Journal of Network and Computer Applications 193 (2021) 103196

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

OpenPATH: Application aware high-performance software-defined switching framework

Prabhakar Krishnan [a,*], Subhasri Duttagupta [b], Rajkumar Buyya [c]

[a] Center for Cybersecurity Systems and Networks, Amrita Vishwa Vidyapeetham, Amritapuri-Campus, Kerala, India
[b] Dept. of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, Amritapuri-Campus, Kerala, India
[c] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Keywords:
SDN
NFV
Service function chaining
Network function parallelism
Intent-based networking
Software switching

ABSTRACT

Currently, core networking architecture is facing disruptive developments, due to the emergence of SDN for control, NFV for services and so on. SDN promises more versatility in routing and managing traffic flows, while NFV represents a large shift in how network functions and services are built, deployed, and managed. We present *OpenPATH* (aPplication Aware software-defined swiTcHing framework)—A software-defined switching framework for NFV processing and orchestration of Network Functions (NFs) and steering the flows through service chains. Inspired by the potential benefits of encapsulating the application logic into the SDN dataplane, OpenPATH is built on the concept of a modular dataplane, which consists of two layers - switching fabric layer to control packet forwarding; and switch management layer, which inspects the incoming packets, steers the flows through a sequence of NFs and determines the next forward/drop action. The application logic of the NFs can be introduced and pushed to the dataplane at runtime and the framework offers fast packet processing and I/O functionalities to support NF parallelism in the Service Function Chaining (SFC) scenarios. OpenPATH is a modular framework for software switches and offers flexibility for programming run time functions depending on the dynamic behavior of the network traffic and cyberattacks. The architecture components are not hard-coded or rigidly implementations in conventional switches/bridges and standard OpenFlow based SDN stacks. The design allows the vendors, operators, or developers to configure policies at run time and deploy custom logic and NF (also series of NFs) through software programs embedded in the switching fabric. While the basic concept is similar to some pioneering works in this area, OpenPATH does not sacrifice portability, performance, or security for programmability. The OpenPATH as a programmable switching platform takes a different approach to meet most of the requirements of application-aware and intent-based networking. OpenPATH helps administrators to quickly configure network security services using a rich set of standard APIs, with simplified flow tables. The evaluation shows that our design can leverage complex states in the data plane without overloading the SDN controller. Compared to conventional SDN methods, this provides much greater versatility and precision. The key findings indicate that OpenPATH achieves lower cost for scaling, higher overall throughput, and reductions in latency for real-world service chains.

1. Introduction

The Software-Defined-Networking (SDN) paradigm has transformed the way networks are managed, by using a logically structured control plane that can enforce carefully designed rules and programs that govern individual packet flows (Feamster et al., 2013) through all network groups and across modern virtualized datacenters. It changed the network vendor's approach to networking. Network virtualization (NFV) (Han et al., 2015a) paradigm which replaces specialized middlebox systems in the network, has emerged as the enabling technology to operate a myriad of infrastructure services as software applications (Network Functions/NFs) running on virtual machines (VMs) in Commercially available Off-The-Shelf (COTS) systems (H wang et al., 2014). The developments in softwarized networking infrastructures are aimed at improving network management, dataplane programmability, and the agility of networks. The software scheme also facilitates efficient traffic steering and orchestration of services and functions on the go. However, the limited interactions among current SDN controller(s) with

* Corresponding author.
E-mail address: kprabhakar@am.amrita.edu (P. Krishnan).

https://doi.org/10.1016/j.jnca.2021.103196
Received 30 October 2020; Received in revised form 12 July 2021; Accepted 11 August 2021
Available online 20 August 2021

NFV platform(s) restrict the extent to which new transformations can be realized (Sherry and Ratnasamy, 2012). The control plane, which is represented by the controller, imposes the traffic rules, yet has no control or knowledge of the running status/details from the hosting elements of NFV. SDN controller does not usually take advantage of the information that can be derived from individual packets such as inter-packet arrival times, sequence, protocols meta-data, session and connection state info of the applications. This overly rigid data and isolation of the controller limits the awareness of the control decisions and the versatility of the flow handling functions. Although SDN seeks to provide a simplistic management interface of the streamlined dataplane (s), the fact is that the functions and logic that exist within the dataplane switches of a real network is becoming increasingly complicated and fluid.

Modern Networks today are designed not only from simple elements such as routers and switches. It is also composed of appliances and middleboxes that perform various functions on network traffic such as proxy, cache, router, deep inspection, firewall, VPN, IDS, policies/configuration enforcing, traffic shaping, and monitoring for QoS (Alim et al., 2016). The ability to optimally steer and process packets pose hard problems that current SDN approaches are not likely to solve, because the current *match-action* rules for switches are not concise enough to handle complex policies and functionalities. It is well known that it is too expensive to depend on the control plane for packet processing frequently, instead of making the decisions at the flow-level. This cost-effective flow-based approach is exactly the general spirit of the SDN model. Furthermore, the conventional networking architectures don't help to dynamically track and modify the rules on flows with variable duration and time intervals (for example, a video stream that needs to be changed in time based on available capacity, elephant/ant/mice flows). SDN with its global view has been able to address all of these critical aspects in the dataplane, such as costs, management, and programmability, as well as additional problems such as multi-tenancy. This also brought in new vendors as it reduced the barriers to entry. Furthermore, only the forwarding devices are software-defined in current SDN implementations, such as OpenFlow (McKeownet al., 2008) and its variants, while the other data plane devices – middleboxes and network functions – tend to suffer from all of the above problems. Besides, these machines may often suffer from additional, more complicated issues, such as more complex tasks to process packets. Traditionally middlebox appliances, with their closed source software pre-installed in monolithic hardware systems are deployed as a vendor-built solution. These systems are expensive and vendor-locked in, causing barriers to scale and deliver customized services to the end-users. Apart from simple routing and forwarding functions, IP networks increasingly rely on a combination of advanced functions. The primary objective is to implement service-inferred forwarding for traffic traversing a given domain and distinguished by the collection of invoked Service Functions. Service-inferred forwarding is a policy-driven process, and policies which include dynamic parameters such as: Subscriber-aware, based on flow characteristics, designed to optimize network resource usage, and any combination of all these factors. To apply policy to traffic or to provide network services to it, the traffic must pass through a specific series of VNFs referred to as a service chain. Service function chaining (SFC) (Service function chaining general use cases) is a term that refers to the approaches for sequencing VNFs and enforcing the requirement that traffic must pass through the proper service chain. Service chaining has been applied manually in traditional networks (Joseph et al., 2008). NFV (European Telecommunications Standards Institute(ETSI), 2014)is proposed to reduce the management costs associated with hosting the SFC, and VNFs can be generated dynamically. Optimizing the location of dynamically generated VNFs and deciding the optimal route for VNFs to traverse a service chain are difficult problems. SDN enables the network to be viewed as a single global entity, and a centralized controller provides programmable forwarding rules that simplify service chaining compliance (Anwer et al., 2013; Fayazbakhsh et al., 2013; Qazi et al., 2013; Gember et al., 2013). A closer review of the services/functions of an SFC, reveals that most of the network functions do not overlap/share dependence and can operate parallelly. In the SFC shown in Fig. 1, *Monitor NF* retains statistics from the packets without altering their content. As the chain length becomes 3, this results in a theoretical reduction of the latency by 25 percent. While NFV enhanced scalability and availability, other problems such as minimal and autonomous control of NFs are not addressed. An exhaustive characterization of NFV applications exhibit similarity in processing blocks on the network traffic. For example, the majority of the networking elements/devices comprise of functions/-logic that inspects the headers of the packets, categorizes into types/class of flows, and executes the predefined logic on these flows. NFV vendors when they deploy their applications or services as VNFs provide a proprietary API for the customization and management of the policies and parameters. IETF has published several drafts (Service function chaining (sfc) architecture; Network service chaining problem statement; Network service header) as references for the practical implementation of SFC.

Network administrators support the "*softwarization*" trend because it enables flexibility and dynamic network configuration capabilities. However, implementing and deploying these strategies raises a number of realistic deployment issues. To fully exploit the advantages of NFV and SDN, the network must first carefully integrate a network orchestration strategy (Kim and Feamster, 2013), ensuring that the NFV integration does not complicate the network management environment excessively (i.e., a management challenge). For instance, in order to efficiently manage multiple network functions, a network administrator can create an orchestration strategy that generates a diverse set of associated network flow rules. It is critical to optimize the resulting flow orchestration.

NFV service instances are autonomous software instances, and as such, they have the potential to reduce network service efficiency when compared to older hardware-based solutions (i.e., a performance challenge). As we argue, no modern approach addresses both challenges comprehensively. Our approach is inspired by previous works on efficiently orchestrating virtualized network functions and middleboxes (e.g., CoMb (Sekar et al., 2012) and Bohatei (Fayaz et al., 2015)). Their main focus was on network service coordination, like controlling network flows, and not the most challenging part of managing network flow laws (i.e., at the data plane). Although Eswitch (Molnár et al., 2016), ClickOS (Martins et al., 2014) and NetVM (H wang et al., 2014) focussed on lowering management overhead by the use of advanced I/O handling algorithms, NFV systems continue to incur systematic performance overhead because of *traffic detouring*. This article investigates and tests a method for streamlining NFV flow processing by explicitly extending native services within the SDN data plane.

In NFV hosting platforms –network functions (NFs) are implemented as applications and deployed in one of the following models:

1. Physical machines (one NF/host), multiple hosts have to be utilized to deploy a SFC (a chain of NFs)
2. Virtual machines (one NF/VM) and multiple VMs (VNFs) can be deployed on a physical host.
3. Containers (one NF/container) and multiple containers (VNFs) can be deployed on a physical host.
4. Micro-services on high-end switches/routers/middleboxes (each NF as a micro-service) and the orchestration of the micro-services happen in the advanced hardware components such as network processors and SOCs.

We demonstrate another new model that implements each NF as a function (thread) and deploy these VNFs (implemented in software languages like C, P4) in a modular network operating system on high performance switching hardware platforms. The OpenPATH switches run NF/applications that are written in standard API/native API,

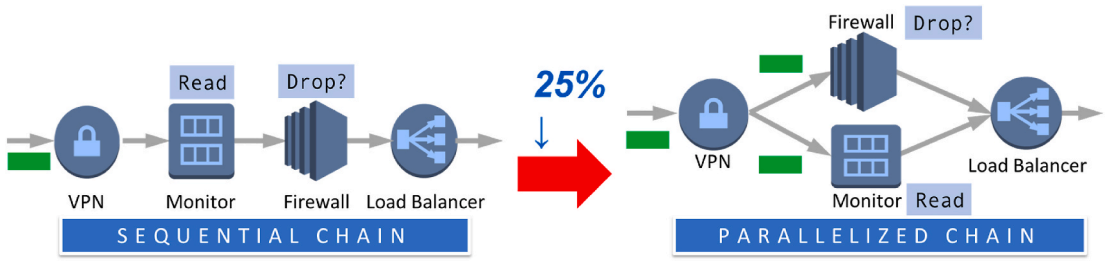

Fig. 1. Service function chaining implementation.

concurrently execute the logic of multiple NFs (chain of processes that share resources optimally) to execute them at the same data plane location (COTS node) and maintain complex stateful dataplane flows/sessions/connection tables and packet characteristics. So fundamentally, there is no notion of "virtual machines to NFs" (that usually run-on server class machine) in our platform, rather we have a mapping of "software processes to NFs" (that are running on switch network operating system) in OpenPATH.

In an attempt to address these challenges and questions, our research has developed a framework called *OpenPATH* (a high-performance SDNFV platform) for deploying diverse types of NFV applications and for NF decomposition into lower-level components to resolve the limitations in the previous approaches and boost the overall efficiency of the network. We focus on a particular component in the NFs service chain that is a known performance and security bottleneck, and we present algorithms to increase its overall turnaround time, efficiency, and resilience. We also leave traditional "*SDN + NFV*" as separate problems by running NFVs in conjunction with SDN to route packets between NFs, enabling the dataplane to handle these NFs (Sonchack et al., 2016; Qazi et al., 2013; Gember-Jacobson et al., 2014). OpenPATH handles the NFV orchestration and integration based on an entire topology view, which includes applications, computing, memory, and other networking resources (e.g. NF location). We show that the flexibility of OpenPATH in conjunction with its coordinated approach to cross-NF management and traffic orchestration allows for significant performance optimizations, e.g., offering 20–30% saving in CPU cycles and 2-4x increase in processing rate/bandwidth. We maintain the SDN controller and NFV MANO ("Management and Network Orchestration") in the integrated architecture. The central SDNFV Orchestrator represents the service chains as function graphs and distributes the sub-graphs (chain of NFs) across the nodes in the dataplane and at run time it manages packet steering through the NF chain. Each hardware node/machine hosts a *vSwitch*, which is managed by a software component called NF Manager. The ultimate objective of the OpenPATH architecture is to divide the network management functions and distribute control hierarchically. Finally, OpenPATH will effectively add application-awareness and stateful knowledge to the network, allowing Deep Packet Inspection (DPI) as micro-services for existing SDN deployments in ways that were not possible with legacy architecture.

OpenPATH framework consists of four major functional blocks - i) provides operators with a policy-specification scheme through northbound API to intuitively define concurrent or parallel intent. ii) SDNFV orchestrator smartly recognizes dependencies and composes policies into an optimal service graph. iii) A logically centralized SDN controller can dynamically install multiple NF application logic on the switches and orchestrate the NFV service chaining with the global view. iv) NF chain is executed in the dataplane switch(es), either implemented solely in software or on different hardware accelerator platforms (e.g., SmartNIC, NetFPGA). The OpenPATH implementation as a programmable switching platform, takes a different approach to meet most of the requirements of application-aware and intent-based networking, e.g., Modular pipeline, that runs arbitrary/custom logic and autonomous and every element of the switching architecture is programmable, not just NFs or Flow tables. The datapath supports both in-kernel for fast-switching in short-lived connections and entirely in-userspace which binds directly to network interfaces (bypassing the kernel) using DPDK, for heavy-duty packet data-processing applications, facilitates the flexible all programmable infrastructure for different use-cases, experiments, and open innovations.

Specifically, we make the following contributions to this research:

- We present a comprehensive overview of the NFV ecosystem, i.e., functions that are virtualized, virtualization techniques, and software acceleration. We articulated the guidelines for the design, development, and operation of a high-performance NFV ecosystem that can cope with heterogeneous and dynamic workloads efficiently. Based on the critical study, we derived an SDN-based datapath architecture that tackles the inter-NF dependencies and parallelism in NFV.
- We design OpenPATH, a high-performance software switching framework for NFV, using the standard Northbound API and OpenFlow protocol, along with algorithms to concurrently execute the logic of multiple NFs (service chain) to execute them at the same data plane location.
- Address the key requirements for switching architecture in future software-defined datacenters: (1) high-programmability and application awareness in the dataplane. (2) high throughput and packet rates. (3) efficient resource utilization (CPU, memory, network). (4) dynamically scalable and agile orchestration, packet-steering. and (5) security and fault-tolerance.
- By integrating major classes of NF/applications into dataplane and various switch fabric combinations, we validate our approach. We test OpenPATH programmable platform based on *in-kernel* Linux bridge, OVS, and *in-user* space implementations such as OVS-DPDK and measure it against other experimental software switching systems.
- Our research focuses on the performance of real-world network functions in a multi-host setting. We show that this framework not only makes development and administration easier, but it also improves the network performance: By executing two or more NFs at the same physical host/switch, exploiting NF parallelism and resource optimization, we are improving throughput by 2x and reducing latency by 50%.

2. Background and motivation

With the increased adoption of SDN architecture in large enterprise networks, SDN-based NFV platforms are widely evaluated now. Fig. 2 shows two potential approaches to implementing security services based on NFV. The network function and services (e.g., VPN or NAT) are usually deployed in middlebox appliances or COTS server machines. NFV operates with VMs and commodity servers, thus minimizing costs and simplifying deployment. A network service is rendered with a sequence of network-functions and steering the flows(packet) through that pipeline/chain (Service-function-chain (SFC). This chaining process could be customized to diverse policy settings on request. Multiple

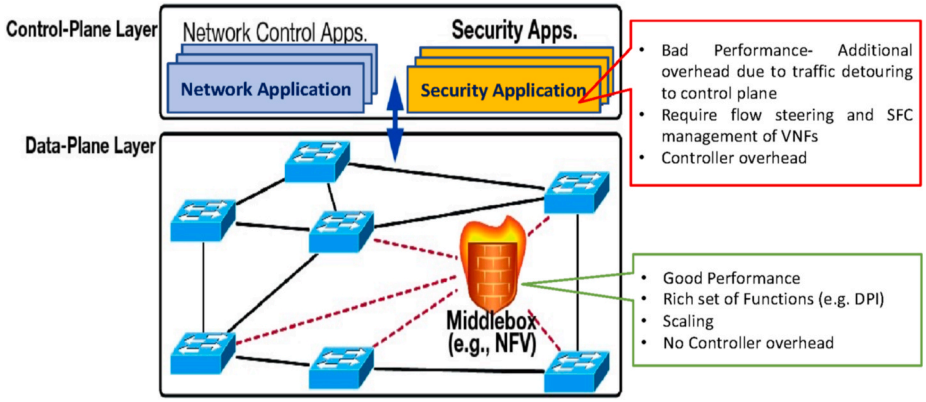

Fig. 2. NFV implementation in SDN infrastructure.

service chains in an NFV platform could re-use or leverage the pool of NFs for optimal resource usage. A single sequential trajectory path is not enough to orchestrate and form traffic in modern networks, as network traffic becomes more sophisticated. Since traffic needs to be handled by multiple NF elements, chokepoints occur in the network. Even the control plane of NFV has limited intelligence about the network. All these issues lead to siloed and isolated computing infrastructure to host inefficient NFV services. The NFs continue to change the packet state, and those changes are invisible to the control plane of the SDN. NFs can allow various types of state changes with operations on packets: modify payload/content/header fields ("e.g., *NAT box switches IP addresses or port numbers*"), drop packets ("e.g., *firewall blocks a flow*"), consuming and creating a new packet ("e.g., *layer 7 load balancer terminates the TCP client session, proxies and creates a new session with the correct server*"). The SDN controller thus is not equipped to track how the NFs in the center change packets and can lose the ability to monitor the flows. In recent years, high-performance DPI has been an active area of study, with plenty of algorithms for improving the DPI efficiency. Today every NF deploys its DPI engine, packets pass through a chain of DPI engines and use different matching pattern algorithms, although they all perform the same basic operation of matching the payload to a set of patterns. The drawbacks of such a situation are significant: first, packets go through an increasingly complex DPI phase, which means higher latency. Second, the Chain's slowest DPI engine is a bottleneck performance for the entire chain. Similarly, if an attack is exposed to one DPI engine (e.g., a denial-of-service attack), the entire chain is exposed to this attack in terms of overall performance. A multi-prong approach is therefore required in the *packet processing* architecture, to construct SFC that combine functions into sequences and also allow multiple service/functional blocks to do operations and examine each packet. Configuring these service chains within the data plane, however, poses an operational obstacle. To operate these service chains, the SDN data plane requires complicated flow rules, as shown in Fig. 3, which present flow-steering challenges in the process of service chaining.

We believe that as open programmable switching platforms and frameworks (Fang et al., 2018) evolve, it encourages creativity in the NF domain and operators can use the building blocks/API/language/compilers provided by the framework. Besides, the network infrastructure features can be expanded beyond these basic building blocks: an application may include a module extension code in the control/data plane. We demonstrated the use of the OpenPATH platform as a concrete deployment context for advancing this NFV domain: the broadband and cellular edge of a carrier network, as expressed in infrastructures of network operators.

