving from one CPE node to the other. The NetFate framework is ideal for customer premise equipment network function deployments; thus, its performance in terms of provider equipment implementation is yet to be evaluated.

4.2.7. Clickos Framework

ClickOS was proposed by Kohler et al. [78], which uses the *elements* from the Click software router [67] to achieve lightweight middlebox packet processing. This runs on Linux VMs and a XEN-based https://xenproject.org/users/virtualization/ (accessed on 4 December 2021) optimised platform. To achieve domain isolation, each click middlebox is run on a separate Linux VM, which provides memory isolation. ClickOS achieves high performance in terms of network I/O by making the following changes to the Xen network pipe: (i) replacing the OvS backend switch, which makes it easier to map VM memory, (ii) moving the netback driver to the control plane, which serves as a communication medium with the netfront driver, and (iii) modifying the netfront driver of the VM to allow the mapping of ring buffers. The framework presented some useful modifications to the Xen backend and frontend modules to achieve faster data transmission rates.

The evaluation results show that ClickOS speeds up networking for Xen-based VMs by applying several well-known optimisation approaches, such as removing unnecessary data paths and batching of processes. The performance of the ClickOS switch was measured, in addition to metrics such as boot time, memory footprint, throughput, delay, chaining, scalability, and middle-box state insertion. An increased throughput from 8 Kp/s to 344 Kp/s was achieved by changing the driver settings, receiving grants for buffers at the initialisation time, and re-use the buffers for all packets. The VALE switch, which is an in-kernel virtual switch in Linux that allows for scalability in terms of the number of ports and throughput, was replaced by Open vSwitch.

4.2.8. Opennetvm Framework

The OpenNetVM framework was developed by Zhang et al. [79] as a framework that is ideal for high-performance vNF deployments using the Intel Dataplane Development Kit (DPDK) https://www.dpdk.org/ (accessed on 8 December 2021) and Docker Containers https://www.docker.com/ (accessed on 17 December 2021). To achieve high-speed packet I/O transfer, OpenNetVM implements zero-copy to reduce the I/O overhead associated with copying packets from the NIC for processing in the user space. Packets are DMA'd directly from the NIC to a shared memory space, which is accessible to DPDK-based network functions supported by the framework.

As depicted in **Figure 2**, a shared memory space is created by the manager, which stores the metadata information, list of service chains, and flow tables. Dependencies are encapsulated in Docker containers that host the network functions; thus, packet transfer between NFs is handled by the TX and RX threads that carry useful descriptors. High-speed packet processing is also made possible by utilising the DPDK poll-mode driver rather than interrupts.