2.1. Stateful programmable dataplane solutions

In recent progress, SDN standard bodies and researchers have begun to consider the possibility of offloading certain state-of-the-art packet inspection and control operations to the dataplane switches. This is primarily for reducing the signaling overhead of the switch/controller and the latency deficiencies caused by the *two-tiered* SDN programming model. More recently there was also a need to support API at the dataplane, a set of registries with statements that persist across several flow packets (i.e., flow states) and which was fulfilled by P4 (Bosshartet al., 2014). The Open Networking Foundation (ONF, standard body) has provided several suggestions to encourage designing the notion of "*state labels*" in OpenFlow. While maintaining of the state within the dataplane may seem to have been a step backward in respect of the SDN principles, initiatives such as (Hanet al., 2016) may be more conservative (hybrid) where the switches retain the flow-state, while the controller can continue to supervise/control handling stateful table within the dataplane switches. Moreover, it is the central controller that formalize through dataplane program (for example, P4) to determine how the switches can respond to packet-level events and adjust forwarding rules. OpenState (Bianchi et al., 2014) and FAST (Moshref et al., 2014) were

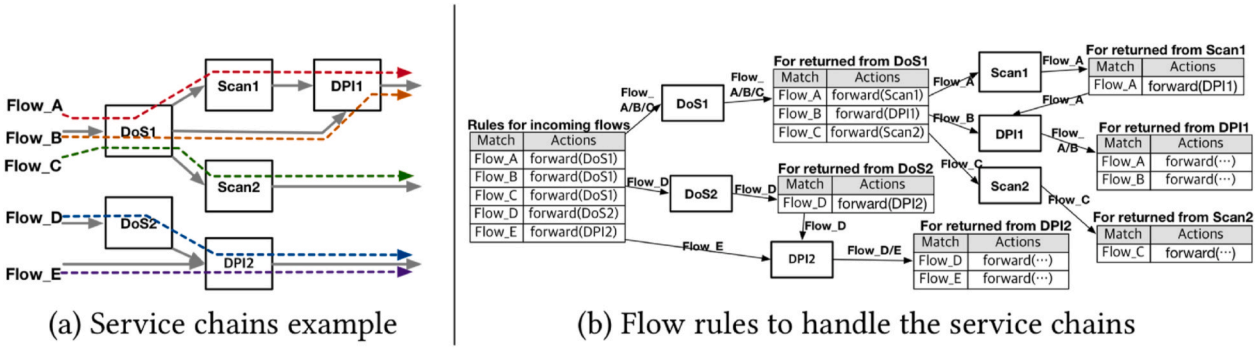

Fig. 3. Traffic Steering Challenge for complex service chains.

the first ideas that proposed the stateful dataplane and programming abstractions. Given the inherent conceptual coupling between language constructs and packet processing, we give an overview of the basic classification of programmable dataplane.

- **Data flow graph:** This represents computational logic as a line. The Vertex node representing elementary computing function (NF) and the Edge represents the path of data between two nodes. The Click modular software router (Kohler et al., 2000) pioneered this flow graph model. The packets traversing the graph through the vertices on which network functions are performed. ClickOS (Martins et al., 2014), "Vector Packet Processing (VPP) FD.io " (VPP FD.io, 2016), and the "Berkeley Extensible Software Switch (BESS)" (Han et al., 2015b) follow a similar design, with data units are represented as packet *vectors* rather than a packet.
- **match-action**: The model defines programs with a series of search table(s) arranged hierarchically as lookup tables. These lookup tables perform 'match' operation with specific header fields (tuple) and based on the result of the match - hit/miss, some 'action' is performed on the packet. OpenFlow is the most popular example of adopting this model for the forwarding dataplane and Open vSwitch is one widely used implementation of this highly programmable dataplane model.
- **Hybrid switch**: By representing hierarchical "match-action pipeline in a data flow graph and look-up/NF as *nodes* and *goto-table* action as *edges*", hybrid implementations are employing combined abstractions in a switching framework.

FastClick (Barbette et al., 2015) extends the Click Modular Router codebase (Kohler et al., 2000) by integrating optimized I/O mechanisms *DPDK* and *netmap* for the datapath by using acceleration strategies. Snabb (Paolino et al., 2015) proposed a composable modular architecture for faster switching in operating systems that hosts hypervisor VMs and demonstrated NFV services. Snabb realized their switch with a newly designed hypervisor called *vhost-user* that directly moves packets between processes bypassing the kernel. BESS (Han et al., 2015b) Berkeley Extensible Software Switch (formerly known as SoftNIC) is a modular framework for softwarized switch and network functions. It utilizes DPDK Poll-Mode Drivers (PMDs) to perform high-performance packet I/O with direct hardware NIC access. VPP (VPP FD.io, 2016) *Vector Packet Processing* is one of the well-designed routing stacks, that makes use of modern optimization approaches like packet batching and interleaving technique, where packets are received and grouped into super-frames as vectors. VALE (Rizzo and Lettieri, 2012) is another software switching solution that exploited the fast-I/O *netmap* mechanism for packet buffer movement across the processes. They also adopted batch computations and packet pre-fetching and designed to be used as the interconnect between VMs. The mSwitch (Honda et al., 2015) is an improved design of VALE targeted for SDN solutions, by using novel forwarding algorithm, kernel by-pass I/O, and no-CPU core bindings for the NF processes and at the same time scaling for a large number of ports. DPX (Park et al, 2019) proposed a framework that supported security service functions in the dataplane, and they are deployed through standard OpenFlow "match-action" rules. Their model avoids the decoding of packets by providing the operators with a simpler way to configure functions to the network. They enforced complex security policies through the implementation of "action clustering" which aggregate activities from various flows to a smaller ruleset. OpenFlow API is expanded by OFX (Sonchack et al., 2016) for deploying NFs in the dataplane, by installing an agent application that offloads some of the operations of a control plane for the switches. It supports the "Berkeley Packet Filter (BPF)" programming model in the Open vSwitch while retaining its high-performance benefits. SoftFlow (Jackson et al., 2016) retains the run-to-completion model based on Open vSwitch and runs arbitrary programs as *Free-Flow* activities in the datapath, but it is far more complex to configure policies/parameters to these programs. The *Open Packet Processor* (OPP) (Bianchi et al., 2016) is an attempt for combining configurable multiple hardware with limited dataplane capabilities. The OPP improves on the OpenState, even though it varies slightly from a design perspective, it provides SDN networks with a cutting-edge data plane using complete XFSM in place of the simpler OpenState version. The packet processing pipeline of a classic SDN architecture's stateless data plane is limited to the initial primitives in OpenFlow. SNAP (Arashloo et al., 2016) is a programming language used for stateful SDN switches and it allows flexibility for the program to make the decisions on traffic policy and flow-rules at run time, almost providing the power to function as a proxy controller. In recent years, researchers (Kaljic et al., 2019) have introduced applications that run on stateful dataplane, and all the stateful schemes from the literature share similar design notion at a very high level.

2.2. Switching and I/O management

The following are research aspects that have been widely discussed among the many challenges of the ETSI NFV standard (European Telecommunications Standards Institute(ETSI), 2014): NFV architectural design, NF parallelism performance, NF management, NF placements, and NF chaining. Many initiatives are considering how architectures in SDN and NFV need to grow such as ONF (ONF Solution Brief, 2014). Nevertheless, as yet there are few demonstrated solutions. Palkar et al. proposed E2 (Palkar et al., 2015) an NFV system that relies on a centralized SDN controller to handle the location, interconnection of services, and to dynamically scale a variety of NFs. The presence of SDN controllers is important for all the complex management of services and orchestration of the NFs. Mekky et al. (Mekky et al., 2014) proposed an application-aware SDN dataplane architecture that depends on functionalities of the OVS based dataplane. NetVM (H wang et al., 2014) employed shared memory for fast data copying between VMs, while it leveraged on ClickOS to deploy more lightweight function modules inside VM instances. To reduce overhead packet movement, Zhang et al. (Zhang et al., 2016b) developed SDNFV on top of NetVM and added features for parallelizing NF Chain on a single server and over multiple hosts. Slick (Anwer et al., 2013) presented a virtual middlebox elastic architecture that consisted of a central controller and a dataplane of lightweight COTS servers, to develop NFV applications. They focused on solving the NF placement problem and the southbound protocol is not defined as well. Flowtags (Fayazbakhsh et al., 2013) suggested network-wide policies for dynamic middlebox behavior, through explicit tagging in the packet headers. However, explicit tagging operation in high-bandwidth and heavy traffic switching platforms with complex service chains and flows is cumbersome and can cause semantic errors in a parallelized SFC process.

In data centers and enterprise networks, the service chain for the traffic is deployed with network functions and services (e.g. NAT, VPN, Firewall, IDS, encryption), which end users are normally unaware of their presence. In traditional networking settings, the network services are statically interconnected in a serial pipeline and predefined paths for the packets are configured. The introduction of NFV and SDN has significantly enabled SFC engineering (Qazi et al., 2013), exploiting the logically centralized control plane, providing flexibility in service chaining and forwarding plane programmability. OpenNF (Gember-Jacobson et al., 2014) proposed a closed NFV architecture in which network functions are deployed in the dataplane COTS and the management/orchestration is done at the customized control application. The architecture retains the spirit of SDN, with the standard SDN controller's oversight and individual vertical subsystems for coordinating with the NFs running in the OpenNF domain. NEWS (Mekky et al., 2017) is an expanded SDN architecture that uses a modified flow-table data structure "NF/App table" to manage packets and implements service chaining inside the data plane as a static sequence of modular functions. CoMbs (Sekar et al., 2012) proposed to consolidate multiple middlebox appliances (decomposing NFs) to one software on a COTS-based

dataplane node for improving the overall networking performance. Click Modular Router was advanced further by PacketShader I/O (Han et al., 2010) which exploits parallelism and multi-queue support of recent NICs, batching to improve overall performances of Click and ClickOS (Martins et al., 2014) is a runtime platform for virtual NFs. ClickOS offers NF I/O enhancements and reduced latency for packets at the same physical location that traverse multiple NFs. OpenBox (Bremler-Barr et al., 2016) presented an SDN architecture with distinct dataplane and deployed the NFs as OpenBox instances and controlled through the OpenBox protocol. The instruction cycles of the NFs are divided into independent blocks and executed as modules in the switches and allows for the reuse of software modules across network functions and enables faster adoption of packet processing hardware accelerators. OpenBox generalizes Click's modular approach to provide a network-wide platform for the development of modular NFs.

2.3. NF service function chaining (SFC)

Many research efforts (Bifulco and Rétvári, 2018) tackled the efficiency downside of software-based NF switching frameworks. In particular, software-based NFs that add substantial overheads and latencies, which may be unacceptable with the chain length for different applications operating under ultra-low response times. The literature (Fei et al., 2020) has suggested several groups of proposals that approach the problems in SFC from various perspectives. Some possible approaches for SFC latency optimization problems are i) accelerating NF cycles internally. ii) Instructions-Level Parallelism (ILP). iii) high-speed packet I/O. Some NFs do not share any dependence and can operate in parallel. So, we can perform acceleration vertically running parallelly multiple NFs on the same or copied set of data, on different dedicated CPU cores. ClickNP (Li et al., 2016) leveraged on offloading costly operations to SmartNICs, NetFPGA, and specialized hardware for horizontally accelerating NF to achieve overall performance on the NFV implementation. NetBricks (Panda et al., 2016) followed a similar approach as Click, but they designed the functions to run on a physical host, dedicating one core per VNF in the chain. ClickOS and NetVM (H wang et al., 2014; ntop; Zhang et al., 2016b) proposed solutions for the acceleration of packet delivery across the virtual machines and VNFs. These approaches proposed modular SDNFV architectures to distribute the network functions as "NF Blocks/Instances" onto the individual nodes and using efficient merging techniques, implemented an NFV platform to solve the SFC problem. NFVNice (Kulkarni et al., 2017) builds on OpenNetVM (Zhang et al., 2016a) DPDK platform, provides a userspace control, and efficient I/O management framework for NFs, that enables dynamic backpressure scheduling for NFV chains and demonstrates improved NF Throughput, Fairness, and CPU Utilization. ParaBox (Zhang et al., 2017) also attempts to explore NF parallelism in NFV. However, its NF parallelism detection remains preliminary and lacks a comprehensive analysis on NF action dependency. ParaBox has to provide different packet copies for NFs running in parallel which introduces large resource overhead. In comparison, we propose a comprehensive framework with three layers to enable NF parallelism and enhance NFV performance. NFs are implemented as a set of software modules in our system, at a finer granularity than the individual NFs. For example, one can build service by chaining the firewall module and DNAT module, making the composition of the service more versatile and effective. Our research proposes enriching the prior solutions, to address extensive processing in the data plane and tackles issues related to deep packet inspection NFs and meet the demands of other knowledge-based intelligent network functions. To tackle the management problems in the middlebox/Virtual NF world, prior efforts focus mainly on organizing network resources ("i.e., on the control plane"), only a few solutions attempted to solve the inherent problems in packet I/O processing ("i.e. on the data plane"). Although the popular initiatives such as OpenBox, BESS, and NFP are attempting to boost NFV platforms by designing Fast I/O handling, Parallelism, Efficient packet processing mechanisms, the infrastructures are still lacking optimal performance and resource utilization due to complex traffic switching/routing overhead.

Table 1 shows a summary of the taxonomy of software switching architectures. In comparison to the current literature, our work aims to advance our understanding of software switch efficiency, SFC, and to aid in the resolution of possible bottlenecks.

This research work discusses and examines one method for streamlining the NFV traffic directly in SDN architecture by expanding native functions in the dataplane. We argue that NFV requires a packet-processing software environment that addresses the problems such as NF placement, SFC scalability and also grow at run time, parallelism and load balancing across nodes, isolating the NF resources, monitor, and fault recovery and other run-time parameters. To this end, our research addressed some of these key issues targeting performance and presents a framework, which we call *OpenPATH*. From a functional point of view, our approach offers advantages such as (i) enabling developers, to focus on the core applications logic and leverage on external frameworks for common standard functions. (ii) simplifying the operator's tasks, by automation and shared management.

3. OpenPATH framework

OpenPATH is inspired by the desire to expand the data plane's

Table 1
Summary of Software switching/NFV Works.

Work	Architecture and Programming Model	Best Features for NFV SFC Placement, parallelism, security
BESS (Han et al., 2015b)	Modular, Data Flow Graph, Programmable NIC. Structured, C, Python, RTC, Pipelined	Forwarding between physical NICs. Not compatible with the advances in QEMU.
NetVM (H wang et al., 2014)	Modular Router, Data Flow Graph, Structured, C++, Run-To-Completion (RTC)	VNF chaining. Live NF migration.
OPP (Bianchi et al., 2016)	Self-Contained, Flow Graph, C, RTC	Full Router, VNF chaining, Supports live migration
NfvNice (Kulkarni et al., 2017)	Self-Contained, SDN Switch, Structured, C, callbacks, RTC	Stateless SDN deployments, Supports OpenFlow protocol, specifically designed for NFV management
OpenNetVM (Zhang et al., 2016a)	NetVM on DPDK, Shared memory, Netmap, Structured, C, NUMA-aware, polling mode	specifically designed for NFV management layer for flexible and efficient service chaining
E2 (Palkar et al., 2015)	Self-Contained, Virtual L2 SDN switch, Netmap, Structured, C, Berkeley Packet Filter (BPF), Pipelined	flexible packet processing pipelines, large number of ports. Supports dynamic scaling and orchestration through NFs (e.g., location, device interconnection).
OpenNF (Gember-Jacobson et al., 2014)	This retains two distinct vertical sub-systems, outside of the SDN remain NF. Modular, Netmap, Structured, C, callbacks, NUMA aware, polling mode	Their research focuses primarily on the sharing of the state and the transmitting problems that occur with replication and migration,
OpenPATH (This work)	Modular, SDN/NFV switch, DPDK, Shared Memory, Data Flow graph. Match-Action, C, Python, NUMA-aware, polling mode	Allows for easy reconfiguration of the NFV chains and fine-grained and/or dynamic control of the forwarding rules. OpenFlow based API. Inter-VMs (or container) abstractions are not Defined. Supports SFC Security, Dynamic Expansion, Acceleration.

functionality and to enhance the current SDN architecture. SDN's performance enhancement challenge has been solved by modifying the data plane in a variety of ways:

- *Stateful packet processing* - which is cognizant of the data plane's state. Numerous studies have shown that the stateless design of the OpenFlow switch does not adequately accommodate packet processing from stateful protocols (e.g., TCP, FTP). The most common way to achieve stateful packet processing is to introduce finite automata into the data plane.
- *Flow table structure* - additional data to support the treatment of data plane, energy usage, and QoS issues, among others.
- *Flow table Look-up method* - often occur in conjunction with systemic improvements to flow tables. In certain situations, the latest frameworks are built on enhancements to the OpenFlow flow table lookup mechanisms already in place.
- *Packet classification mechanisms* - permit classification of packets based on the headers of higher-layer protocols. As a result, sophisticated mechanisms such as DPI can be implemented.
- *Data plane architecture and abstraction* - have been suggested in a large number of studies as a solution to the existing OpenFlow standard's restricted versatility.

While theoretically feasible, *application logic offloading* presents significant practical difficulties. Since simple extensions will not address the programmability and flexibility issues associated with current data plane architectures, we advocate for the creation of an entirely new SDN data plane architecture that will provide a high degree of flexibility for future network evolution. OpenPATH is built in a manner that seeks to maintain as much as possible the original SDN concept (i.e., simple data plane) by designing its extensions as modular components of the SDN data plane. "*NFV Enablement Inside SDN Data Plane*" is an approach that aims to do network management in a controller, but distributing the NFs/applications across the dataplane, providing them with organic, reliable, and scalable support. We show that the OpenPATH system programmatically fuses the processing of multiple NFs and greatly reduces latency and improves performance.

3.1. Design challenges

In this section, we discuss the key challenges we encountered in designing the proposed switching framework.

Fast Packet I/O Processing: This topic brings us to a variety of design challenges and there is no need to reinvent the wheel in terms of efficiency and processing rate. Many current switches achieve good efficiency, and we adopt some of these techniques: packet-batching (Rizzo and Lettieri, 2012; Rizzo, 2012), lightweight packet and merging representation (Honda et al., 2015), and optimized memory copies (ntop; Sun et al., 2017). The Packet I/O is the critical function of the switching system and data movement, memory overhead, excessive protocol inspection operations on the packets impact the latencies and throughput, as systematically discussed by works (Han et al., 2010; VPP FD.io, 2016; Rizzo, 2012). To this end, OpenPATH implemented software acceleration mechanisms minimizing the datapath overhead in legacy hardware middlebox appliances. We performed a detailed analysis of the programmable dataplane for NFV's high-speed I/O (Gallenmüller et al., 2015; Lettieri et al., 2017) solutions. Some production grade solutions use SR-IOV (Dong et al., 2010) and softwarized switches such as OVS-DPDK (Open, 2019), Netmap (Rizzo, 2012), PF_RING ZC (ntop), and Snabb (Paolino et al., 2015). PF RING is a fast packet processing framework from NTOP corporation. PF_RING directly exposes the NIC ring buffer to applications in userspace, thus reducing the high overhead of the OS network stack. *Netmap* exposes userspace applications with a zero-copy network packet I/O for standard operating system environments. SR-IOV allows a physical NIC to export multiple virtual-interfaces to the hypervisor layer in the OS and VMs can access the network transparently like connecting to an interface. But, SR-IOV capabilities are vendor-dependent and can be connected directly to the guest OS running on VMs using PCI pass-through. Using Data Plane Development Kit (DPDK) gives an efficient user-space standard software I/O interface for the datapath to achieve throughput and ultra-low latency.

Usage Scenarios of the Software Switching framework: (See Fig. 4)

(a) Steering traffic between co-resident containers, VMs.
(b) Instantiating separate overlay networks over a shared datacenter fabric, using VXLAN headers.
(c) Steering packets between a sequence of network functions (NFs), by inspecting and modifying a Network Services Header (NSH).