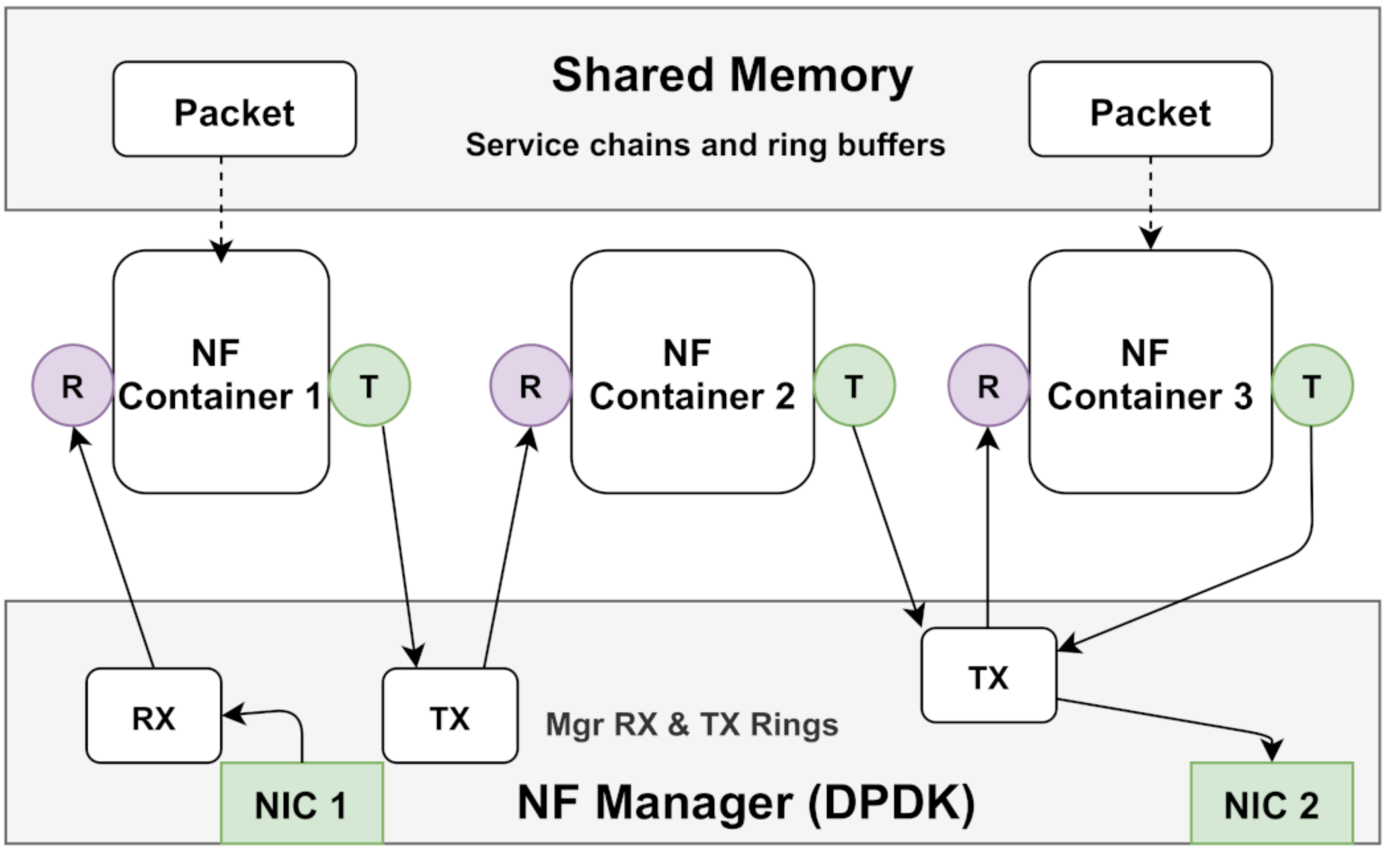

Figure 2. OpenNetVM framework.

In terms of NF-to-NF communication, OpenNetVM uses a centralised logical controller, which communicates using the OpenFlow protocol to orchestrate NF activities. The network function manager is responsible for managing memory and the NF life-cycle by handling inter-NF communication and sending keepalive messages. The transfer of packets between the NICs and NFs is also handled by the manager using the TX and RX threads. Network functions are either implemented using DPDK or as a user-space container process, which are tied to specific CPU cores in both scenarios. The framework was evaluated using metrics such as the scalability of multiple ports, overflow of the flow director, performance of service chains, and the flexibility of the framework can steer packets, which yields a much better performance when compared with ClickOS, especially in terms of throughput with variable chain length.

4.2.9. Phantomsfc Framework

PhantomSFC was proposed by Castanho et al. [80], which is an SFC framework aimed at decoupling the underlying network from the service plane. The design was implemented to be transport-independent, network agnostic, elastic, and maintains a small footprint by considering end-to-end throughput and latency. The PhantomSFC framework is based on the IETF SFC reference architecture presented by [24], which comprises a classifier, an SFF, SF, and a proxy component. NSH, which was standardised by [81], was used as the encapsulation protocol for SFCs in PhantomSFC.

Components such as proxies, forwarders, and classifiers are deployed as vNFs, and a centralised SDN controller is employed for the realisation of chain configuration and instantiation. Tasks involving chain configuration, such as creating a new chain and removing and modifying configuration rules in proxies, classifiers, and forwarders are carried out by the logically centralised controller. Using PhantomSFC, resources can be scaled by SPs based on service demands; thus, the PoC evaluation of PhantomSFC achieved improvements in throughput, jitter, and latency, using the DPDK application.

4.2.10. GNF Framework

The Glasgow Network Functions (GNF) framework was proposed by Cziva et al. [72] as an open and lightweight implementation framework for network functions in OpenFlow network environments. GNF, which is a container-based framework, achieves low overhead in terms of performance, high NF reuse, and fast deployment speed, when compared to most NFV deployments. To achieve the routing of traffic in a typical NFV scenario, policies can be implemented by adding entries or by adding a middle-box in the path of the traffic. These two approaches have drawbacks that the GNF framework attempts to eliminate.