SFC Traffic Steering scheme: There has been several proposals from the IETF working groups through RFCs and reference designs for the practical implementation of service functions chaining in SDN/NFV converged infrastructures. To enforce traffic steering at the network level in SDN managed infrastructures, the following major approaches are recommended:

- Network Overlay/Underlay: These methods give an efficient framework for steering packets across the SFC by re-using the standard protocol header from the packets. The shortcoming is that meta-data is not shareable and hence lacks the ability to re-classify or modify SFC (after the initial chaining sequence is assigned), thereby inhibiting the dynamic extension/modification of NFs at run time.
- Explicit tagging**: "**Network Service Header (NSH)" (Quinn and Elzur, 2016) draft proposed by IETF addresses the SFC encapsulation (RFC8300). NSH specifies the topology independent format of overlay-headers that are carried by the packets on the dataplane. The NSH metadata gives the ability to the middleboxes/services/NFs in the SFC to distinguish the services that operated on flows and steer the packets forward in the chain. FlowTags (Fayazbakhsh et al., 2013) allows SFC to identify tag-field-enhanced packets where NFs create and check the tagged fields when exchanging packets. Such approaches can encourage reclassification and steer the packets across any NF chain path, the caveat being the action-results cannot be shared as they operate in a different context. Moreover, these methods are more complex to manage and add steering overhead to SFC.

Policy design to describe service graphs: Network operators delegate unique roles to NFs in a service chain for sequential chaining of operation. Nevertheless, since we expect to promote NF parallelism in NFV that conventional NF positions specification approaches were unable to describe. To this end, OpenPATH enriches the semantic representation of the policies, and priorities in SFC.

Orchestrator design to manage service graphs: The standard sequential chain is converted into an efficient parallel service graph. Therefore, managing and orchestrating the NFs at run-time to fulfill the order and dependency between the NFs in the chain is complex. To this end, OpenPATH provides an interface to the operator to define the

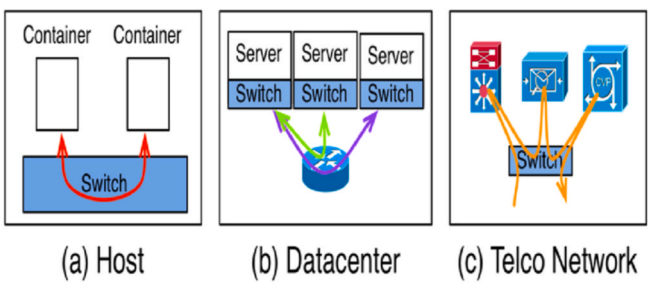

Fig. 4. Software switching use cases.

order, dependencies, priorities and at execution time the SDNFV orchestrator utilizes an automated analysis and placement algorithm to fulfill the SFC efficiently.

Optimizing the resource overhead: SFC parallelism demands both copy-free/copy-based functionalities from the underlying packet steering system, to execute the SFC graph. These copying operations and movement of packets may increase the load on the memory subsystem and communication fabric respectively, and it's challenging to orchestrate and optimize the usage of resources(Memory and Bandwidth). OpenPATH framework performs resource-overhead analysis considering the SDN dataplane capabilities and shared-memory, Fast I/O mechanisms to define multiple graphs for the same SFC and installs the flow-table *match-action* rules.

Designing efficient infrastructures: NFV parallelization poses some questions about the architecture of infrastructures. Next, the network should enable the lightweight copying of packets to reduce overhead copying and the network needs a merging module to combine processed packets in the final output from parallel NFs. Also, the packet movement and Fast I/O approaches in the NFV platforms (Martins et al., 2014; Fayazbakhsh et al., 2013; Sun et al., 2017; Zhang et al., 2016a) rely on a virtual centralized switch to arbitrate and steer the packets between NFs. In these approaches, due to a centralized switch as the hub, packet queuing/routing would become a bottleneck and compromise the efficiency. These problems may exacerbate during NFV parallelism, which may multiply (copies) the packets to be steered across two or more parallel paths in the SFC graph. In OpenPATH we exploited shared-memory abstraction for packet copying and packet-steering through the NF handler engine attached to each VNF in the chain, thus avoiding the complexities.

Placement of NF: Modern networks demand optimized infrastructure services and for deploying a NFV platform with a string of network-functions/services. The naive placement approach would be to divide the NFs equally across the available NFV nodes and due to multiple traversals of packets across the nodes, they incur additional overheads, throughput will also suffer in long NF chains that span multiple nodes. OpenPATH employs smart *affinity-based* placement for SFC which incurs lower latencies compared to traditional placement approaches.

Service Chaining: Sophisticated networks and datacenters deploy complicated and scale-out network service plans with a large number of services (long and complex SFC pipelines). In the legacy configuration (NFs on VMs) the processing rate and throughput decrease as we increase the NFs and number of disjoint paths in the chain. This bad performance is due to the non-optimal I/O, packets copying, across the stack and buffers, multiple contexts switching within the node and as the number of NFs increase the chokepoints, packet touring back and forth between the VMs through the switch. In OpenPATH architecture, with the NFs already embedded in the data plane switches nodes, we save all the above-mentioned overheads and the other resource optimization techniques such as zero-copy and smart load-balancing across multi-node SDNFV configuration have played a key role in achieving maximum practical sustained peak throughput even with the scaling of the chain.

Scalable Deployment: Scalability and elasticity are accomplished by using OpenFlow or other load balancing strategies to dynamically reconfigure the flow tables on the vSwitch nodes (dataplane). This means that both the number of written services and the traffic in our proposed solution scale-out. The forwarding algorithm should be designed, which scales up with port density.

Service Chain Management: The NFV platform orchestrator is provided with the global view and control to dynamically add network functions or services in the chain. To this end, OpenPATH's central SDNFV orchestrator is responsible for enabling the NFV/applications at run time through the NFV MANO and other management processes, e.g., "*load the connection restricting < blacklist IP> and restrict TCP connections to 80*."

In Summary, OpenPATH is designed with the following principles and requirements:

- *Performance*: service chain management should have minimal latency.
- *Centralization*: the control plane should offer a centralized overview of all network link states. NF order and dependencies in an SFC to be analyzed factoring in the correctness of SFC outcome.
- *Generalization*: supports all standard stateful connection-oriented protocols.
- *Extensible*: should not demand modifications to NFs code or application logic. provide compatibility to run legacy NFs. OpenPATH supports new features offered by advancing the SDN and NFV standards.
- *High Scalability*: SFC efficiency scales with the number of cores on the host nodes and supports complex placement graphs that span with the number of nodes in the chain.
- *Operational Complexity*: running the framework control operations should not incur too much overhead and no single point of failure and coordinating with the stateful dataplane does not cause bottlenecks in a broad set of use-cases.

3.2. Framework overview

OpenPATH framework builds upon the implementation of the stateful SDN architecture (Krishnan and Achuthan, 2019). The framework offers a high-performance programmable dataplane switch called *VARMAN* (Krishnan et al., 2019a), API/SDK to implement a suite of security analytics on the control plane and lightweight VNFs (Virtual Network Functions) on the data plane for monitoring and policy enforcement. The network-wide policies have to be converted into flow-rules and match-action control for every domain. The SDN controller, NFV MANO, and NF Manager within each domain co-operatively decide and interact over control channel to enforce the rules on traffic flows and packets. Following the principles of several popular SDN-based frameworks (McKeownet al., 2008; Pfaff et al., 2015; Open, 2019; Barbette et al., 2015; Paolino et al., 2015; Rizzo and Lettieri, 2012; Honda et al., 2015; Sonchack et al., 2016; Jackson et al., 2016; Bianchi et al., 2016; Arashloo et al., 2016; Fayazbakhsh et al., 2013; Bremler-Barr et al., 2016; Zhang, 2019), OpenPATH offers the rich set of declarative/abstraction interfaces, that are used by the control plane applications to communicate to the SDNFV dataplane switches in each domain how traffic should be processed and rules/actions on the flows. We also export a native SDK/library package, that allows NF/applications to utilize the optimized functions, in contrast to un-changed "legacy" applications/NFs that are portable and directly run on standard sockets and northbound API supported by the OS.

The OpenPATH framework has mainly two planes: management and smart dataplane as shown in Fig. 5.

- **Management & Control Plane -** SDNFV orchestrator, SDN controller, NFV MANO processes manage the traffic packets that flow through the series of NFs in the SFC. The NF Dependency inspection process analyses the chain of NFs, their instruction sets, and function blocks, fields of the packet and the corresponding operations (read/write), the dependencies between NFs. The results of the analysis will produce a dependency matrix to mark the parallel and sequential parts of the SFC. The head/tail NF of the chain and the intermediate distribute/merge functions are also setup. SDN controller manages the programming functionalities of the network devices and traffic and also hosts the NFV services. NFV-MANO is tasked to coordinate and manage the NFV services and network functions that are deployed in the vSwitch nodes.
- **SDNFV Data Plane -** OpenPATH switches are configured and interconnected in different topologies in the dataplane layer and connected to the management layer through the standard OpenFlow protocol and the controller, and through both native/open standard

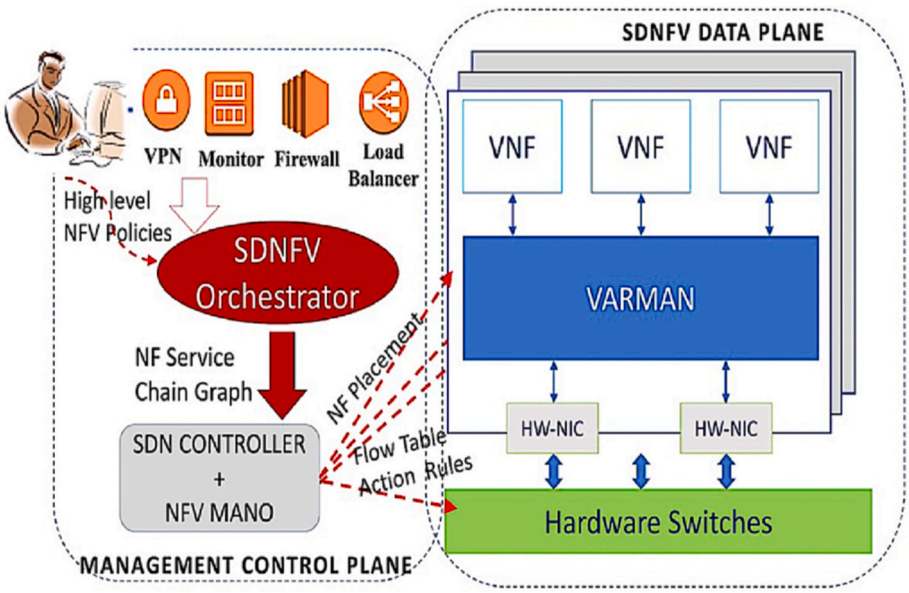

Fig. 5. High-level illustration of OpenPATH framework.

API to the applications at the top layer. The infrastructure for the data plane consists of a low-level packet processor, multi-physical-port network adapters, hardware whitebox switches, hypervisors and their internal virtual switches. The NFs and packet forwarding pipeline are designed for implementing in software/hardware/hybrid-model with SmartNICs. Every node in the SDN domain is supervised by an NF Manager, that communicates with the SDN controller. A dataplane instance vSwitch receives a service chain graph from the NFV-MANO controller and the NF-Manager on that vSwitch node, apply the flow-rules on the packets (match-action). A packet would go through the VNFs in the service chain deployed across the corresponding switches.

The OpenPATH framework is designed to provide efficient NFV management by employing a modular and extensible datapath and the standard OpenFlow communication protocol for orchestrating SFC and controlling the forwarding/steering the packets across the network. The Controller is responsible for launching and executing application logic in dataplane switches and enforcing the policies/actions on the matching flow/packets defined by the operator. The dataplane can be directly controllable (with delegated access levels by the SDN controller) using dataplane programming languages. (e.g., P4). Using the latest optimization algorithms, we present in this framework and the network-wide view of the packet processing tasks in the network available to the cross-plane SDNFV orchestrator, the controller fuses processing steps of multiple applications in such a way that eventual processing stays the same, but packets do not go through the same processing over and over again, thus improving the overall performance considerably. The complete network state awareness of SDN can be managed in a central controller when supporting NFs natively in the dataplane. An extended SDNFV cohesive architecture is designed that hosts both NFV and network management services. It means that there are no separate NF agents or new API, so they do not have separate control protocols unlike the solutions proposed by (Sonchack et al., 2016; Fayazbakhsh et al., 2013; Gember-Jacobson et al., 2014; Bremler-Barr et al., 2016). In OpenPATH, NFV integration and service function chaining technologies are hosted within the SDN architecture. The aim is to have only one common SDNFV orchestrator in the network who has the visibility of the networking infrastructure, manage the switching policies, and also control the deployed NFV services.

4. Application-aware SDNFV architecture

Fig. 6 shows the main system components in OpenPATH architecture to handle the applications and NFV services including a policy definition engine, SDN controller, NFV MANO, SDNFV orchestrator, and NF Manager. OpenPATH offers a versatile platform that can be used to speed up packet operations, an efficient packet I/O, and enables the application/NF to concentrate on programming the logic and run time policies. The framework provides an intuitive interface to the operators for representing SFC policies and intents for the orchestration of the applications/NFV services. The NF Chain represents a network function graph deployed in a series of VNF instances, distributed over one or more virtual switches (vSwitch) or server nodes. OpenPATH is designed on the concept of a "split data plane" comprising of a *switching fabric* for high-speed packet forwarding between ports; and *switch management*, that processes packets, runs application logic and policies and determines the path/route/next hops for them. This disjoint design and loose coupling inside the dataplane allow OpenPATH to deliver a smooth, high-performance fabric and capabilities for programming the packet processing logic, operators to deploy custom service chains.

The SDNFV orchestrator communicates with vSwitch nodes on the dataplane via an NF Manager (NFM) and provides flexible control functions from the management plane and VNFs/NF-Manager on the vSwitches are given independence for routing and steering the packets. The main aim of the framework is to provide a flexible open switching infrastructure and extended API. The SDN controller orchestrates the overall operation of a network domain, the NF Manager coordinates the operations/flow tables within each vSwitch node. Thus, the data plane operates like the softwarized switching fabric that underlays VNFs at each switch node. The standard interfaces such as sFlow, OpenFlow are used on the southbound control channel by all the management/control plane processes.

4.1. SDNFV dataplane

This paper builds on our earlier work VARMAN (Krishnan et al., 2019a; Krishnan et al., 2019b) a software switch as the dataplane in OpenPATH. We followed the de-facto method for network management in production grade dataplane by using Open vSwitch (OVS) (Pfaff et al., 2015) and OpenFlow (McKeownet al., 2008), although the Click (Kohler et al., 2000) styled interfaces are common in academic contexts. Fig. 6

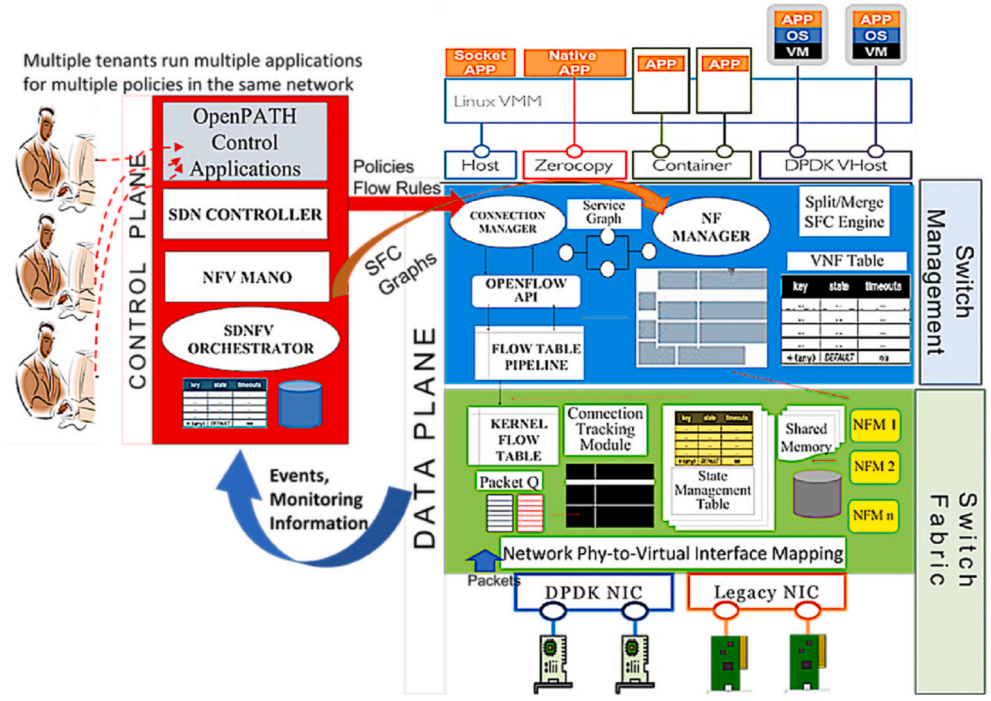

Fig. 6. Application-Aware NFV integrated SDNFV Architecture.

shows the OpenPATH overall architecture, the dataplane which consists of two internal layers that has divided the functions of a switching framework. The stateful dataplane makes the "*match-action:forward/drop*" decision for the majority of the normal flows and executes the pre-defined "network-function/application logic" inside the dataplane switching framework, without forwarding to the controller ("unless there are suspicious flows/exceptional conditions"). The design spans across both the planes of the SDN to cooperate in managing the stateful modules/NFs deployed in the dataplane and their corresponding control applications in the control plane. The **NF Manager** with the worker thread PacketQ coordinates the zero-copy packet movement operations by setting up reference-descriptors (Tx/Rx) and manage the I/O operations from the NIC to the VNF queues. The **PacketQ** poll-mode driver connects virtual or physical interfaces and the device modules. The OVS-DPDK abstraction establishes the queues/descriptor rings (VNF Qs) in the shared-memory regions for a direct copy of Rx/Tx packets from these NIC queues. OpenPATH PacketQ/DPDK handles the Ingressing packets and with the help of NF Manager steers it towards the head of the NF chain so it can also apply actions to the packet and with the unique forwarding mechanism built inside each VNF, the packet gets processed on the NFs in the chain until it reaches the tail NF and egresses the node. There is a delineation between a high-performance switching fabric (forwarding packets between ports) and switch management (deciding the egress ports). The switching network can be extended to large numbers of virtual ports and also provide parallel access to egress ports when multiple Tx VNFs forward packets independently in the chain. The dataplane is designed to support both software and hardware switches, to speed up the transfer of packets. Another advantage of OpenPATH is its inherent multi-tenancy support: Multiple network tenants may run their NFs in the same network, on the same data plane resources, while essentially isolating themselves. Two separate administrators, for example, could deploy two separate IPSs with different sets of rules. In OpenPATH, these two IPSs can be combined on the data plane to one NFM switch, while maintaining isolation on the control plane of the program. This greatly decreases ownership and running costs, since the data plane can be utilized even better. The *OpenPATH* switching fabric will employ all possible software acceleration techniques for packet I/O processing. The NF Manager can dynamically install new policies to process packet flows which save from overhead involving the control plane.

4.2. Management and control plane

Fig. 6 shows the framework that integrates SDN/NFV control and is supervised by the **SDNFV Orchestrator**.

SDN Controller is a critical management application, with various sub-systems and functional modules that run as worker threads. It implements OpenFlow (OF)standards for the communication channel to dataplane and processes the messages from OF switches and provides standard REST/JSON/native OpenPATH API for the applications. The controller runs the entire network as the brain of the framework. The controller which has the purview over the entire network gathers and maintains high-level state of the vSwitch flow-tables, OpenFlow pipeline installed in VNFs, network path characteristics, health, policies, and topology changes in the network. The controller then launches the NFs on vSwitch nodes and transmits the flow-rule tables to the NF Manager on those nodes for steering the flows through the NFs. The other key operations include: *Topology Collection* (using the LLDP protocol), *Flow Analyzer* and *OVSDB as the* datastore for the OpenFlow network.

NFV MANO is another critical management application that controls and orchestrates the NFV platform integrated inside the switch nodes. This centralized application schedules CPU and other resources for the VNF instances and places them on the vSwitch nodes in the data plane. The NFV MANO handles the NF parallelism, dependency analysis and placement and employs a smart affinity-based placement technique by co-hosting NFs that have shared operations/cycle, as this will decrease the unnecessary trips of packets flying back and forth between the NFs in the chain. The key function of the NFV MANO is to deploy NFV applications/programs as a service-chain on the data plane nodes and orchestrate them during the execution.

NFVx augments the data plane and aids in the execution of lightweight functions. On the switches, the OpenFlow software stack is enhanced to execute the applications/NFs and NFV functions. On receiving the service graph and the corresponding flow-tables, the NF Manager will install the NF functions in the SFC and set up the match-action OF pipeline in the VNFs. This module coordinates with the

global service-chain-aware SDN controller.

SDNFV Orchestrator- This is the most critical management and control software in the OpenPATH framework. It is a cross-functional application that communicates with all layers of the networking infrastructure. It coordinates the integration of NFV platform services with SDN infrastructural components and protocols. In multi-domain SD-Cloud deployments, this interfaces with the controllers and network management services across the domains. The Network Operators can communicate their policies through the templates/interface provided to the orchestrator. The flow rules represent the policies, which include the intents and info for the service chain. Inside this plane, the SLA manager converts these logical policies into a machine-readable form for further application processing. It communicates with the Topology Manager, which maintains the network topology. The Topology Manager tracks and collects network data, i.e., network utilization, switch TCAMs, NF position, and load on nodes. This runs a heuristic that determines the order of NFs in service chains and optimal placement for NFs. The orchestrator supervises the fulfillment of the SLA/QoS policies at run time and monitors the Service Function chaining performance, NFV platform utilization, vSwitch health and efficiency, faults and recovery processes, controller responses. It gathers data from all components across the network, maintains the topology map and configurations of the deployed NFV.

Control Applications- The management/control layer allows operators to run custom apps for policy creation/management/monitoring their NFs deployed in the OpenPATH infrastructure or match-action tables in JSON format over southbound and northbound API.

5. Network function service chaining

This section describes the strategies and techniques we followed to transform the design requirements for an efficient NFV platform under the SDN architecture. The complete workflow in realizing the SFC optimization in OpenPATH is illustrated in Fig. 7. OpenPATH consists of two major functional units in the workflow: the order-dependency analysis function as part of the SFC Control engine, and the mirror/merge functions on the software/vSwitch device.

5.1. Service graph description

The first challenge in the SFC process is to describe the intent of the operator in terms of Sequential chaining and priorities/preferences of the application in parallelizing certain functions in a particular order. Also, the description should be intuitive and expressive in the interface. OpenPATH provides an interface for the semantic representation of the policies, priorities in SFC. SDNFV Orchestrator is designed to define, construct and manage service graphs through the entire life cycle in execution. The standard sequential chain is converted into an efficient parallel service graph. Therefore, managing and orchestrating the NFs at run-time to fulfill the order and dependency between the NFs in the chain is complex. We provide an interface to the operator to define the order, dependencies, intents, priorities and at execution time the SDNFV orchestrator utilizes an automated analysis and placement algorithm to fulfill the SFC efficiently. The Policy Definition Templates are:

- **Order (NF#1, before/after, NF#2):** defined to express the order in which any two NF#s have to be executed.
- **Priority (NF1 > NF2):** defined to specify the Priority of one NF over another NF, in parallel execution.
- **Position (NF#, first/last):** defined to specify the position of an NF in the chain.

5.2. NF dependency analysis and parallelization

OpenPATH designs a hybrid SFC platform, in which both parallelism/sequential chaining is facilitated. OpenPATH recognizes parallelism opportunities through its dependency analysis feature. The NF Parallelism has some key requirements: i) When a set of NFs are parallelized, packet data are distributed (multiple copies) and once the parallel NFs are executed the output packets are merged for the processed packet traverses through the chain. ii) No source code or logic core changes should be done to the NF/application for parallelism or scaling the instances. We employed the dependency-analysis and NF Parallelism algorithm as illustrated in Fig. 8, to determine parallelizable NFs from all possible pairs in the SFC. This algorithm decides whether two NFs can be parallelized (with-copy/copy-free) or not. Several critical parameters are taken into account when performing NF dependency analysis: (1) The read and write operations performed by NFs on data packets. (2) Flow termination or packet drops (e.g., by a firewall) in an NF. (3) packet reconstruction (e.g., by a WAN optimizer|, NAT). (4) load balancer in front of several instances of the same NF. The consistency condition ensures that the packet coming out from the final node in the parallel parts must be similar to that packet if it had crossed the NFs in a conventional sequential manner. The algorithm depicted in Fig. 8 accepts a serial SFC ("designated as s") as input, with the help of two tables (Table 2, Table 3) and outputs the parallelism indicator for each pair of adjacent VNFs in the SFCs. In a service graph, a parallelism indicator is added to show whether two adjacent VNFs will operate in parallel.

$$P(m,m+1)=\begin{cases}1, & \text{if } f_m^s \text{ and } f_{m+1}^s \text{ can run in parallel}\\ 0, & \text{otherwise}\end{cases}$$

Algorithm: Parallel sub-SFC Placement on Single Node

Input: A SFC $s=(v_1,\ldots,v_M)$ with M VNFs
Output: Parallelism Indicator of every pair of adjacent VNFs $P(m,m+1),\ m\in\{1,M-1\}$;

foreach m ← 1 *to* M ← 1 **do**
  Fetch the operations of f_m^s and f_{m+1}^s according to Table 2
    // f_m^s is the m^{th} *VNF in the s*
  Lookup the parallelism capability (R) according to Table 3
  Switch R **do**
    case "Y"
    $P(m,m+1)$ =1; // f_m^s and f_{m+1}^s can run in parallel
    return;
    case "N"
    $P(m,m+1)$ =0; // f_m^s and f_{m+1}^s can't run in parallel
    return;
  end
end

Fig. 8. NF parallelism analysis.

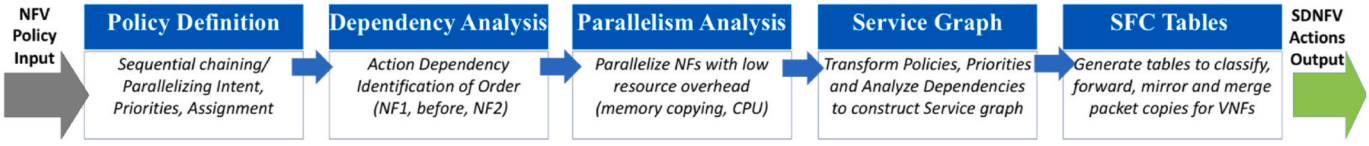

Fig. 7. Workflow of the SFC implementation.

Table 2
Operation table.

Virtual Network Function	Packet	Flow
Firewall	R	Y
Gateway		N
Intrusion Detection System (IDS)	R	N
Video Optimization Controller (VOC)	R	N
Netcache	R	N
Virtual Private Networking (VPN)	R/W	Y
Network Address Translation (NAT)	R/W	N
WAN Optimization and Bandwidth Shaper	R/W	N
Traffic Monitor	R	N

Packet Operations: R denotes Reading. W denotes Writing.
Flow Operations: Y denotes 'Can Modify', N denotes 'Cannot Modify'.

Table 3
Capabilities table.

VNF1 \ VNF2	Read Packet	Write Packet	Modify Flow
Read Packet	Y	Y	Y
Write Packet	N	Y	Y
Modify Flow	N	N	N

Column 1 corresponds to actions of first VNF.
Row 1 Corresponds to actions of the second VNF.
Y denotes "VNF1 and VNF2 Can run in Parallel".
N denotes "VNF1 and VNF2 Cannot run in Parallel".

5.3. Service graph construction

OpenPATH represents a service graph for a network-function/service sequence/chain. The graphs can represent vertex nodes and edges, the multi-path trajectories of the packets based on the content inspection. The edges leaving a vertex node determines the next hop for the packet. The NFV MANO constructs the final service graph (Fig. 9) from the outcome of the NF Dependency Analysis and operator's policy template. The resulting service graph and (possible sub-graphs) are transformed to OpenPATH flow-table rule definitions so that the SDN controller can install and manage the NFV processing at the dataplane switches. When a VNF is done processing a packet, the NF handler invokes - "Match: <result of the VNF>, Action: *Discard/Send to/Default*". The SDNFV Orchestrator/operator defines the "Default path" for every node, one of the edges (highlighted with bold lines), and "Default action" is usually taken by the VNFs.

In other dynamic SFC cases (Fig. 10), the NFV MANO can install match-actions rules for some VNFs that can take a custom action/execute a logic based on the result from the current VNF. This decision is taken on per-packet for the next path/edge to traverse. In an Enterprise Edge gateway use-case, the SFC may have a sequence of NFs for the packets to go through - e.g., IDS, Firewall, Sandbox, DoS scrubber, honeypot, and so on. The IDS will be coupled with Sandbox, so that when it detects anomalies and malicious flows the packets are detoured to Sandbox (with the *match: action* rule configured to call Sandbox VNF). With this scheme on the IDS VNF, only the suspicious packets are steered

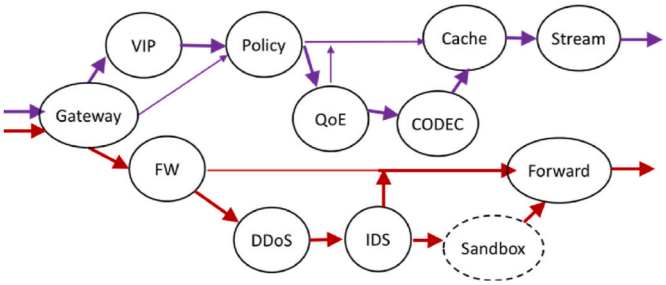

Fig. 9. Default service graph example.

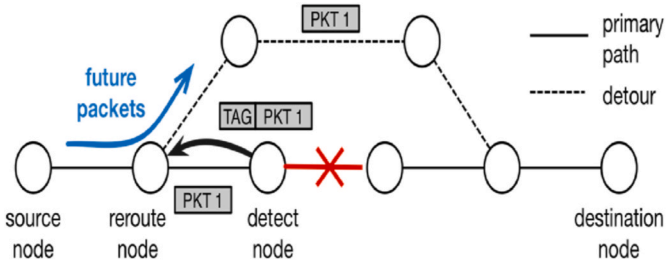

Fig. 10. Dynamically Changing Service graph.

through the Sandbox VNF path and the benign traffic flows through the default path. Also, through feedback messaging, the future packets of that flow would be re-routed through the path of the Sandbox NF or dropped at the Firewall itself, much ahead in the chain.

5.4. Fault Tolerance and smart placement

OpenPATH represents a service graph for a network-function/service sequence/chain. OpenPATH is optimized for virtualized SDN environments, enabling smooth failover, load balancing and fault-tolerance by careful route allocation. A significant limitation of generic topology-aware routing methods is their reliance on topology structures and their inability to handle temporary connection changes and failures. SDN enables the introduction of new networking abstractions and the simplification of network management. Three types of errors can be tolerated from a fault-tolerance standpoint: (1) VNF process failures or software application faults inside a vSwitch node which may lead to breaking/blocking the chain of NFs. (2) If a lower-level switch goes down, hosts can lose connectivity with the VNFs attached to it. We also consider connection faults on the data plane, because switch failures activate the same warnings and generate the same answers. (3) Management/control plane failure - control channel saturation or congested link disconnects leading to split SDN stack. Controller application failures or overloading. NFV-MANO failure, NF-Manager un-responsive or terminate to co-ordinate with the SDNFV orchestrator. The dataplane failure recovery capability is described by the VARMAN (Krishnan et al., 2019a) as the stateful OVS stack is derived completely from that work. The failure of any of the OpenPATH components such as NF-Manager or termination of any of the NF handler threads on the vSwitch node is considered as a break in the SFC path and hence backup-paths are calculated ahead of time so that the traffic interruption to switch the VNFs is minimal at run-time. The NF parallelization mechanism breaks up larger flows into smaller flows. Switches forward separate sub-flows through the associated operating VNFs in the sub-SFC. Each sub-SFC is allocated a distinct sub-flow. Fig. 11 shows method in which the main SFC is divided into sub-SFC graphs. When a fault happens in an operating sub-SFC, the state and metadata of the VNF is migrated in the back-up SFC nodes and will be activated for future flows. OpenPATH provides immediate protection against all single-failure scenarios, without requiring the controller to compute alternate routing or update flow tables. It is, nevertheless, possible to detect several or additional failures in a single network domain with the help of the controller's reactive measures. The serialization module can aggregate sub-SFC data. In parallelization, some information and state data often need to be synchronized. Based on usability criteria, the length and proportions of the VNFs may be modified in the SFC, Sub-SFC and back-up SFCs. The backup sub flow increases the overall availability of the SFC. The NFV MANO constructs the final service graph from the outcomes of the NF Dependency Analysis/Parallel and Placement algorithms and operator's policy template. The resulting service graph and (sub-graphs) are transformed to flow-table rule definitions.

After provisioning back-up sub-graphs and for parallel service graph, the sub-SFCs are mapped to vSwitch-nodes, considering affinity factors

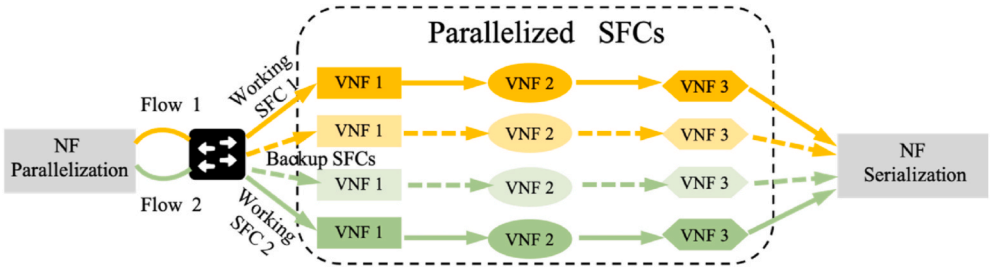

Fig. 11. SFC distribution.

for the best overall SFC processing time and also fail-safe factors for the reliable completion of the SFC. In summary, VNF failures and node availability characteristics are factored-in for deploying the serial/parallel sub-SFCs. The placement strategy involves splitting the paths of the working-SFCs and the corresponding backup-SFCs based on affinity levels namely: High-Affinity and Low-Affinity. This design reduces steering overhead between VNFs. When a fault or attack happens in the operating sub-SFC, the state and metadata of the VNF in back-up SFC nodes incur additional overhead.

High-Affinity: All the NFs within one SFC are deployed in one node. There is no internal overhead for steering flows between VNFs and during failures.

Low-Affinity: The active sub-SFCs and the associated back-up SFCs are not placed in the same node. There is additional synchronization overhead when failures occur.

Fig. 12 shows the algorithm for deploying a set of VNFs that form a given SFC. The general strategy is to first place the active sub-SFCs on the nodes, and then the associated back-up SFCs are deployed depending on the degree of affinity. In summary, to recover from failures in the Management/Control Plane, OpenPATH framework uses '*active-active*' fail-safe method with two machines actively in sync to take over when one of these critical components fail. For the failures of any of the components (network partition or chain broken), the backup-path approach will be followed.

5.5. SFC packet forwarding

The SDNFV Orchestrator coordinates with the NFV-MANO and SDN Controller to create flow-tables for each of the VNFs in the SFC. Besides, the SDN controller can set rules to accommodate a broad category of flows to reduce the flow table size. The categorized packets of those flows are steered through the chain of NFs. We use Flow-based traffic management, and this method is not based on header fields, proxy, and other additional tags. Rather, it employs an independent forwarding scheme leveraging the OpenFlow, from overlaying and preserving packets unchanged/transparent in the SFC. Our scheme uses the combination of most recent updated OpenFlow 1.5.1, forwarding policies defined in Yang format, in the SFC friendly OpenDayLight (ODL) controller. A global forwarding table for managing the flows is created by NFV MANO and transmitted to the NF Manager. Fig. 13 shows the path mapping technique to steer the packets/flows through different paths/NF pipeline.

Algorithm: Affinity Based Parallelized SFC Placement

Input: A set of parallelized SFC S = $(v_1, \ldots, v_M)$
Output: Set of SFC accepted nodes to deploy: $\mathbf{S}^{A}$
while $\boldsymbol{S}$ is not fully deployed **do**
  Fetch undeployed sub-SFC $s \in \mathbf{S}$
  foreach *undeployed* f_m^s with *High Affinity in s* **do**
    $\mathbf{C}^{f} \leftarrow$ Get the first x working nodes in $\boldsymbol{P}_{f_m^s}^{W}$;
    foreach $n \in \mathbf{N}$ with *Low Affinity* **do**
      $\mathbf{C}^{n} \leftarrow$ Get the y backup nodes **in** $\boldsymbol{P}_{f_m^s}^{B}$;
      if *match* $(\mathbf{C}^{f}, \mathbf{C}^{n}) = true$ **then**
        $R(n) \leftarrow R(n) - r(f)$
        Remove n from $\boldsymbol{P}_{f_m^s}^{W}$;
        Finish deploying f_m^s ;
    end
  end
  if s is fully deployed **then** $\mathbf{S}^{A} \leftarrow \mathbf{S}^{A} \cup \{s\}$
  Update $\mathbf{S}^{A}$
end

Fig. 12. SFC placement strategy.

The NF Manager divides the global table and applies individual VNF flow-table rules. **NFV MANO** distributes the packet forwarding task across the dataplane switch nodes and configures VNFs to directly/parallelly forward packets to the next VNF in the chain without any coordination. An **NF handler** function is plugged into each VNF forwarding path, to participate in the packet steering process, and as this is transparently done by the OpenPATH so VNFs don't require code changes. The NF Manager in the vSwitch is the communicating agent at run time and manages a *forwarding table (FT),* that corresponds to the sub-graph view of the global service graph. This node manager installs the most updated packet-steering rules in each NF handler in the VNF chain. **PacketQ** module is a poll-mode driver that manages the Ingress/Egress queues and the physical network interface adapters in the vSwitch node. This module receives the packets from the virtual-NIC and performs lookup with *5-tuple* matching on the flow-table and forwards to the first VNF in the service-graph. In a typical SFC scenario, the VNF at the head of the chain receives the packet, runs the logic on the packet and NF handler will pass on the packet to the next VNF in the sequence or make copies to forward to the next VNFs in parallel. This packet processing and steering continue until the packets reach the tail of the chain where they are merged for output Egress processing. Network topologies have been configured in such a way that all VNFs that are connected to a common edge switch are on their own subnet. NFV-MANO will have full knowledge of the interconnecting network topology. Other traffic is redirected and therefore must go through a lower-layer VNF. The lower- and upper-layer switches in any given pod in a fat tree are set up to perform forwarding/filtering functions. As a result, all upper-level switches in the pod will contain a route to the destination subnet, and each one underneath them will too. For all other inter-pod traffic, the *default/0* prefix has a secondary path TAG (PT) (the least-significant byte of the destination IP address). We use the Service Chain (SCID values) as a source of probabilistic entropy; it will distribute network traffic equally across the outgoing links. Also, the subsequent packets will travel the same direction, so there will be no packet reordering. Forwarding state can be incrementally built into the switches.