GNF achieves dynamic placement of network functions by simply rerouting the traffic to the server with the requested NF, which allows service providers to utilise the same hosts when handling network and compute functions, thus minimising the overall infrastructural costs. Using the OpenFlow protocol, GNF can match ingress packets to the match-action table before routing packets to the specified destination.

For ease of network function deployment and management, GNF provides a user interface for global control and view of the network, a manager, GLANF router, and agent. The life cycle of network functions is handled by the GLANF Manager, which makes use of the OpenDaylight SDN controller for performing tasks such as creating, starting, stopping, and deleting primitives. Tasks such as resource allocation are also handled by the manager, which allocates network functions to hosts that have available resources.

To evaluate and demonstrate the performance of the GNF framework, six network functions were deployed: a wire, which routes packets from its ingress to egress ports, an HTTP filter, traffic control, a load balancer, intrusion detection, and a firewall, which is based on iptables.

4.2.11. MVMP Framework

The NFV platform proposed by Zheng et al. [82], i.e., the Multiple Virtual Middlebox Platform (MVMP) is a high-performance framework that has been built using the Intel DPDK platform and Docker containers. The three major components of the proposed MVMP framework are (i) abstracted virtual devices, (ii) a control plane, and (iii) a shared memory space. Packet processing by NFs is achieved by an abstraction layer that supports the deployment of multiple NFs on a single hypervisor. Network functions are run in user space as processes, which makes them lightweight, thus requiring fewer resources for packet processing. Packets are polled directly from the NIC using the DPDK poll mode driver and sent to several network functions, which also adds to the fast packet processing speed of the proposed framework.

In terms of implementation and evaluation, network functions were implemented and chained together, and the service chain performance was evaluated and compared with the OpenNetVm [79] framework, which yields 3x better throughput as the number of network functions is increased in the chain, with an overhead of approximately 4% with regard to network function isolation.

4.3. Resilience and Fault Recovery Frameworks

The resilience of virtual network functions to link, node, and chain-wide failures has been addressed using diverse technologies and methodologies in the literature. The state of all network functions (active and standby) in a chain is vital when creating a resilience mechanism; thus, building a fault-tolerant middlebox and service chain becomes imperative [83]. Different frameworks make use of various mechanisms to detect faults, fix them, and resume normal packet processing operations with as little downtime as possible [84].

4.3.1. Reinforce Framework

Kulkarni et al. [85] proposed REINFORCE, a framework for achieving resiliency of DPDK-based NFs, which provides the check-pointing of applications that reduce the state of network functions to be replicated. REINFORCE provides failure recovery of network functions across the entire chain, with the detection of node and link failures within the shortest possible time. Packet processing overhead is minimised by the separation of network function behaviour into deterministic and non-deterministic, thus committing to check-pointing the states of standby NFs in non-deterministic scenarios.

REINFORCE emphasises stateful network functions, which maintain the state of connections either globally or per-flow. The characterisation of state information enables the framework to decide whether flow updates are deterministic, which helps with the synchronisation of NFs that operate in a particular chain. The use of lazy check-pointing of the NF state and the replay of packets is used by REINFORCE to speed up the process of recovering from failures.

4.3.2. FTC Framework

Ghaznavi et al. [83] presented a framework for fault-tolerant chaining (FTC). FTC uses a different approach, which opposes existing solutions where middle-box snapshots are taken for the purpose of replicating state, or approaches where the state of middle-boxes is stored in a fault-tolerant data store. The design requirements of the FTC framework are (i) correctness of middlebox recovery, (ii) quick recovery from failures with low processing overhead, and (iii) efficient use of servers hosting middleboxes. Middlebox state information is added to the packets as they traverse the SFC chain, which is replicated in the host servers. The deployment of fault-tolerant chains is handled by the ONOS controller, which serves as a centralised orchestrator for the management of NF and chain life cycles.

To achieve fault tolerance, FTC makes use of replicas, which comprise data and control plane modules for interacting with the orchestrator. New threads are spawned by the control module in fail-over scenarios [83]. To optimise the amount of memory used for service replication, updates that have been added to the standby middleboxes are removed. FTC middleboxes are built using Click [67] elements that interact with the ONOS controller. Parameters such as replication factor, time required for failure recovery, throughput, and latency were all measured while varying the length of the service chains deployed first in a cluster of 12 servers and second on distributed servers on the cloud. The FTC was compared with FTMB [86] and NF, which is a baseline framework designed with no fault tolerance. FTC produces a much higher throughput (with an increase in service chain size) with a chain-wide overhead that is less than that obtained with FTMB.