5.6. Flow table management

The NFV MANO constructs the final service graph based on the NFV dependency analysis and operator's policy templates. After the service-graph is generated (possible sub-graphs) at the controller, they are

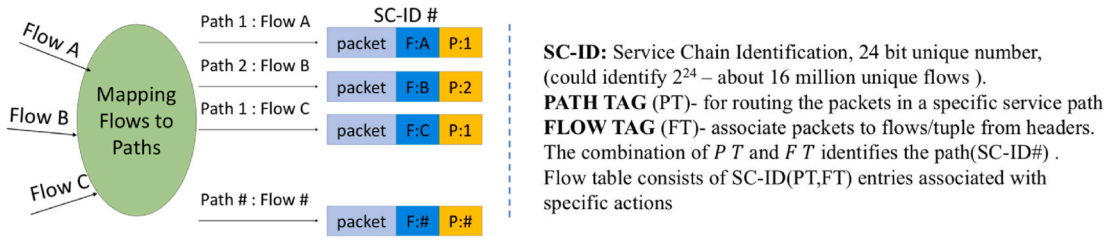

Fig. 13. Packet Paths mapping Methods.

transformed to flow rule definitions. The flow tables at the vSwitch nodes are managed by the NF Manager. We need to account for saving flow table capacity of SDN devices through aggregating micro-flow for high scalability. The flow-table installed by the SDN controller to each vSwitch follows the standard OpenFlow but is expanded to support our SFC steering scheme, by re-purposing some of the fields in the packet headers: i) some fields are added - SFC ID, PATH ID, TAG, SERVICE ID. ii) match-action rule has a new "*flag*" that indicates if the current packet is split into multiple copies to traverse through parallel paths. For parallel action paths, multiple VNFs will be invoked next for the same packet. iii) match-action rule also has a field for the tag that carries metadata used by NF handlers in the VNF for run-time monitoring and statistics collected by the NF manager. Fig. 14 shows the flow table management.

5.7. Network status synchronization and SDNFV Co-ordination

In OpenPATH, we do not directly address external state coordination across NF replicas; rather, we concentrate on how the state handled at each layer influences flow routing decisions made across both the Control Plane and Data Plane in complex service chains. The hierarchical control system makes decisions based on the internal and external states that each layer maintains. OpenPATH forwarding decisions are not just based on flow rules, but on flow states, which the switch retains and modifies in response to packets and timeout events. The concern is, therefore, whether this new state should be reported and reapplied during reconciliation. Flow state synchronization and reconciliation are unnecessary for OpenPATH operations to be reliable. Furthermore, the switch's forwarding action is independent of the flow condition of every other switch. Since flow-based networking makes more frequent use of the control plane than conventional networking, it has higher overheads. Its dependence on the control plane entails inherent overheads: the bandwidth and latency associated with communication between a switch and the central controller. The stateful data plane of OpenPATH appears to contradict the classic architecture, which centralizes the stateful decisions with the controller, removing the need for devices to implement complex software to manage state distribution. Indeed, modern SDN platforms (such as our research work OpenPATH) use a distributed flow rule database to synchronize the switches. The latency and overhead associated with inter-switch communication on the data plane fabric is much lesser in our architecture compared to traditional SDN architecture with stateful dataplane, where the decisions have to be routed across the much slower control channel and centralized switches as in the case of VM based NFV platforms.

The OpenPATH network describes the overall service as a hierarchical network of network functions that are distributed over one or more vSwitches (VNFs). All nodes keep track of NF state, and make decisions based on that. "NF Managers" allow for the development of load balancing data and reaction to errors/overload. Additionally, the SDN Controller contains the flow table, which may be maintained by the NFs. We coordinate with each node's SDNFV Orchestrator to decide flow rules but allow local table rules to be updated as appropriate. We organize the control and system administration into a three-layered structure. OpenPATH's mission is to consolidate state visibility, while distributing information. state inside the VNF is contained as a node specific. Transient states include things like the application and data caches, i.e., state that is needed for the NF, but does not impact on the rest of the network and the SDN controller. The node-specific state presents the current utilization of the vSwitch (OVS), as well as resource utilization on the vSwitch. There is a state from the outside that determines how packets are handled. Internal and external state relevant for flows are stored in the VNFs. This state has to be consistent across all or some of the NFs, particularly if they are handling network flows. When an NF has to take notice of an external state, we use the NF Manager to notify other NFs. In multi-node SFC scenarios, as the NF state is distributed across multiple nodes, the SDN controller initializes the tables across the nodes and periodically synchronizes the state tables. This problem and the management overhead are not unique to OpenPATH, and we solve this issue efficiently through the native design choice of "Statefulness" in the data plane. Since flow-based networking makes more frequent use of the control plane than conventional networking, it has higher overheads. Its dependence on the control plane

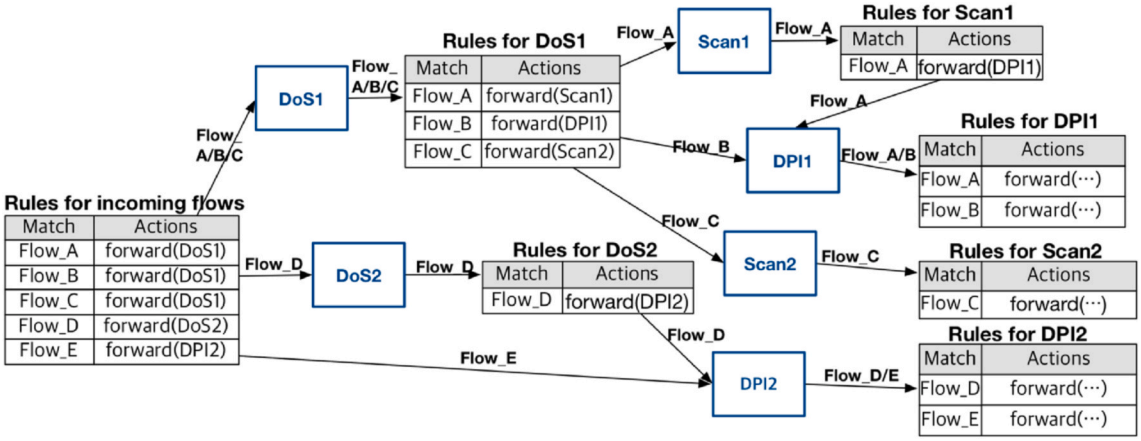

Fig. 14. Flow Table generated for Complex service chain.

entails inherent overheads: the bandwidth and latency associated with communication between a switch and the central controller. OpenPATH has defined messaging interface over OpenFlow between all the components in the SDNFV ecosystem - SDN controller, NF Manager, NF handler, VNFs and the NFV MANO for the following actions: (i)Specify processing graph and block configuration. (ii)Events. (iii) Load information. (iv) Isolation between NFs. (v) Network-wide view. (vi) Automatic scaling, provisioning, placement, and steering. (vii) NF/application match-action definitions. Each command message applies to flows matching some criteria, F, for a Switch Node N and Service S (one or more flows or wild card) in the SFC. Some example commands are:" *SkipMe (F, N), RequestMe (F, N), ChangeDefault (F, N, S) and Message (N, X, Y)* for sending JSON format (key/value) from VNF N to NF Manager." The control communication through Messages that happen on the southbound channel is encapsulated in special message "*OpenFlow VENDOR*" and handled by the SDN Controller.

OpenPATH provides a rich set of inter-VNF messaging abstraction API- that facilitates communication between VNFs for cooperative SFC processing. Some example API actions: "(i) reconstruct TCP byte streams (to avoid the redundant overhead at each NF), (ii) per-packet metadata tags that accompany the packet even across NF boundaries, and (iii) inter-NF signals (e.g., a notification to block traffic from an IDS NF to a firewall NF)." These inter-NF messaging are useful for SFC optimization and facilitate designing fine-grained NFs modules that can be manipulated at run-time. OpenPATH extends a control API exposed to NF manager: i) boot-up/spin-down VNFs instance. ii) for creating/destroying virtual interfaces to VNFs. iii) merging/splitting multiple VNFs in a chain inside a vSwitch host or between nodes. iv) exchanging alerts/events between NFV MANO such as faulty/rogue/overloading nodes, that could trigger exceptions to respond/remediate/recover. The cross-layer control interface is provided over JSON-like structured messages and three programmable APIs - Python/C/Rest and scriptable configuration language - CLI and every parameter in the SDNFV and SFC environment is run-time configurable.

6. Implementation

Having defined a set of design principles, this OpenPATH implementation is based on a high-performance, programmable stateful dataplane SDN stack and an application aware SDNFV framework. The hierarchical framework reference implementation is illustrated in Fig. 15. OpenPATH is an all-softwarized switching framework to integrate the NFV platform with the SDN architecture, at the distributed dataplane switches (programmable white boxes or server nodes) and sharing/coordinating the control and service chaining with the SDN controller. A dataplane instance/node of OpenPATH is called as **vSwitch,** which is installable fully as software on a high-core density server host machine or deployed in a hybrid software/offloaded-hardware-NIC model. The software switching and management components are developed in the Open vSwitch (OVS) opensource code base and we augmented the core stack with native VNF modules for supporting NFV services.

6.1. Stateful dataplane

Software switch solutions following the "match/action" approach such as OvS-DPDK (Open, 2019) and P4 (Osiński et al., 2020) (Bosshartet al., 2014), employ high-performance lookup algorithms for matching (packet header metadata) and applying the respective actions. The proposed design of the dataplane is realized by re-architecting the stock OVS code. We ground our discussion in our experience implementing OvS-DPDK (Open, 2019), a faster variant of Open vSwitch (Fayaz et al., 2015). It relocates Open vSwitch's data plane into user space and uses DPDK poll-mode drivers to send packets, fully eliminating the overhead associated with kernel and interrupt handlers.

The key components of the data plane are:

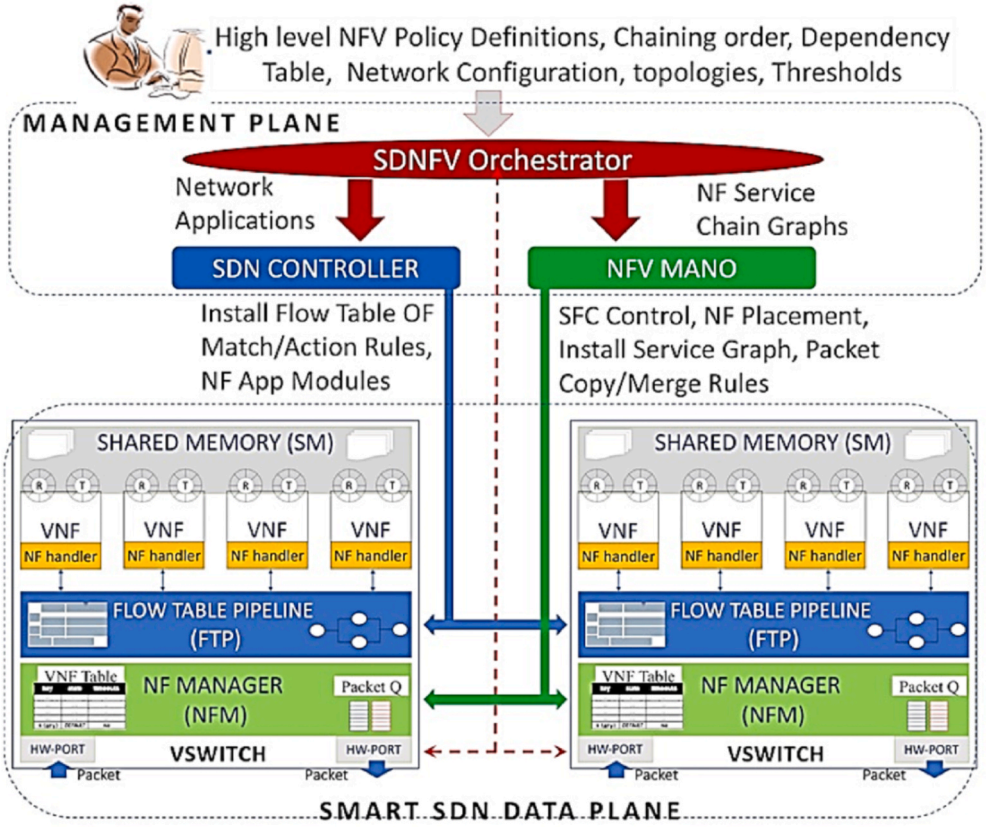

Fig. 15. OpenPATH switching framework Implementation.

- **vSwitch-** These are called nodes that host the VNFs/applications deployed by the NFV-MANO. The nodes can be Whitebox programmable switches that can boot Custom-Linux kernel with Open vSwitch and other modules, that exports the ports as software interfaces to the kernel. The nodes can also be server-grade machines with multi-core and high-memory configuration (COTS) which can host multiple NICs for emulating a multi-port software switch. Multiple vSwitch nodes can be interconnected through another switch to build a scale-out SDNFV cluster platform. The node is built with dual Xeon X5650 @ 2.67 GHz CPUs (2 × 12 cores), NVidia GPGPU, Intel 82599EB 10G Dual-Port NIC (Both are connected to an experimental LAN for data-plane traffic), 48 GB (8 GB for huge pages), Ubuntu 16.10 (kernel 3.5, DPDK, QEMU). Each machine has one 1 GbE Intel NIC for control and management traffic.
- **Data Traffic Switch:** NFV Nodes connect to a Cisco SX350X-24F Managed Switch | 24 Ports 10 Gigabit Ethernet (GbE).
- **OpenFlow Switch** (running OpenPATH Switch software): Whitebox high-end switch for control and management traffic (its private vendor who we had to anonymize since we did not get a permission to use the vendor's name, running OpenWRT (OpenWrt Wiki, 2017)) We also experimented with Pica8 (Pica8, 2013) P-3290 (with a modified Indigo OS (Indigo)). For applications, a Stateful API library is exported. We used the OFPT EXPERIMENTER extension to encapsulate OpenPATH control messages in the OpenFlow protocol packets.
- ***NF Manager*** is the core system process that runs on every vSwitch in the dataplane. This is a kernel process that runs with high priority and handles packet movement between the VNFs and maintains the flow-tables.
- ***Upcall*-handler** as part of the vSwitch node is a key process that interfaces NF Manager with the SDN Controller and so it maintains a separate queue for OpenFlow pipeline messages to-Controller/from-Controller in a separate memory space and forwards the new-flows/packets/unknown-flows over a safe channel to the SDN controller. We are repurposing OFPT FLOW MOD messages from the OpenFlow protocol to define the forwarding behaviors across the VNFs in a service chain.

The opportunity for combining *match-action* on flows with "state labels" on the chain of tables is an integral aspect of our scheme (See Fig. 16). A packet that enters the switch is matched in flow-table(s) and based on the lookup result; the *action* may be - *forward/modify/drop* the packet. We extended the OpenFlow v1.5 +, with the entries by the corresponding headers describing the events, sessions, and connection state (metadata). We added supplementary fields - "State labels, Flow ID, ID, Path ID" that would be employed by the match-action right from the head VNF in the service chain and subsequent VNF tables and until the end of the SFC. As most of the NFV applications are stateful type, (i. e., they maintain connection or transient session data across multiple instantiations of the network function) and the infrastructure is expected to provision a high-speed memory for storage of protocol, metadata, headers, options, and other flags. Hence, the OpenPATH framework defines special data structures *synapses* (maintained by the connection tracking module) that are provided by the shared-memory infrastructure, for storing and retrieving state information. A high-speed "key-value" database is provisioned in each vSwitch node, and it is widely utilized by many stateful network applications such as IDS, FIREWALL, WEB PROXY.

While building the stateful extension to Open vSwitch we had to address flow-table cache preservation strategy. As OVS is designed on the premise that the flow-table rules seldom vary relative to the rate of arrival of packets, it follows a conservative caching scheme, which is important for its high efficiency. In comparison, after each change of state, switches may have to rebuild cache entries, removing any caching advantages. The NF Managers integrated into the switches are intended for data plane state maintenance and communication between the controller and the switches through OpenPATH southbound APIs. The controller will initialize an application contained within a switch. The controller can add, alter, and delete table entries in vSwitch nodes proactively during runtime. Additionally, the controller may configure switch properties, such as the interval between state reports. The controller exposes north bound APIs to allow applications to change state tables in order to enforce stateful processing logic. The data plane maintains state and updates it in response to incoming packets or internal/external events. The state information can be uploaded to the controller, allowing the controller to maintain the network's global state information. The controller may specify the frequency at which switches send update messages based on the application's requirements. For instance, switches can be programmed to communicate with the controller on a periodic basis rather than sending a single message for each update. The controller will regularly obtain *state* reports from the data plane and update the local record in order to maintain state synchronization. The controller's internal state can be used to recover from failures. The controller will replicate the *state* of a failed switch and redirect flows appropriately. Due to the fact that the controller does not have the most recent state when a switch fails, the state to be configured in the replacement switch may be inconsistent with the most recent state in the failed switch. Take note that in the OpenPATH architecture, the controller continues to serve as the centralized intelligence. The controller continues to perform traditional functions such as connection discovery, topology detection, and forwarding. We incorporate the OpenPATH API into the architecture and store state in the data plane to improve the efficiency and scalability of stateful applications.

To address the challenges of implementing agile data plane with stateful operations and managing the trade-off between the resource constraints in the switches and optimizing the controller interface, we

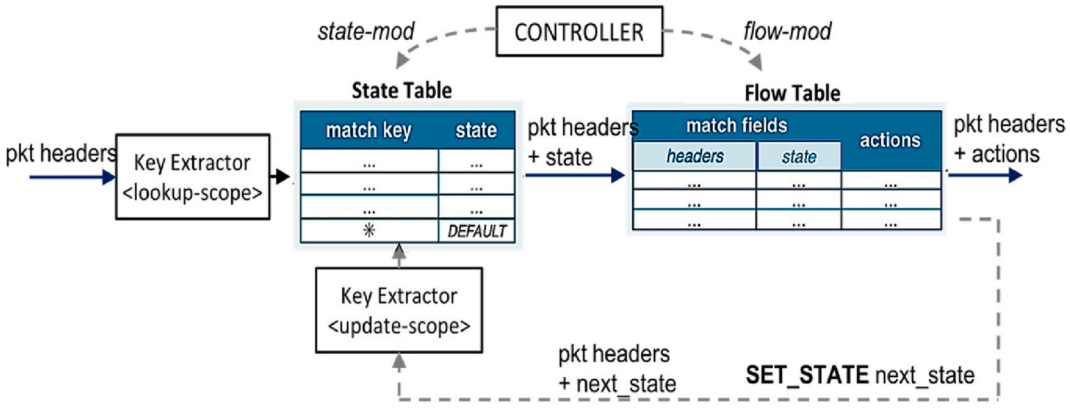

Fig. 16. Stateful Flow Table Operations workflow.

make two key implementations.

- We aim at collecting a *compact* representation of packet distributions while retaining enough information about flows to enable high accuracy on packet inspection and flow classification tasks. Achieving such a representation requires the design of a flow compression scheme that is simple enough to be efficiently implemented by the primitives available in current programmable switches, but which is able to retain meaningful features for classification purposes.
- Computing *compact* representations of packet distributions should not consume the majority of resources in the switch, enabling the system to co-exist with other typical applications, e.g. forwarding, or to be used in tandem with complementary network telemetry solutions.

It is imperative to find the correct balance between memory savings and accuracy. OpenPATH balances this trade-off, for different use cases, through a parameterization during the profiling phase. First, we devised a new compact representation of packet length and inter-time packet arrival distributions which is small yet provides enough information to perform accurate application-specific traffic classification. We name such representations as *synapses*. OpenPATH data plane layer (called as *VARMAN*) offers similar accuracy scores while using significantly less memory, e.g., covert channels can be detected with at most 3% loss in accuracy using only a 20-byte memory footprint per flow. When compared with related methods for capturing compressed packet frequency distributions (Nasr et al., 2017), OpenPATH consistently outperforms them in terms of the classification accuracy under similar memory restrictions, considerable bandwidth savings when compared to network telemetry approaches (Sonchack et al., 2018) that rely on a server infrastructure responsible for flow analysis. The *synapses* generated by vSwitch profiler depend on the parameters, i.e., the *compaction table*, dictated by an application-specific profile which determines how efficiently the switch SRAM will be used and how accurate the flow classification will be. In general, finding the parameters that offer an optimal trade-off would require an exhaustive search of the parameter space. To search on the parameter space for a configuration that offers a good trade-off between *synapse* size and classification accuracy, the profiler implements alternative customization policies that can be enabled by the OpenPATH operator. Note, that synapses are tightly coupled to a specific implementation of the profiler in VARMAN (Krishnan et al., 2019a) and nothing prevents the use of alternate optimization techniques. Unlike other ideas, such as OpenNF (Gember-Jacobson et al., 2014), we envision a more distributed, hierarchical architecture for state management. Internal and external non-NF-related states will be managed and adjusted by the centralized SDN controller. This reduces the SDN controller's potential overhead. In addition, it enables finer-grained state maintenance at lower levels of the hierarchy, which obviates the need for regular SDN controller updates and the associated communication overheads. As the NF state is distributed across multiple nodes, the SDN controller initializes the tables across the nodes and periodically synchronizes the state tables. This problem and the management overhead are not unique to OpenPATH, and we solve this issue efficiently through the native design choice of "*Statefulness*" in the data plane. It is critical to enable the data plane to not only characterize flows using packet sequences, but also handle traffic using aggregated data and compact representations across multiple flows. Our experiments demonstrate that OpenPATH can handle traffic in a multi-flow aware manner without requiring costly communication with the SDN controller.

6.2. NF manager

NF Manager is the core system process that runs on every vSwitch in the dataplane. This is a kernel process that runs with high priority and handles packet movement between the VNFs and maintains the flow-tables. Zero-copy exchange of packets between the VNFs is accomplished by allocating DPDK DMA packets from NIC buffers to the "shared memory *huge pages" region* in userspace. The descriptors to the Tx/Rx ring buffers are passed for the VNFs to give access to packets in the queues. When packets enter the NIC Rx threads in the NF Manager use the polling mode worker module (PacketQ) in DPDK to move packets into the SM area that all VNFs can access. The Rx threads check the packets of a flow and execute the match-action for the matching rule in the TCAM, copy the packet descriptors to Ring Buffers, and finally the waking up the NF processes to running state in the dedicated CPU cores. Once the application logic of the NFs on a packet is run, the NF Handler/Tx Thread transfer the packet descriptors to the next VNF Rx queue through shared memory. When the packet reaches the end of the chain, the NF handler on the last VNF moves the packet to the NF-Manager which merges the packets from all the paths of the SFC and invokes the PacketQ to forward the flows on Egress ports. For each VNF the NFM holds "a pair of ring buffers Tx/Rx" for exchanging packet any VNFs in the Chain. The Rx threads of PacketQ ("DPDK polling mode device driver that avoids interrupt handling overhead") receive the packets arriving at the Ingress ports of the vSwitch node. The PacketQ *looks up* at the SFC flow-table setup by the NF Manager and finds the head VNF in the chain and the SC- ID defined for the new packet. The descriptors to the new packet are added to the Rx Queue of that VNF. If the PacketQ lookup returns a table-miss/no path is defined, then the new flow packet is moved to Upcall-handler's queue (to be forwarded to the controller). We are repurposing OFPT FLOW MOD messages from the OpenFlow protocol to define the forwarding behaviors across the VNFs in a service chain. The SDN Controller must recall the forwarding-rules implementing the part of the SFC graph from one node and rules to properly forward packets from the last NF in the chain. The packets are forwarded from one network domain to another domain, deploying different segments of a single SFC graph across multiple nodes (or a transfer on the way to that node). The controller installs service SC-ID to process flow and sets the flow operation to be "output to port SC-ID." When a packet reaches the end of the chain (tail/last VNF), the merging of all the versions of that packet has to be done. This process is defined by the NF-Manager and done by a dedicated VNF process. The NF Manager raises the parallelization factor by the reference counter. If multiple VNFs are processing a packet, it is likely that each will ask for a different action at the end or merged into a single match-action. The "match-action" on the final result is executed to either drop/forward to the Egress port or steer the packet to another SFC/SC-ID path. When running network functions involving complex operations/analyzes of packets, packet-processing overhead varies across the flows. As a consequence, routing packets to NFs using round-robin scheme will possibly cause an imbalance in the queuing that could lead to dropped packets and erratic latencies. So, the NFM is designed to load balance the NFs based on the active ring-buffers used by the VNFs. The simple methods such as "round robin or queue-size balancing" cannot be used for scheduling VNFs involving temporal flow state, as the descriptors have to be defined for each thread to avoid contention. To tackle this problem, we implemented *flowID* based load balancer making sure that all the packets of a *flowID* go through the same NF instances/threads to avoid context switching complexity. We extended the DPDK with data-structures and fast-lookup tables for storing the VNF-packet-paths and dependencies to handle concurrent access on packets by multiple VNFs.

6.3. Management and controller plane

The OpenPATH Management/Control plane system comprises three major centralized software services: SDNFV Orchestrator, per-domain SDN Controller, and ETSI (European Telecommunications Standards Institute(ETSI), 2014) compliant NFV MANO. SDN Controller manages the dataplane instantiations (vSwitch): setting processing logic, controlling provisioning, flow-table, loading VNF modules dynamically on the switches, routing, and scaling of network bandwidth. The SDN

controller, NFV MANO components use a secure communication OpenFlow over TLS and OpenPATH control overlay in simple JSON (Bray, 2017). On SDN side, Open vSwitch (OVS) control plane design was implemented without modification, on OpenDaylight(ODL) OpenFlow controller (Medved et al., 2014). There are three ways to control the OpenPATH: Python/C APIs/Scriptable configuration language, OVS-like CLI and everything component is run-time configurable.

6.4. NF processing

NF/application processing logic is implemented in the OVS context and through the *netlink* interface, the flow-tables in the kernel space are updated. The VNF table is handled in conjunction with the traditional OpenFlow tables. Every Packet is looked up with the flow-table for matching-rule and the action *function ()* for that rule is run. If table-miss in lookup, that packet is sent to the control plane. An NF Manager has these responsibilities: (i) determines the action for the current flows. (ii) modifies the corresponding state table persistent state. (iii) generates or modifies the rules for the kernel flow-table. (iv) generates/receives packets to/from vSwitches nodes. (v) forwards "*Packet_In, Flow_Removed, or NF_update vendor*" messages to the control plane. Each VNF is provided an **NF handler**, which handles OpenFlow instructions and flow updates through the NF Manager from the controller. OpenPATH framework is designed to allow new applications or VNFs to be deployed on the fly even when vSwitches are running. Through OF_VENDOR action command, the new NFs are sent as sub-actions by the SDN controller to the NF Manager which installs the appropriate flow rule in the VNF NF handler agent in the SFC.

6.5. SFC placement

Traditionally, network services are deployed as function chains (SFCs). As, each SFC's overall performance is limited, we define the parallelized SFC in which the data within each working sub-SFC is less. To get closer to reality, we include both VNF and PM failures when we evaluate parallelized SFC availability. OpenPATH NFV Placement algorithm maps a set of SFCs. The operational sub-SFCs are mapped first, and then the backup sub-SFCs are mapped. Each NF instance must be mapped to a single NFV server (vSwitch node). The purpose is to reduce inter-node traffic since it incurs reduced delay and requires fewer processor cycles to traverse via the NICs while also constraining the link bandwidth between nodes and the switch.

- **Dependency and affinity**: The metadata sharing between the NFs (for e.g., sharing the same IP headers in the NAT and Load Balancer) have full dependencies and placed adjacent in the process-queue chain. A majority of stateful firewalls can only operate on one instance at a time, making affinity important. There are instances, where separating an NF's instance (and accompanying input traffic) is necessary. Any methods that retain cohesion rely on state migration methods, which is expensive and not compatible with legacy functions. There is no switching overhead across the switches to VMs.
- **Fault Tolerance:** Fault tolerance and backup SFC nodes (except one work by Wang et al. (2020), the failures and recovery are not addressed). The two existing NFV execution paradigms are RTC and Pipeline. Reduced PL cost is one of RTC's objectives. By distributing a single VNF in a PM, network operators are able to manage VNFs. We create the placement strategy and a hybrid placement algorithm that incorporate the advantages of two existing NFV execution models. To retain affinity, the software switch and controller work together to build a novel placement technique. Our system doesn't require state migration, uses less flow table entries than the switch when transferring traffic to NF instances. Our offered solutions can reduce SFC delay and resource utilization while guaranteeing Availability.
- **Microservice model:** the placed NFs are application processes that are already deployed in the system. The processes are in Sleep-Wait state and will be scheduled (chain of processes) as soon as a SFC request is ready for execution. (Our model is akin to "*micro-services*" deployed in high-end enterprise class routers and process the SFC requests in non-blocking mode at wire-speed)

6.6. Software acceleration

OpenPATH uses the DPDK I/O abstraction and improves it with SM that enables VNF/processes/apps to pass buffer descriptors via the zero-copy API feature. The OpenPATH threads DMA data by-passing the Linux kernel to the *SM region.* We employed "*NUMA*-awareness (forcing socket locality)" and network driver in polling-mode to avoid costly interrupt handling operations and other types of notifications. The performance improvements such as tuple-space search algorithms for table lookups (Srinivasan et al., 1999), Read-Copy-Update rather than traditional locking (McKenney and Slingwine, 1998), and packet batching (Kim et al., 2012) have all brought substantial benefits to switching in both code-and data-driven designs. The zero-overhead/copy-free Shared Memory based communication channel is utilized to move packets and other messages between the processes/VMs/containers and the physical vSwitch host. We use asynchronous ring buffers mechanism in conjunction with shared memory to implement zero-copy operations on packets and also ensured consistency without the use of locks. The Receive Queue resides in the SM (user-space), so that the VNF NF handler has to write a descriptor to the Rx Q of the next VNF in the chain for moving the packets. To further reduce copying overhead for parallel operations, we implemented optimization techniques such as "*Header-Only Copying*". The PacketQ engine is a kernel poll-mode driver that runs on a dedicated CPU core that transfers frames from the hardware via DPDK and kick starts the SFC workflow with the head VNF in the chain. The softwarized switching implementations are nowadays comparable with hardware/ASIC based solutions and high-speed software networking requires acceleration techniques in various axes which are illustrated in Fig. 17. OpenPATH implemented various software acceleration methodologies as listed in Table 4.

6.7. OpenPATH API and applications

We use our taxonomy to build a Stateful API for flow-aware NF/application logic for exporting, importing, and minimizing changes to their code. We use the specified transport layer connection-oriented protocols (TCP/UDP) metadata fields (e.g., TCP or UDP connections) and define the abstraction for the controllers to specifically determine which state to be exported or imported to the switches. OpenPATH

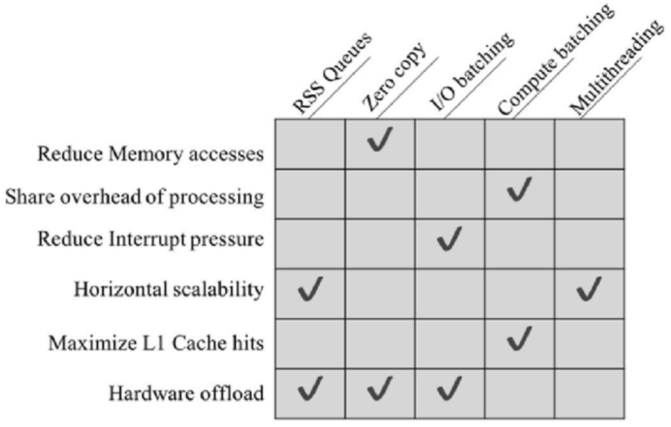

Fig. 17. Acceleration techniques and benefits.

Table 4
Software acceleration methods.

Flow awareness	Zero-copy	Packet processing	Coding Practices
Per-flow processing	Limited state	I/O batching	Multiloop: Many packets at once
NIC calculates hash function of 5-tuple	Using shared memory	Compute Batching	• More instructions per clock cycle
Hash value is exported at userspace	NIC use Direct-memory access	Poll-mode device drivers	• CPU pipeline always full
Packets traverse Rx/Tx queues	Independent processing	Reduce Interrupt pressure	Prefetching: start reading next packets
Multi-core scalability	Payload is unmodified	Packet Arrivals in Batches	• Reduce overhead of memory access
One CPU core/HW queue	Only metadata is accessed	No individual packet departure	• Exploit cache hits
Receive-side Scaling (RSS)			

library and SDK enumerates various types of API for NFs and control applications to implement application logic in the dataplane and to manage/monitor state transitions of the corresponding flows and connections. The APIs provide programming interface (1)to configure global state table and the state management table, (2) to retrieve state information from the global state table, (3) to register call-back functions in OpenPATH to subscribe specific state-based events and (4) to manage and manipulate the flows based on the state-based events in VNF.

In Fig. 18, a basic NF/app module is shown in full as an example. The NF/application requires OpenPATH to register a function. To realize the design explained, OpenPATH has incorporated a set of core network functions that are employed as worker processes in the SDNFV dataplane. Some key functions that operate on packets are: load monitoring, flow tracking, load balancing, packet classification, merging/splitting, boot-up/shutdown/placement, DoS detector/scrubber/sandbox. We export a native API to provide direct access to the optimized library and use OpenPATH socket abstractions. The legacy NF/applications can be unchanged and utilize the standard sockets/other northbound APIs supported by the OS. OpenPATH SDK and OVS-DPDK library export API to implement custom NF and packet/flow-based inspection/monitoring/filter policy definitions. Also, the native modules and special datapath application modules that are already built in the OpenPATH vSwitch software can be utilized to experience the highly programmable switching framework.

7. Performance evaluation

The findings of our research on the programmability and efficiency of this proposed switching platform are summarized here. The evaluation environment is intended to serve as a laboratory for experimenting with various design choices, benchmarks (Tahhan et al., 2017) and reconfiguring the OpenPATH SDN-based-NFV Platform software for various test cases. We evaluated both in real hardware-based testbed (described in Section 7.1) and in simulated Mininet-based network testbed (Section 7.8). Against each sub-section in the Evaluation section, we specify the testbed setup (hardware or simulated) for the experiments in that section.

7.1. Hardware testbed configuration

To measure how well our OpenPATH framework would perform on industry-grade switch hardware for the vSwitch nodes, we tested our implementation of a Whitebox router that supports Open vSwitch switching software.

The Major components of the testbed as shown in Fig. 19 are:

- **vSwitch Nodes** (NFV platform running OpenPATH): 4 server class machines with dual Xeon X5650 @ 2.67 GHz CPUs (2 × 12 cores), NVidia GPGPU, Intel 82599EB 10G Dual-Port NIC (Both are connected to an experimental LAN for data-plane traffic), 48 GB (8 GB for huge pages), Ubuntu 16.10 (kernel 3.5, DPDK-1.4 and QEMU-1.5). Each machine has one 1 GbE Intel NIC for control and management traffic.
- **Data Traffic Switch:** NFV Nodes connect to a Cisco SX350X-24F Stackable Managed Switch | 24 Ports 10 Gigabit Ethernet (GbE)
- **Traffic Generator:** Network Function Performance Analyzer (NFPA, (Csikor et al., 2015)), is a benchmarking tool using the DPDK *pktgen.* NFPA and OpenPATH ingress run on dedicated machines.
- **OpenFlow Switch** (running OpenPATH Switch software): Whitebox high-end switch (its private vendor who we had to anonymize since we did not get a permission to use the vendor's name, running OpenWRT (OpenWrt Wiki, 2017)) We also experimented with Pica8 (Pica8, 2013) P-3290 (with a modified Indigo OS (Indigo)).

7.2. Comparison of OVS-DPDK SDN stack

The softwarized switching implementation in OpenPATH is comparable with hardware/ASIC based solutions exploiting acceleration techniques in various axes which are illustrated in Fig. 17 and in Table 4. The OVS-DPDK classical implementation doesn't adopt the above optimizations and acceleration in software. This is explained in Section 6.6 Software Acceleration.

The observations in the design and implementation of OVS-DPDK are

Custom NF : Module developed with OpenPATH

```
static struct nf_hook_ops myhook_ops
__read_mostly = {
.pf = NFPROTO_IPV6,
.priority = 1,
.hooknum  = NF_INET_LOCAL_OUT,
.hookfn   = myhook_fn,
};
static int __init myhook_init(void) {
return nf_register_hook(&myhook_ops);
}
static void __exit myhook_exit(void) {
nf_unregister_hook(&myhook_ops);
}
module_init(myhook_init);
module_exit(myhook_exit);

Main function:
static unsigned int myhook_fn (unsigned int hooknum,
struct sk_buff *skb, const struct net_device *in, const
struct net_device *out,int (*okfn)(struct sk_buff *))
{
pr_info("OpenPATH Hook approves your packet!\n");
return NF_ACCEPT;
}
```

Fig. 18. Example network function implementation.

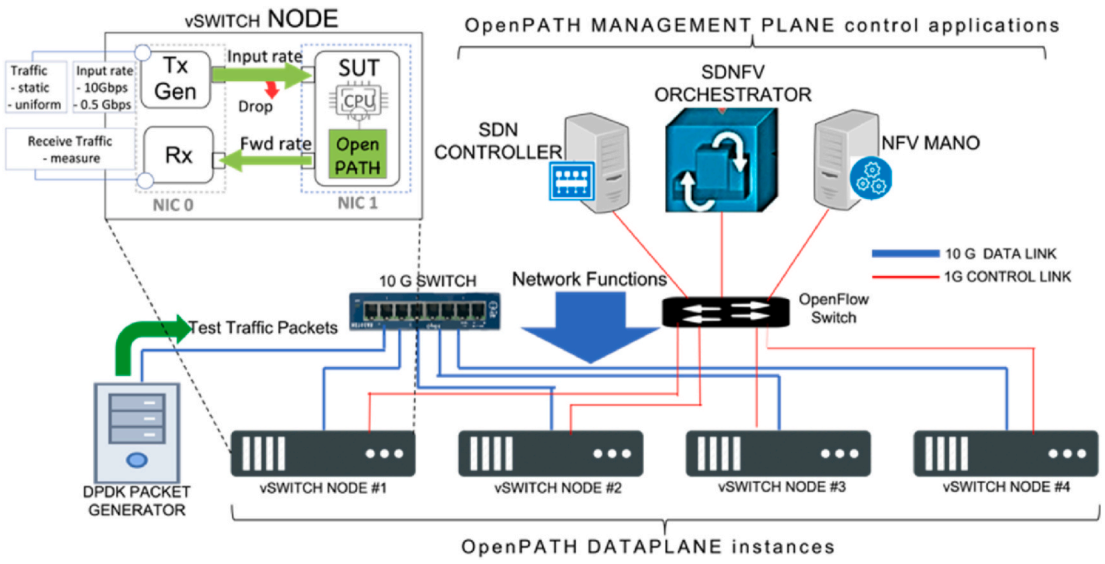

Fig. 19. Testbed and SDN/NFV infrastructure for OpenPATH.

enumerated below:

- The major overhead in the OVS is the match/action pipeline to orchestrate flows, flow table. OVS tries to optimize this function with aggressive flow-caching. In synthetic traffic conditions, the flow caching further worsens the matching speed. OvS-DPDK suffers performance degradation due to the match/action pipeline overhead (Zhang, 2019).
- OVS-DPDK is slower than the other passthrough systems because of the VirtIO. The design of VirtIO offers an additional degree of indirection (helpful for live migration), which is not present in the other options: each packet sent or received by the VM goes via the VirtIO queues before reaching to the network backend (i.e., the software switch) (Gallenmüller et al., 2015; Lettieri et al., 2017).
- There is a queuing delay in the pipeline architecture-model which generally involves multiple threads and scheduling/wait state (Rx/Worker/Tx) for a single packet forwarding and the packet is internally buffered twice to take over the packet handling between threads. Though OVS-DPDK doesn't incur queuing delay due to DPDK's Run-to-Completion model that entire packet processing is performed by a single thread, there is buffer saturation and flow control that could lead to packet drops.