4.3.3. Hmaity et al.

Hmaity et al. [87] presented a solution that uses integer linear programming to solve the problem of link and node failures in service chains. They modelled the physical infrastructure using a directed network graph, with representations for physical devices with the ability to host virtual network functions and links. The latencies of physical links and packet processing latency by vNFs are also represented in the model of the physical network topology. The other components that were modelled are the service chains and the vNFs, which are considered as abstract components that can process ingress packets before forwarding to the next vNF or host device.

The constraints considered for the proposed model are (i) node capacity and link latency, which capture the current state of a node, the capacity of the link in use, the latency and the highest number of virtual machines that can be hosted by a particular node; (ii) constraints related to routing, such as the location of virtual links in the physical network, to ensure that routing paths are made available on the right physical node; and (iii) constraints related to the placement of vNFs on physical devices, by ensuring that active and standby vNFs are not collocated on the same node.

The proposed models were evaluated using two example service chains, that is, a chain consisting of a web service and another service chain that models online gaming, where the impacts on node capacity and latency were considered. To solve the formulated ILP model, a bandwidth of 100 kbit/s was set for the online gaming service chain, while 50 kbit/s was set for the web service scenario. Latencies of 500 ms and 60 ms were set for the service chains. The proposed and solved models showed that achieving SFC resilience requires additional (redundant) nodes of approximately 107%. Although the proposed solution serves as a good mechanism that can handle link and node failures in service chains, it lacks the ability to provide shared protection against failures.

4.3.4. Resilient SFCs—Medhat et al.

Medhat et al. [35] proposed an OpenStack and OpenDaylight-based environment to deploy and orchestrate resilient SFCs in cloud environments. The proposed framework follows the ETSI NFV model, which is capable of traffic rerouting, in the case of faults occurring at runtime. The researchers proposed an extension to the ETSI NFV framework for SFC orchestration and management, with the implementation of a service chain consisting of two firewall network functions in standby and active modes.

They used the Open Baton framework [62] as the NFVO, which uses a messaging queue to communicate with the SFC orchestrator (the OpenDaylight SDN Controller), and the Zabbix network monitoring tool. Service function failure is simulated by abruptly terminating the process running the NF, then the Open Baton Fault Management System (FMS) switches to the standby network function, and the failed NF is recovered by the Orchestrator.

4.3.5. OpenFlow Fault Recovery

The framework proposed by Nguyen et al. [88] used the OpenFlow group table to provide a quick fault recovery and fast fail-over scheme. Their proposal eliminates the use of a centralised logical controller by exploiting the use of the OpenFlow group table for managing service function chains. Fault recovery and detection were implemented in local OpenFlow switches, which utilise an OF group table.

The need to contact the controller or NFV-MANO in the event of a failure is eliminated, which saves time on fault notification and recovery. The framework was tested using OpenStack and OF, with two vFirewall functions deployed for redundancy, which showed a reduction in SFC packet loss and an improvement in link throughput, as well as quick failure recovery.

References

1. Herrera, J.G.; Botero, J.F. Resource allocation in NFV: A comprehensive survey. IEEE Trans. Netw. Serv. Manag. 2016, 13, 518–532.
2. Cherrared, S.; Imadali, S.; Fabre, E.; Gössler, G.; Yahia, I.G.B. A survey of fault management in network virtualization environments: Challenges and solutions. IEEE Trans. Netw. Serv. Manag. 2019, 16, 1537–1551.
3. Paganelli, F.; Cappanera, P.; Cuffaro, G. Tenant-defined service function chaining in a multi-site network slice. Future Gener. Comput. Syst. 2021, 121, 1–18.
4. Laghrissi, A.; Taleb, T. A survey on the placement of virtual resources and virtual network functions. IEEE Commun. Surv. Tutor. 2018, 21, 1409–1434.
5. Bujari, A.; Palazzi, C.E.; Polonio, D.; Zanella, M. Service function chaining: A lightweight container-based management and orchestration plane. In Proceedings of the 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019; pp. 1–4.
6. Santos, J.; Wauters, T.; Volckaert, B.; De Turck, F. Towards delay-aware container-based service function chaining in fog computing. In Proceedings of the NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–9.
7. Li, X.; Qian, C. A survey of network function placement. In Proceedings of the 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 9–

12 January 2016; pp. 948–953.

8. Haleplidis, E.; Joachimpillai, D.; Salim, J.H.; Lopez, D.; Martin, J.; Pentikousis, K.; Denazis, S.; Koufopavlou, O. ForCES applicability to SDN-enhanced NFV. In Proceedings of the 2014 Third European Workshop on Software Defined Networks, Budapest, Hungary, 1–3 September 2014; pp. 43–48.
9. Fei, X.; Liu, F.; Zhang, Q.; Jin, H.; Hu, H. Paving the Way for NFV Acceleration: A Taxonomy, Survey and Future Directions. ACM Comput. Surv. (CSUR) 2020, 53, 1–42.
10. Zhang, T.; Qiu, H.; Linguaglossa, L.; Cerroni, W.; Giaccone, P. NFV platforms: Taxonomy, design choices and future challenges. IEEE Trans. Netw. Serv. Manag. 2020, 18, 30–48.
11. Hantouti, H.; Benamar, N.; Taleb, T.; Laghrissi, A. Traffic steering for service function chaining. IEEE Commun. Surv. Tutor. 2018, 21, 487–507.
12. Bonfim, M.S.; Dias, K.L.; Fernandes, S.F. Integrated NFV/SDN architectures: A systematic literature review. ACM Comput. Surv. (CSUR) 2019, 51, 1–39.
13. Yang, M.; Li, Y.; Jin, D.; Zeng, L.; Wu, X.; Vasilakos, A.V. Software-defined and virtualized future mobile and wireless networks: A survey. Mob. Netw. Appl. 2015, 20, 4–18.
14. Xie, Y.; Liu, Z.; Wang, S.; Wang, Y. Service function chaining resource allocation: A survey. arXiv 2016, arXiv:1608.00095.
15. Medhat, A.M.; Taleb, T.; Elmangoush, A.; Carella, G.A.; Covaci, S.; Magedanz, T. Service function chaining in next generation networks: State of the art and research challenges. IEEE Commun. Mag. 2016, 55, 216–223.
16. Bhamare, D.; Jain, R.; Samaka, M.; Erbad, A. A survey on service function chaining. J. Netw. Comput. Appl. 2016, 75, 138–155.
17. Hamdan, M.; Hassan, E.; Abdelaziz, A.; Elhigazi, A.; Mohammed, B.; Khan, S.; Vasilakos, A.V.; Marsono, M.N. A comprehensive survey of load balancing techniques in software-defined network. J. Netw. Comput. Appl. 2021, 174, 102856.
18. Mirjalily, G.; Zhiquan, L. Optimal network function virtualization and service function chaining: A survey. Chin. J. Electron. 2018, 27, 704–717.
19. Bera, S.; Misra, S.; Vasilakos, A.V. Software-defined networking for internet of things: A survey. IEEE Internet Things J. 2017, 4, 1994–2008.
20. Kaur, K.; Mangat, V.; Kumar, K. A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture. Comput. Sci. Rev. 2020, 38, 100298.
21. Hantouti, H.; Benamar, N.; Taleb, T. Service Function Chaining in 5G and Beyond Networks: Challenges and Open Research Issues. IEEE Netw. 2020, 34, 320–327.

22. Quinn, P.; Beliveau, A. Service Function Chaining (SFC) Architecture. draft-quinn-sfc-arch-04. 2014. Available online: https://www.ietf.org/proceedings/89/slides/slides-89-sfc-10.pdf (accessed on 20 January 2022).

23. Gasparakis, J.; Smith, K.; Zhou, D. Evaluating Dynamic Service Function Chaining for the Gi-LAN. In White Paper; Intel: Santa Clara, CA, USA, 2016.

24. Halpern, J.; Pignataro, C. Service Function Chaining (sfc) Architecture. In RFC 7665; IETF: 2015; pp. 1–28. Available online: https://www.hjp.at/(de)/doc/rfc/rfc7665.html (accessed on 20 January 2022).

25. Grønsund, P.; Mahmood, K.; Millstein, G.; Noy, A.; Solomon, G.; Sahai, A. A solution for SGi-LAN services virtualization using NFV and SDN. In Proceedings of the 2015 European Conference on Networks and Communications (EuCNC), Paris, France, 29 June–2 July 2015; pp. 408–412.

26. Naik, P.; Vutukuru, M. libVNF: A Framework for Building Scalable High Performance Virtual Network Functions. In Proceedings of the 8th Asia-Pacific Workshop on Systems, Mumbai, India, 2 September 2017; pp. 1–8.