We conducted experiments with service chains composed of 1–4 NFs with three software switches (OVS-Kernel, OVS-DPDK and OpenPATH), with 256 bytes packets. The packet delays due to the softwarized switch, in the datapath between two physical ports, is measured. DPDK-based packet I/O threads continuously poll the ports (in a busy-wait loop) and do not rely on interrupt calls and so avoids the context switching overhead. This makes 100% of CPU cycles available to the packet logic processing rather than for moving(copying) the packets. But the interrupt moderation overhead has an adverse effect on the OVS-kernel and so higher latencies are seen. Table 5 shows that OpenPATH achieves low latencies and high throughput rates for all chain lengths utilizing available CPU cores for NF Parallelism (packets split across paths/cores). As the chain length and cores increase, scale-out performance closer to the theoretical limit of 10Gbps is achieved and OpenPATH exhibits a consistent and lower latency profile.

We evaluated throughput sensitivity to concurrent flows, comparing OpenPATH and OVS-DPDK on basic L2/L3 routing workloads and concluded that OpenPATH maintained higher throughputs (See Table 5) than OVS-DPDK as the number of concurrent flows increased. However, a deeper inspection of the match/lookup algorithms reveals a more complex story. OVS-DPDK uses a flow-caching strategy to reduce lookup times for complex multi-table lookups required for network virtualization. This flow-caching strategy introduces an additional step in packet processing that increases per-flow overheads for short, single-table lookups (e.g., L2/L3 switching). However, the authors of OVS claim it improves throughputs for longer, multi-table tasks (e.g., network virtualization) by benchmarking the differences in performance between the algorithms in isolation of other design choices and optimizations within the switch implementations and showing their strengths (e.g., multi-table lookups) and weaknesses (e.g., large numbers of independent flows).

Table 5
Software switching performance.

Chain Length	CPU Cores	Throughput (Gbps)			Latency (µs)		
		OVS Kernel	OVS DPDK	Open PATH	OVS Kernel	OVS DPDK	Open PATH
1	2	7.82	8.87	9.45	254	178	127
2	4	6.14	8.24	9.36	328	254	132
4	6	5.26	7.20	9.27	478	378	142

CPU Caching (Fig. 20): After the traffic locality is gone, performance of OVS-DPDK flow caching deteriorates. With 100 active flows, packet rate decreases by a factor of two (or even worse). When differing cache hit intensities are experienced, the flow processing, shifts from the

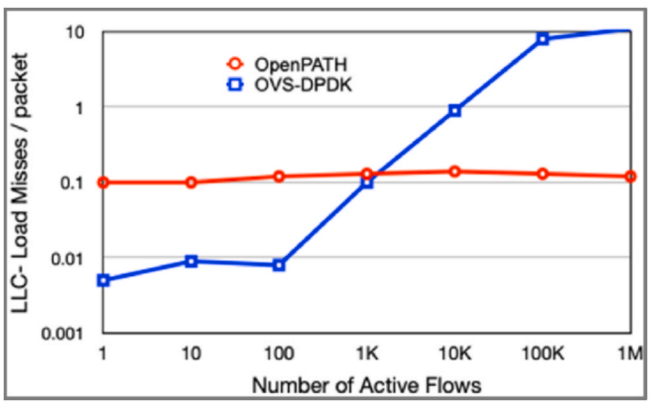

Fig. 20. CPU caching (LLC) behavior.

highly rapid microflow cache to the slower megaflow cache. Additionally, cache affinity is compromised leading the flows through the vSwitchd slow path. We monitor the LLC misses using *perf* tool, as the active flow set develops.

CPU Latency (Fig. 21): As the active flow set expands, the latency on the gateway pipeline increases. The OpenPATH processing time in the data path depends on the active flow set, although latency for OVS-DPDK might vary between 0.3 and 0.14 μs. The constructed datapath (in OpenPATH) delivers reduced and predictable latency compared to a flow-caching-based datapath (in OVS-DPDK).

We tested over a range of pipeline complexity levels and a growing diversity of traffic mixtures. The key inferences are:

- OpenPATH side-steps flow caching and achieves a consistent and high packet rate across virtually all OpenFlow pipelines and traffic mixtures studied, routinely exceeding 12–14 Mpps packet rate (max ~ 15 Mpps). When applied to complex pipelines with many active flows, OpenPATH can perform by two orders of magnitude better than vanilla OVS-DPDK. For the Gateway domain, an OVS-DPDK throughput decline of a hundredfold takes place, at 1 million flows, resulting in a total denial of service for the user population. Meanwhile, OpenPATH delivers above 9 Mpps packet rate, which suggests that it is not vulnerable to these assaults. For everything else, OpenPATH will employ OpenFlow style match-action tables. This, together with global flow caching and categorization coalescing, lets OpenPATH take advantage of both. For big packets, like network virtualization, OpenPATH is the perfect fit.
- OVS delay can be traced to generic flow-caching in the datapath code; In contrast, OpenPATH's compact bespoke data pathways offer significant switch performance and modest working set size. OVS-DPDK is designed with the classical SDN architecture, wherein Controller does all the heavy lifting work and complete decision making and a stateless dataplane. But OpenPATH has a stateful dataplane in which switches have been delegated with some control over the decisions.
- Finally, the classical OVS stack is coded as a reference implementation for generic Linux architecture and there are scope for improvements, optimizations, software accelerations, customized logic tuned to the deployed hardware platform.

We hope this explains the improved performance of "OpenPATH (OVS-DPDK)" against "vanilla OVS-DPDK", though we leveraged on the code base of OVS-DPDK.

7.3. Stateful operations and flow table optimizations

Packet processing is among the most demanding operations for network switches, and hence specialized hardware and advanced software acceleration methods are used to complete it. For the hardware side, it's possible to implement the data plane functionality in an ASIC (Application Specific IC), an FPGA (Field-programmable Gate Array), or a network processor, using dedicated packet classification engine chips (TCAM) to build a software switch. In hardware switches, ASIC capacity and the built-in TCAM capacity limit the performance, but software switches are hampered by the intrinsic computational difficulty of packet classification on general purpose CPUs. In an SDN switch, a forwarding table includes per-flow routes. Wildcard fields and concurrent lookup of all table entries are typical uses of TCAM forwarding tables. For accurate rules, they can be implemented in TCAM or SRAM (Liu et al, 2010). Though we see it as a single table, it may be implemented using multi-tiered tables. Apparently, in order to properly maintain the SDN in a dynamic network environment, we need to consider dataplane switch reconfiguration to balance the on-chip memory (TCAM and System RAM) utilization from time to time. Although this idea is straightforward, realizing it effectively is still challenging from both algorithmic and systemic aspects. Data paths in DPDK-based appliances translate to flow entries in the action's internal flow table. The hardware-based middleboxes may perform lookup from internal TCAM tables or other storage media. Action elements in the processing graph represent each action's read flow and write flow. The *match-action* switching model represents the lookup tables (flow-tables) layout, size, updating and dynamic management of the TCAM and the SRAM. Whenever packets arrive at the Ingress ports the matching flow in the tables are *looked-up* for *match* and the corresponding *action* (e.g., forward) are taken by the logic (ASIC in the hardware/application in the software) as part of the pipeline in the switch. So, the delay and resource usage for this function are some key parameters in SDN enabled switching architecture.

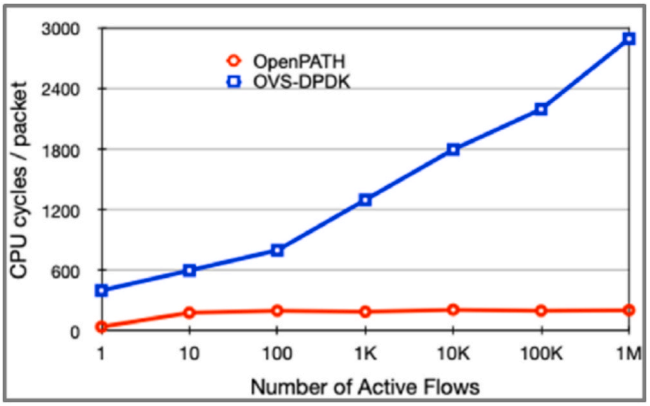

Fig. 21. CPU Processing cycles.

As a substitute for the NIC with a programmable TCAM, we used a software stack in the current testbed to imitate 1k TCAM entries. We discovered that it scales smoothly when offload rates approach 100%. The current simulation features a modest TCAM and a limited hash tables (SRAM) but provides all the functionalities needed for reverse path applications. First, we collect performance-related metrics such as the number of in-flight instructions, the current flow table occupancy, and request batch size. To find key factors affecting rule lookup/update speed, we sample the entire parameter space. The number of rules contained in a flow table is critical for a switch. Bigger tables enable finer-grained traffic control. However, TCAM space is expensive, therefore tables that support intricate matching tend to have restricted dimensions. We expanded on this feature by using generic OpenFlow-like switch models. The OpenFlow Switch (running OpenPATH Switch software) is a *Whitebox* switch (its private vendor who we had to anonymize since we did not get a permission to use the vendor's name, running OpenWRT (OpenWrt Wiki, 2017)). We also experimented with Pica8 (Pica8, 2013) P-3290 (with an Open-vSwitch (OVS) support and CrossFlow technology enables mixing of switching, routing and OpenFlow traffic, Modified Indigo OS (Indigo)).

To evaluate the efficiency of Stateful data plane management scheme using *synapses* and truncated table maintenance with the given hardware resource usage on the switch, we focus independently on the data plane and on the control plane. To give a general insight into the performance of these synapses scheme, we considered three different traffic pattern, network function (VNFs) with usage scenarios that are common in Firewall systems.

Flow Table Scalability: Table 6 presents the scalability gains of our stateful firewall application deployed in the vSwitch (OVS) node, when it is used to classify flows for covert protocol detection, HTTP fingerprinting, and botnet attack detection. For these experiments, we generated the possible combinations of synapses for the three considered use case scenarios and assessed whether they allow for accurate flow classification despite their compact size. Packet lengths (PL) vary from 1 to 1500 bytes (MTU), and each cell of a synapse has a size of 2 bytes. In general, the absolute number of flows that stateful table

Table 6
Scalability of flow tables.

Firewall NF Type	Synapse Size (Bytes)	Full Data Size (Bytes)	Scaling
Covert Channel Detection	20	300	150 x
HTTP Monitor Fingerprint	94	300	32 x
DDoS Botnet Detection	302	10200	34 x

management scheme can handle ultimately depends on the switches' available SRAM.

Switch Resource Utilization: Table 7 shows the average hardware resource consumption of synapses *data store* across all stages of the switch, both in simulated TCAM/*Whitebox* switch). The table shows that besides the SRAM required for the tables and register, the consumption of other resources is negligible. Since our flow matching logic entirely relies on exact matching, the vSwitch's flow table consumes small fraction of the TCAM resources on the switch. In tandem with the deployment of flow tables in SRAM, our scheme leaves over 60% of SRAM available. Overall, these results suggest that *Synapses* scheme makes enough space in the switch memory and also bandwidth on the control channel for the concurrent execution of many other common forwarding behaviors, like access control, rate limiting or encapsulation, that do not necessarily require an extensive use of the stateful memory in the switch pipeline. We argue that the lookup/update operations are very fast and optimized in the datapath pipeline and we hypothesize the storage overhead of OpenPATH will be manageable even for hardware targets with limited SRAM/TCAM (Liu et al, 2010).

The key findings are:

- With compact synapses Stateful switch can monitor at least 32 times more flows when compared to the baseline setup with complete raw data from the flows maintained in the switch table and forwarded to the controller.
- The system (combination of Controller/Switch) manages to achieve a classification accuracy of 94%, only 2% shorter than the result obtained using raw packet length distributions.

7.3.1. *Stateful TCP-Offloading overhead*

This experiment compares the TCP/IP Firewall in stateful and stateless mode. Whenever the host on the internal network establishes or terminates a network session, the flow rules are changed. The delay to update the rules can affect TCP link timeouts. We used a software gateway for the hosts to connect to the HTTP server. The number of hosts ranges from 8 to 256. Fig. 22 shows that for new sessions, SDN stateful TCP/Firewall creates a bump-in-the-wire causing an additional delay than the stateless gateway switch. The OpenPATH based stateful firewall requiring just extra 2 ms or around 10% longer time to manage the state tables/sessions and flow tables in the switch. However, once the connection is active, the switches manage the flows and sessions on their own, so the control plane load does not have to be increased.

7.4. *Comparing SDNFV switch and NFV-over-VMs*

In this experiment, we ran a series of micro-benchmarks with a cluster of machines (in hardware and simulation) to compare the

Table 7
Switch resource consumption.

Resources	Computational			Memory	
	Lookup- Match-Action	Routing Gateway	SFC Tagging	TCAM	SRAM
Usage	7.56%	4.21%	3.19%	1.2%	37.21%

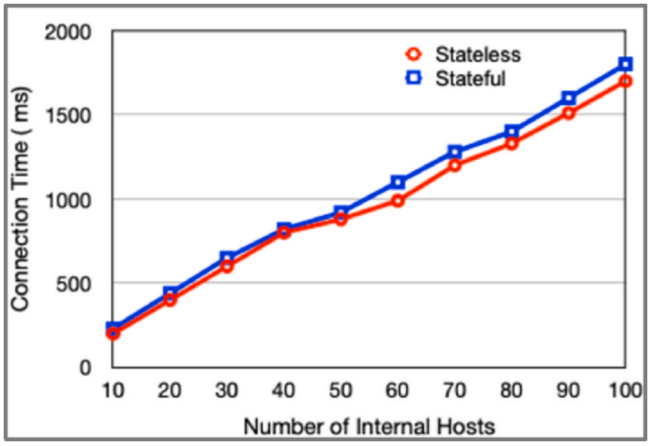

Fig. 22. Connection setup time.

efficacy of the OpenPATH and the traditional NFV platforms. With OpenPATH SDNFV switch, network functions are already integrated into the switch node, and the chain is formed by creating a set of flow-tables with output: action (call the next NF) function call in the same OS context. We ran these tests in a controlled environment under a single experimental node (as an OpenPATH switch or as physical host to VMs) and this removes any outside network delays. This design gives the flexibility to represent a complicated SFC graph, avoiding the I/O transfers over the physical node across the VMs. We also ran simulations with Mininet (Lantz et al., 2010) and performance benchmark iPerf (Iperf) application. Fig. 23 shows the testbed design for the experiment. In Fig. 23(a), the NFs are implemented in the VMs and interconnected through a switch. A packet has to be processed for "n+1 match-action lookups" in the OpenFlow flow-table, to be steered in an SFC comprising "n NFs. In Fig. 23(b), with the OpenPATH SDNFV switch, the NFs are implemented into a single node, and with 1 multi-fields tuple-lookup, the traversal through the chain of NFs is just invoking another function call. The differences in processing overhead are measured through latencies and throughput.

TEST CASE 1: The overhead of changing the NFs (order or policy or priority) in the SFC chain triggers more processing in the SDN pipeline as the flow-tables have to be updated and we compared this key metric as it indicates how agile the NFV platform is when the dynamic traffic policies are changing. In this test case, we measure the NF switching overhead/delay for the new NF to be installed and the arrival of the first packet. Fig. 24 shows the new NF switching delay in the legacy approach (VM) is higher as it involves more I/O cycles and booting overhead of the new VM in the chain. With the OpenPATH approach, the switching does not change while spinning up the NFs in the initial stage and also at run time re-installing another NF into the chain, as the overhead for function calls and initializing the Tx/Rx packet queues are constant.

TEST CASE 2: In this test case, we compare the forwarding rate between OVS-DPDK, OVS-CLASSIC, and OpenPATH with a load balancer. As discussed previously, due to redundant packet detours, sub-optimal I/O and data copy overhead the classic and also the OVS-DPDK perform badly with the scaling of NFs. Fig. 25 shows the effect of optimizations such as fast I/O, load-balancing, and shared-memory/zero-copy architecture. OpenPATH switch can perform close to the peak network speed, sustaining with increasing the NF chain.

TEST CASE 3: We compare the latencies between the two approaches. Fig. 26 shows that as the number of NFs increases, the processing latency increases considerably when running NFs on VMs. The processing latency is increased due to each VM's I/O transmission delay to direct traffic to specific NFs on a service chain. A packet often requires multiple lookups on the switch so that various NFs can be reached in the correct order. More packets are waiting in the queue for a longer service chain, which increases transfer latency than a shorter service chain.

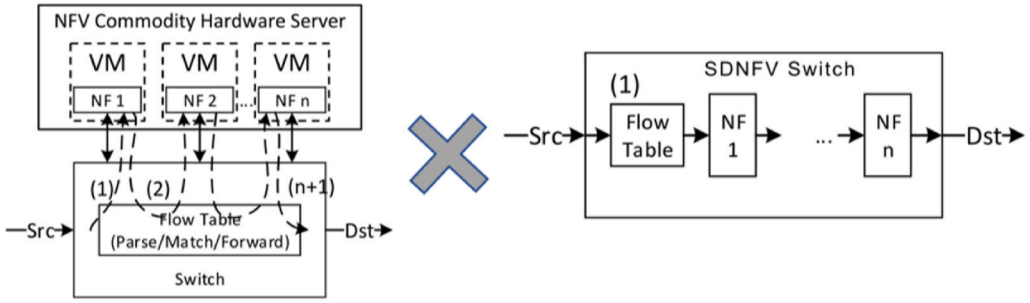

Fig. 23. NFV Platform (a) VM based Processing (b) Switch based processing.

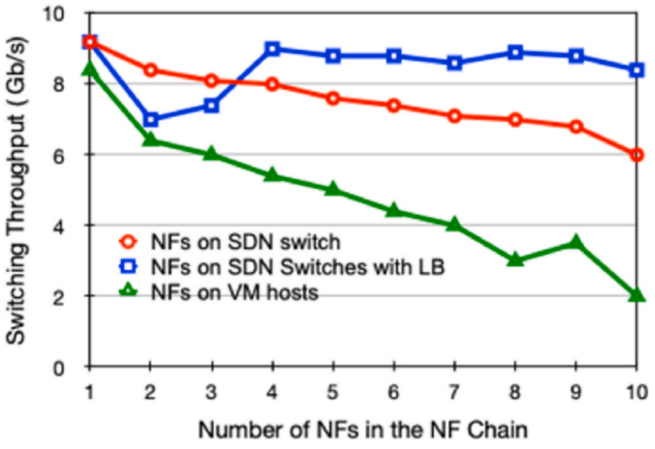

Fig. 24. Overhead for spinning up an NF.

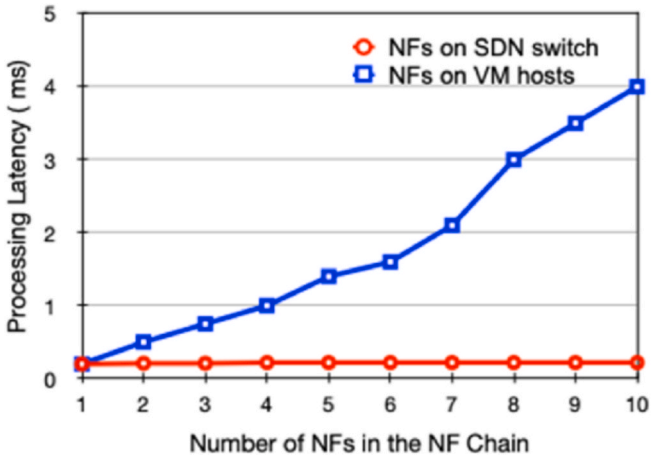

Fig. 26. Latency comparison.

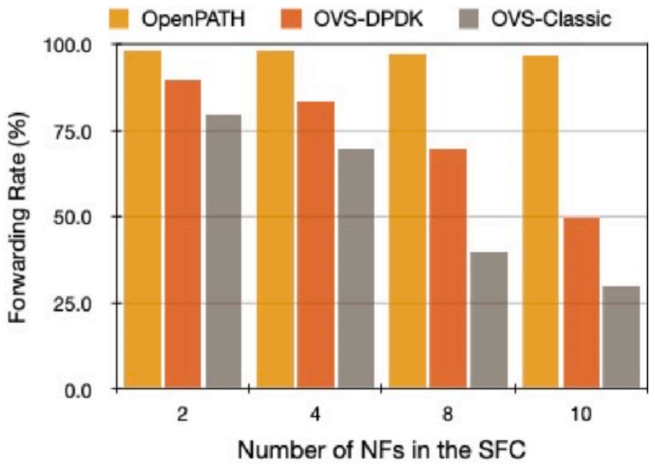

Fig. 25. Forwarding Rate comparison.