27. Turk, Y.; Zeydan, E. An Implementation of Network Service Chaining for SDN-enabled Mobile Packet Data Networks. In Proceedings of the 2019 International Symposium on Networks, Computers and Communications (ISNCC), Istanbul, Turkey, 18–20 June 2019; pp. 1–6.

28. Kaur, K.; Kumar, K.; Mangat, V. A road to network function virtualization and applications. Adv. Math. Sci. J. 2020, 9, 4059–4066.

29. Brown, G.; Reading, H. Service Chaining in Carrier Networks. Heavy Read. 2015. Available online: https://www.qosmos.com/wp-content/uploads/Service-Chaining-in-Carrier-Networks_WP_Heavy-Reading_Qosmos_Feb2015.pdf (accessed on 20 January 2022).

30. Shojafar, M.; Pooranian, Z.; Sookhak, M.; Buyya, R. Recent advances in cloud data centers toward fog data centers. Concurr. Comput. Pract. Exp. 2019, 31, e5164.

31. Cunha, V.A.; Cardoso, I.D.; Barraca, J.P.; Aguiar, R.L. Policy-driven vCPE through dynamic network service function chaining. In Proceedings of the 2016 IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Korea, 6–10 June 2016; pp. 156–160.

32. Yan, Z.; Zhang, P.; Vasilakos, A.V. A security and trust framework for virtualized networks and software-defined networking. Secur. Commun. Netw. 2016, 9, 3059–3069.

33. Liu, Y.; Zhou, F.; Chen, C.; Zhu, Z.; Shang, T.; Torres-Moreno, J.M. Disaster protection in Inter-DataCenter networks leveraging cooperative storage. IEEE Trans. Netw. Serv. Manag. 2021, 18, 2598–2611.

34. Zhong, X.; Wang, Y.; Qiu, X. Service function chain orchestration across multiple clouds. China Commun. 2018, 15, 99–116.

35. Medhat, A.M.; Carella, G.A.; Pauls, M.; Monachesi, M.; Corici, M.; Magedanz, T. Resilient orchestration of Service Functions Chains in a NFV environment. In Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, 7–10 November 2016; pp. 7–12.
36. Sarmiento, D.E.; Lebre, A.; Nussbaum, L.; Chari, A. Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey. IEEE Commun. Surv. Tutor. 2021, 23, 256–281.
37. Medved, J.; Varga, R.; Tkacik, A.; Gray, K. Opendaylight: Towards a model-driven sdn controller architecture. In Proceeding of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, NSW, Australia, 19 June 2014; pp. 1–6.
38. Kaur, S.; Singh, J.; Ghumman, N.S. Network programmability using POX controller. In Proceedings of the International Conference on Communication, Computing & Systems (ICCCS); 2014; Volume 138, pp. 134–138. Available online: https://docplayer.net/11300937-Network-programmability-using-pox-controller.html (accessed on 20 January 2022).
39. Zhang, T.; Linguaglossa, L.; Giaccone, P.; Iannone, L.; Roberts, J. Performance benchmarking of state-of-the-art software switches for NFV. Comput. Netw. 2021, 188, 107861.
40. 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.
41. ETSI. Network Functions Virtualisation (NFV): Architectural Framework; Technical Report 002 V1.1.1; 2013. Available online: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf (accessed on 20 January 2022).
42. Binu, A.; Kumar, G.S. Virtualization techniques: A methodical review of XEN and KVM. In International Conference on Advances in Computing and Communications; Springer: Berlin/Heidelberg, Germany, 2011; pp. 399–410.
43. Wray, M.J.; Dalton, C.I. Network Virtualization. US Patent 8,223,770, 17 July 2012. Available online: https://uspto.report/patent/grant/8,223,770 (accessed on 20 January 2022).
44. Wang, M.; Cheng, B.; Wang, S.; Chen, J. Availability-and traffic-aware placement of parallelized SFC in data center networks. IEEE Trans. Netw. Serv. Manag. 2021, 18, 182–194.
45. Özdem, M.; Alkan, M. Subscriber aware dynamic service function chaining. Comput. Netw. 2021, 194, 108138.
46. Li, Y.; Chen, M. Software-defined network function virtualization: A survey. IEEE Access 2015, 3, 2542–2553.

47. Yousaf, F.Z.; Bredel, M.; Schaller, S.; Schneider, F. NFV and SDN—Key technology enablers for 5G networks. IEEE J. Sel. Areas Commun. 2017, 35, 2468–2478.

48. Kak, A. Towards 6G Through SDN and NFV-Based Solutions for Terrestrial and Non-Terrestrial Networks. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2021.

49. Qadri, Y.A.; Nauman, A.; Zikria, Y.B.; Vasilakos, A.V.; Kim, S.W. The future of healthcare internet of things: A survey of emerging technologies. IEEE Commun. Surv. Tutor. 2020, 22, 1121–1167.

50. Huang, M.; Liu, A.; Xiong, N.N.; Wang, T.; Vasilakos, A.V. An effective service-oriented networking management architecture for 5G-enabled internet of things. Comput. Netw. 2020, 173, 107208.

51. Morocho-Cayamcela, M.E.; Lee, H.; Lim, W. Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions. IEEE Access 2019, 7, 137184–137206.

52. Berardinelli, G.; Mahmood, N.H.; Rodriguez, I.; Mogensen, P. Beyond 5G wireless IRT for industry 4.0: Design principles and spectrum aspects. In Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.

53. Katz, M.; Matinmikko-Blue, M.; Latva-Aho, M. 6Genesis flagship program: Building the bridges towards 6G-enabled wireless smart society and ecosystem. In Proceedings of the 2018 IEEE 10th Latin-American Conference on Communications (LATINCOM), Guadalajara, Mexico, 14–16 November 2018; pp. 1–9.

54. Abdelwahab, S.; Hamdaoui, B.; Guizani, M.; Znati, T. Network function virtualization in 5G. IEEE Commun. Mag. 2016, 54, 84–91.

55. Huang, H.; Zeng, C.; Zhao, Y.; Min, G.; Zhu, Y.Y.; Miao, W.; Hu, J. Scalable Service Function Chain Orchestration in NFV-enabled Networks: A Federated Reinforcement Learning Approach. IEEE J. Sel. Areas Commun. 2021, 39, 2558–2571.

56. Sun, G.; Xu, Z.; Yu, H.; Chen, X.; Chang, V.; Vasilakos, A.V. Low-latency and resource-efficient service function chaining orchestration in network function virtualization. IEEE Internet Things J. 2019, 7, 5760–5772.

57. Ballani, H.; Costa, P.; Gkantsidis, C.; Grosvenor, M.P.; Karagiannis, T.; Koromilas, L.; O'Shea, G. Enabling end-host network functions. ACM SIGCOMM Comput. Commun. Rev. 2015, 45, 493–507.

58. 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, Monterey, CA, USA, 4 October 2015; pp. 121–136.

59. Kouchaksaraei, H.R.; Dierich, T.; Karl, H. Pishahang: Joint orchestration of network function chains and distributed cloud applications. In Proceedings of the 2018 4th IEEE Conference on

Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 25–29 June 2018; pp. 344–346.

60. Kouchaksaraei, H.R.; Karl, H. Service Function Chaining Across OpenStack and Kubernetes Domains. In Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems, Darmstadt, Germany, 24 June 2019; pp. 240–243.
61. Katsikas, G.P.; Enguehard, M.; Kuźniar, M.; Maguire Jr, G.Q.; Kostić, D. SNF: Synthesizing high performance NFV service chains. PeerJ Comput. Sci. 2016, 2, e98.
62. Carella, G.A.; Magedanz, T. Open baton: A framework for virtual network function management and orchestration for emerging software-based 5G networks. Newsletter 2015, 2016, 190.
63. 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.
64. Dab, B.; Fajjari, I.; Rohon, M.; Auboin, C.; Diquélou, A. An Efficient Traffic Steering for Cloud-Native Service Function Chaining. In Proceedings of the 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 24–27 February 2020; pp. 71–78.
65. 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 June 2015; pp. 1–13.
66. 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, Florianopolis, Brazil, 22 August 2016; pp. 511–524.
67. Kohler, E.; Morris, R.; Chen, B.; Jannotti, J.; Kaashoek, M.F. The Click modular router. ACM Trans. Comput. Syst. (TOCS) 2000, 18, 263–297.
68. Mafioletti, D.R.; Dominicini, C.K.; Martinello, M.; Ribeiro, R.M.; Villaca, R.d.S. PIaFFE: A Place-as-you-go In-network Framework for Flexible Embedding of VNFs. In Proceedings of the IEEE International Conference on Communications, Dublin, Ireland, 7–11 June 2020.
69. Katsikas, G.P.; Barbette, T.; Kostić, D.; Maguire, J.G.Q.; Steinert, R. Metron: High-performance NFV Service Chaining Even in the Presence of Blackboxes. ACM Trans. Comput. Syst. (TOCS) 2021, 38, 1–45.
70. 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.

71. Li, L.; Ota, K.; Dong, M. DeepNFV: A lightweight framework for intelligent edge network functions virtualization. IEEE Netw. 2018, 33, 136–141.

72. Cziva, R.; Jouet, S.; White, K.J.; Pezaros, D.P. Container-based network function virtualization for software-defined networks. In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), Larnaca, Cyprus, 6–9 July 2015; pp. 415–420.

73. Meng, Z.; Bi, J.; Wang, H.; Sun, C.; Hu, H. MicroNF: An efficient framework for enabling modularized service chains in NFV. IEEE J. Sel. Areas Commun. 2019, 37, 1851–1865.

74. 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 November 2016; pp. 203–216.

75. Eisenbud, D.E.; Yi, C.; Contavalli, C.; Smith, C.; Kononov, R.; Mann-Hielscher, E.; Cilingiroglu, A.; Cheyney, B.; Shang, W.; Hosein, J.D. Maglev: A fast and reliable software network load balancer. In Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), Santa Clara, CA, USA, 16–18 March 2016; pp. 523–535.

76. Yasukata, K.; Huici, F.; Maffione, V.; Lettieri, G.; Honda, M. HyperNF: Building a high performance, high utilization and fair NFV platform. In Proceedings of the 2017 Symposium on Cloud Computing, Santa Clara, CA, USA, 24 September 2017; pp. 157–169.

77. Lombardo, A.; Manzalini, A.; Schembra, G.; Faraci, G.; Rametta, C.; Riccobene, V. An open framework to enable NetFATE (Network Functions at the edge). In Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), London, UK, 13–17 April 2015; pp. 1–6.

78. 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.

79. Zhang, W.; Liu, G.; Zhang, W.; Shah, N.; Lopreiato, P.; Todeschi, G.; Ramakrishnan, K.; Wood, T. OpenNetVM: A platform for high performance network service chains. In Proceedings of the 2016 Workshop on Hot topics in Middleboxes and Network Function Virtualization, Florianopolis, Brazil, 22 August 2016; pp. 26–31.

80. Castanho, M.S.; Dominicini, C.K.; Villacça, R.S.; Martinello, M.; Ribeiro, R.M. Phantomsfc: A fully virtualized and agnostic service function chaining architecture. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 354–359.

81. Quinn, P.; Elzur, U.; Pignataro, C. Network Service Header (NSH). In RFC 8300; 2018; pp. 1–40. Available online: https://www.hjp.at/doc/rfc/rfc8300.html (accessed on 20 January 2022).

82. Zheng, C.; Lu, Q.; Li, J.; Liu, Q.; Fang, B. A flexible and efficient container-based nfv platform for middlebox networking. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, Pau, France, 9 April 2018; pp. 989–995.

83. Ghaznavi, M.; Jalalpour, E.; Wong, B.; Boutaba, R.; Mashtizadeh, A.J. Fault tolerant service function chaining. 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, USA, 30 July 2020; pp. 198–210.

84. Wang, L.; Mao, W.; Zhao, J.; Xu, Y. DDQP: A double deep Q-learning approach to online fault-tolerant SFC placement. IEEE Trans. Netw. Serv. Manag. 2021, 18, 118–132.

85. Kulkarni, S.G.; Liu, G.; Ramakrishnan, K.; Arumaithurai, M.; Wood, T.; Fu, X. REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization-Based Services. IEEE/ACM Trans. Netw. 2020, 28, 695–708.

86. Sherry, J.; Gao, P.X.; Basu, S.; Panda, A.; Krishnamurthy, A.; Maciocco, C.; Manesh, M.; Martins, J.; Ratnasamy, S.; Rizzo, L.; et al. Rollback-recovery for middleboxes. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, UK, 17 August 2015; pp. 227–240.

87. Hmaity, A.; Savi, M.; Musumeci, F.; Tornatore, M.; Pattavina, A. Virtual network function placement for resilient service chain provisioning. In Proceedings of the 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), Halmstad, Sweden, 13–15 September 2016; pp. 245–252.

88. Nguyen, H.B.; Dinh, N.T.; Oh, J.; Kim, Y. An Openflow-based Scheme for Service Chaining's High Availability in Cloud Network. In Proceedings of the International Conference on ICT Convergence, Jeju, Korea, 16–18 October 2019.