With OpenPATH, as the NFs are placed in one node, the network transfers across the switch are optimized.

In this experiment, we measured the throughput with a varying count of NFs in the chain. In the legacy configuration (NFs on VMs) the processing rate and throughput decrease as we increase the NFs in the chain. As discussed before, this bad performance is due to the non-optimal I/O, packets copying, across the stack and buffers, multiple contexts switching within the node and as the number of NFs increase the packet touring back and forth between the VMs through the switch. In the OpenPATH switch, with the NFs embedded in the single node, we save all the above-mentioned overheads and the other resource optimization techniques such as zero-copy have a played key role in achieving maximum practical sustained peak throughput even with the scaling of the chain.

7.5. SFC placement efficiency

With several NFV hosting nodes, VNF chains can be extended beyond the limits of a single node. We used a single core switch vSwitch node, which is linked to two edge vSwitch nodes, to set load balance and smart cross-node placement chaining. We also tested OpenPATH with other hierarchical leaf-spine topologies and configurations. SFC requests are parallelized using the NF Parallelism algorithm, SFC sub-graphs are generated and deployed using the proposed placement heuristic.

The Key evaluation metrics include:

1. Cluster overall-throughput, the bandwidth utilization ratio.
2. Total Accepted SFC Requests processed in the Cluster.
3. SFC Path Lengths, Average latency in processing.
4. Resource Utilization - The amount of used resource in the Cluster.
5. Acceptance Ratio-network wide accepted SFC requests.
6. Scaling with SFC chain lengths/no. of NFs.

The following approaches are Compared and Contrasted:

- Strawman Simple: placement algorithm (Pfaff et al., 2015) that does not take SFC parallelism into consideration, divides the NFs equally across the available NF hosts and keeps all VNFs as close as possible.
- NFP (Sun et al., 2017): "Packing" that greedily packs nodes onto servers while traversing the graph depth-first. It only allows placing an entire SFC on one server as required by NFP (Sun et al., 2017).
- ParaBox (Zhang et al., 2017): a naïve placement algorithm, in which VNFs are placed on different servers running in parallel.
- ParaMatch (Cai et al., 2020): deploying more NFs on a single node.

- FlexChain (Xie et al., 2021): parallelism-aware approximation placement algorithm.

Cluster wide overall-Throughput: OpenPATH seeks to maximize overall throughput by using NFs in a way that uses the least switch capacity. We consider a SFC graph, a linear chain with 7 NFs, and a more realistic random graph with 12 NFs. Fig. 27 shows that our approach outperforms the baseline strawmen approaches and 4 other works in all cases. We achieve about 15% higher throughput compared to other works; FlexChain does well on a simple chain but only achieves 5% more throughput for more realistic SFC graphs. Thus, we see that our placement heuristic in OpenPATH can improve the overall cluster throughput over the baseline algorithm. OpenPATH's overhead doesn't increase drastically when scaling out the number of nodes and VNFs, even if VNF's are placed in multiple physical nodes. The throughput reduces only a small level (~9.6 Gbps/NIC/node) compared with placing all VNFs in one physical node, which is due to the smart placement technique in SFC. With traditional approaches due to multiple traversals of packets across the nodes, they incur additional overheads, throughput is much lower in long NF chains that span multiple nodes.

Resource Utilization: Node resource usage was greater in the OpenPATH algorithm than in the strawman simple approach and other works shown in Fig. 28, as illustrated. Some VNFs may only use a portion of a host's resources, leading to a waste of fragmented resources. In contrast, OpenPATH allocates numerous sub-chains on separate nodes to take advantage of these dispersed resources. Our adaptability helps boost the SFC acceptance ratio.

Acceptance Ratio: Our goal is to maximize the overall number of accepted SFC requests, and a request can be granted only when all its limitations can be met. We conduct performance tests on a variety of network configurations. Between 20 and 300 SFC requests arrive at each topology. Fig. 29 shows the consistency in acceptance of SFC requests. Simple method has around 30% fewer SFC requests than our solution in latency-sensitive situations. Accepting the fewest SFC requests indicates the limitation in the NFP, that an SFC can only be placed on one server.

Chaining Placement Latencies: In this experiment, we measure the impact of the switch fabric type on the latency when running heavy Snort chains and complex SFCs across multiple hosts. OpenPATH employs smart affinity-based placement for SFC which shows lower latencies and saves up to 40% overall overhead for varying chain lengths. SFCs' path length had major impact on network delay and bandwidth. Using the strawman algorithm, the SFC path length output was shorter, but using the NFP it was longer. As shown in Fig. 30 the OpenPATH data path latency was around 52% lower than the simple strawman technique, and ~33% lower than the NFP approach. In addition, NFP exhibited superior capacity and stability to that of our placement approach in lowering the SFC delay. The average path length between the entrance and exit of SFCs grows as the network size increases.

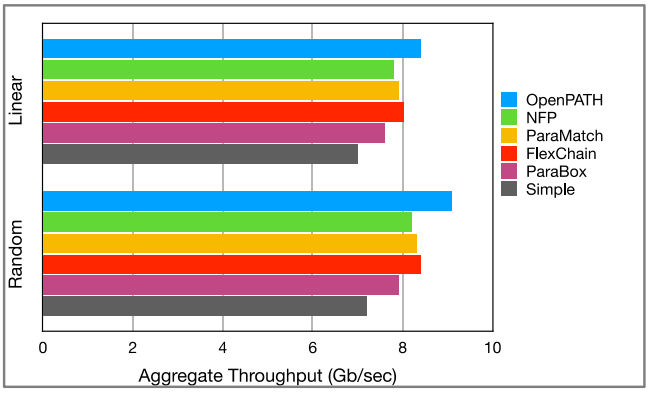

Fig. 27. Cluster wide throughput.

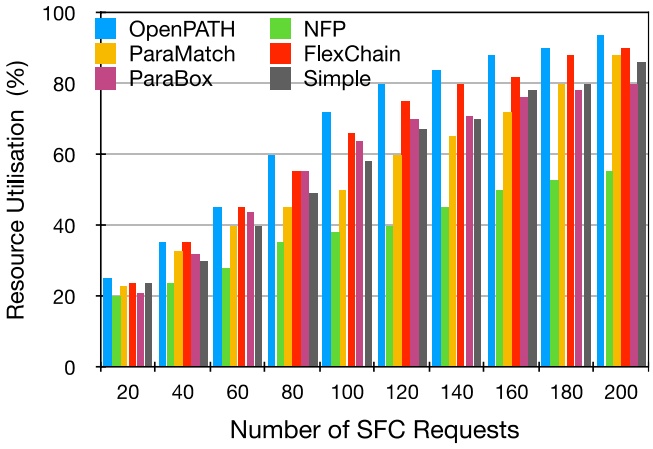

Fig. 28. Resource utilization.

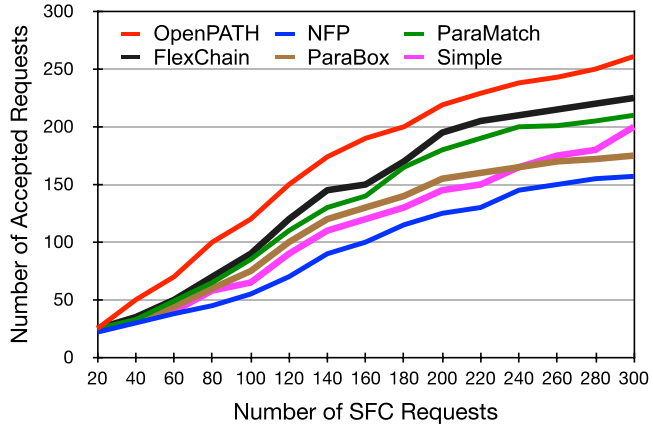

Fig. 29. Total accepted requests.

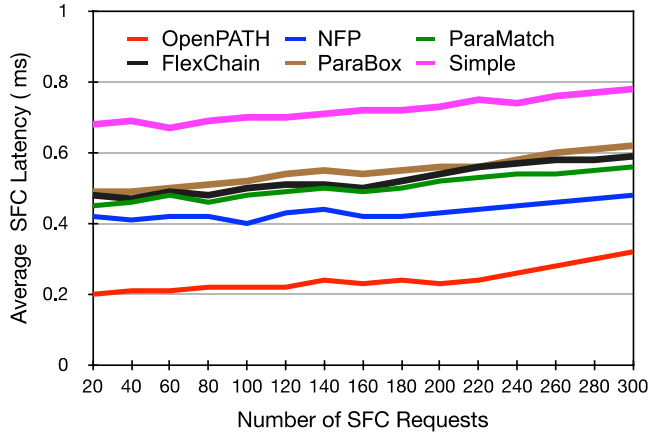

Fig. 30. Average latency of accepted requests.

7.6. NF parallelism overhead

The NFV parallelism and packet steering in SFC can cause considerable overhead due to copy/merge-operations. By employing mechanisms such as asynchronous ring buffer, header-only copy, and large shared memory regions for zero-copy access, we avoid the overhead. By implementing producer/consumer synchronizing mechanisms we avoided locking primitives. Also, we avoid central switch for packet steering, because of the NF handler which pushes the packet through the chain

across the shared memory using descriptor Tx/Rx queues. We compared the overall performance with 2 instances of the same NFs of sequentially and parallelly configured in a chain. The three testbed configurations are illustrated in Fig. 31. The testbed is installed with 7 VNFs in multiple vSwitch nodes and the OpenPATH switching framework.

7.6.1. Effect of different NF complexity

Each test case runs traffic through 2 VNFs instances and we expect OpenPATH overhead to vary across the test cases, depending on the complexity of the application logic. Fig. 32 plot the (a) latencies and (b) throughputs for the 7 test cases/7 NFs. We observed that the optimization level increases with more complex network functions and processing. e.g., the Forwarder NF has simplified instruction cycles and processing overhead is less, whereas IDS/VPN NFs consist of heavier processing, instructions and consequently OpenPATH's parallelism effect is profound.

Effect of Packet Sizes: We plotted in Fig. 33 (a) how the latencies of NF processing vary inside the vSwitch node, for the network packet sizes (64–1500 bytes). We chose one exemplar VNF - Forwarding Function - that can have the lowest number of CPU cycles and least data operations on the packet fields and expected to perform closest to wire-rate. As shown in Fig. 32 (c), OpenPATH achieves a wire-rate, close to 9.6 Gbps across all packet sizes.

7.6.2. Effect of parallelism degree

We measured the effect of the number of parallel NFs that the OpenPATH can facilitate. To test this function, the test configuration deployed IDS NF on multiple VNFs in both sequential/parallelized chain. The traffic packets were 64 bytes size and both copy-less/copy-packet policies were tested. The results in Fig. 33 (b) and (c) show that the optimization effect raises with the parallel instances, proving that OpenPATH performs well with a higher parallel degree. The latencies reduce by 55% for copy-less, and by 35% for copy-packet operations, while throughput numbers remain the same across the NF degrees.

7.6.3. Optimization effect with-respect-to graph structure

With a given set of NFs, different structures of the graph can be generated. Fig. 34 illustrates an example set of graphs for the same service chain. The experiments revealed improved latencies for graphs with short chains. In the same experiment, the graph structure (2) measured the lowest latencies, because the chain length is 1. Graph (5) measured the highest as the length was 03.

7.7. Management and control operations overhead

The main benefit of OpenPATH was to reduce the complexity of flow tables. But because flow tables differ due to network configurations, policies and traffic conditions, it is challenging to generalize flow rules. There is no guarantee of effectiveness for this particular use case or topology, but we do assume benefits close to a traditional NFV deployments could be reached.

For traditional SDN based NFV implementation, we took the leaf-based topology from *DPX* (Park et al, 2019) and built it out into two layers: connecting the end nodes to the switches as depicted in Fig. 35. A node in the leaf fabric is a NFV host that handles NF service. Additionally, we measure the flow table size that is needed to talk to all of the NFV service chain components (including path traversal). Fig. 36 shows that the number of necessary flow rules exponentially increases as the number of nodes in the NFV/SDN platform grows and as the length of the service chain increases. The network will need around 1280 rules when the leaf nodes reach 8 and the SFC length 4.

OpenPATH will bypass the SDN controller with an enhanced data-plane (seen in Fig. 37), as the NFs (NFV)are located in the SDN data plane. Therefore, the OpenPATH network does not need any flow rules to route the flows to the final destination. We also contend that poor NFV efficiency on the traditional SDN/NFV platforms is caused by a combination of associated re-processing overhead on the NFV and switching overheads. Connecting an additional NFV system can lengthen the service chain. Policies that have the ability to be articulated in an OpenPATH-architecture flow-rule framework may reduce both data plane and network functions. For any SFC length and complexity there are fewer flow rules required. For example, even for service chain length 4/16 hosts, OpenPATH requires approximately 128 rules to ensure communications between all and every link in the network. The OpenPATH architecture also lowers the number of NFV packet management functions, while simplifying complex network and SFC policies are accessible from a single table.

7.7.1. Flow table installation and controller load

We increase the load on the network by flooding it with new connections/flows, triggering a burst. Switches forward packet in messages to the controller via the OpenFlow channel, and the controller responds with "flow modification" messages to the switches. As a result, the controller's CPU and network port will be consumed by processing these "new-flows." This is a critical performance criterion for SDN-enabled networks. The controller on the OpenPATH system (See Fig. 38(a)) maintained the number of "flow installations/second" when the number of vSwitch nodes increased in our tests (See Fig. 38(b)). With a Classic-SDN OvS switch, the controller was overloaded by new-flow flooding and finally crashed due to CPU/Memory overload.

7.7.2. Control channel/southbound interface load optimization

OpenPATH's expanded SDN architecture has a minimum controller overhead compared to typical OpenFlow-based SDN. The data plane processing does not cause any extra overhead in the switches' typical OpenFlow pipeline. Using microbenchmarks, we investigated the controller overhead (traffic flow from the data plane to the control plane) and the redirection ratio (traffic flow to DPI and the other NFV modules) to get valuable insight into how to optimize the design. Only TCP packets, which account for 77% of the input traffic, are forwarded to the controller. In OpenPATH, the augmented switches will retrieve events using DPI capabilities, and ~1.5 percent traffic flows get through the southbound interface and analyzed for security policy violations or

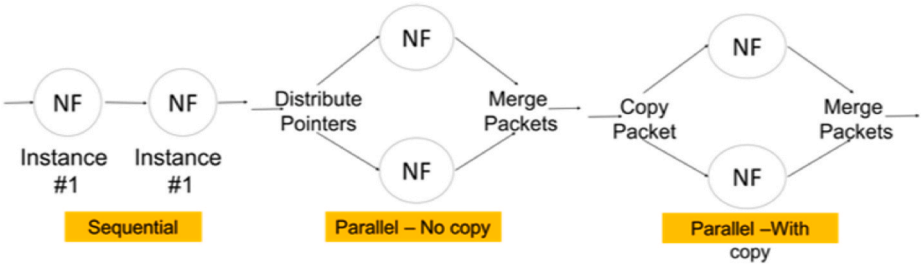

Fig. 31. Three modes of NFV Configuration.

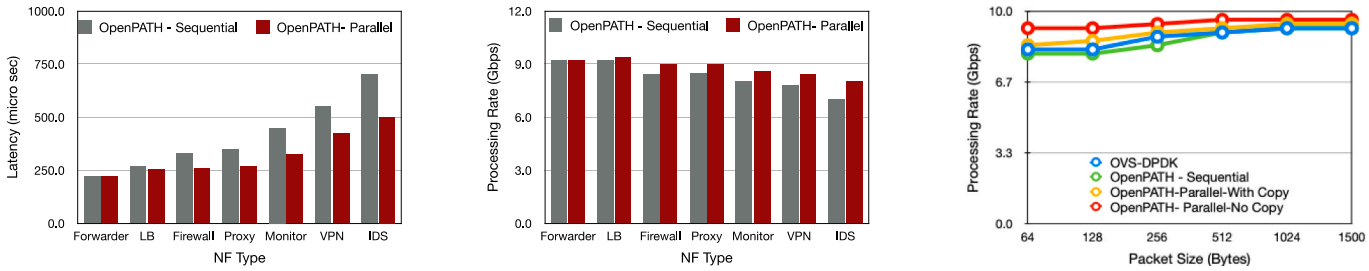

Fig. 32. (a) NF Complexity Latency (b) NF Throughput (c) Throughput Effect for packet sizes.

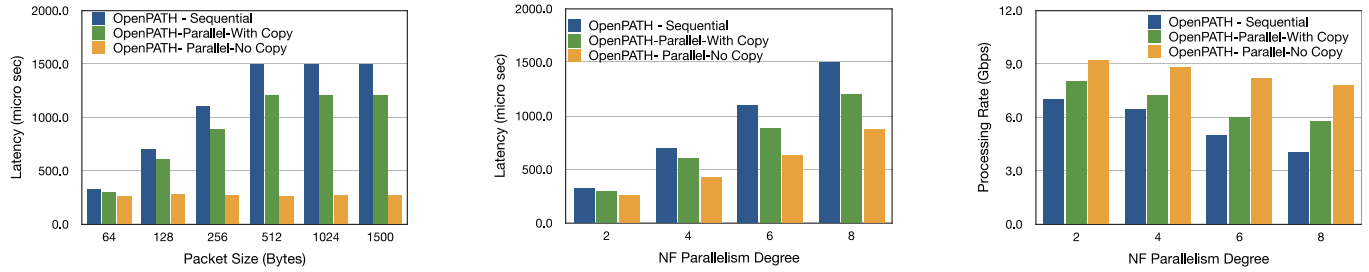

Fig. 33. (a) Latency Effect for packet sizes (b) Latencies varying parallelism degree (c) Processing Rate with parallelism degree.

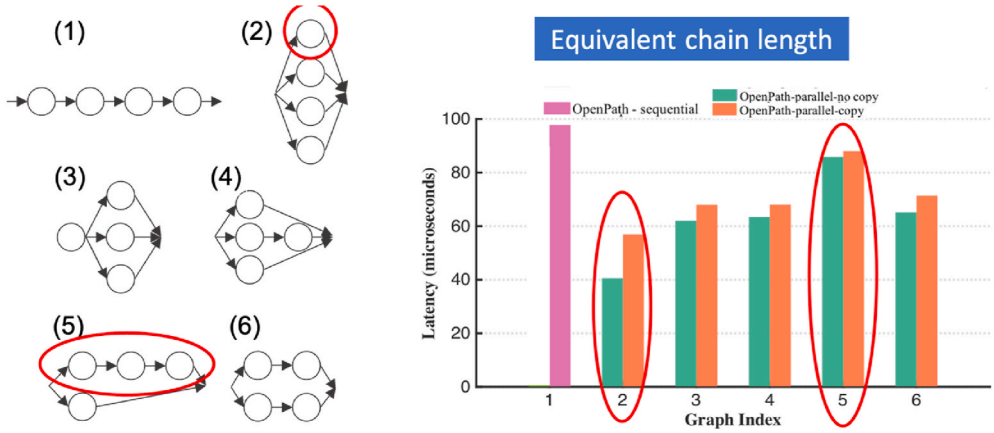

Fig. 34. Performance with different Graph structure.

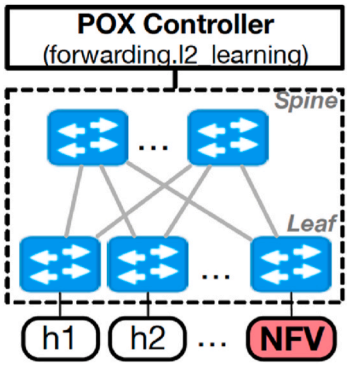

Fig. 35. NFV Platform with traditional SDN.

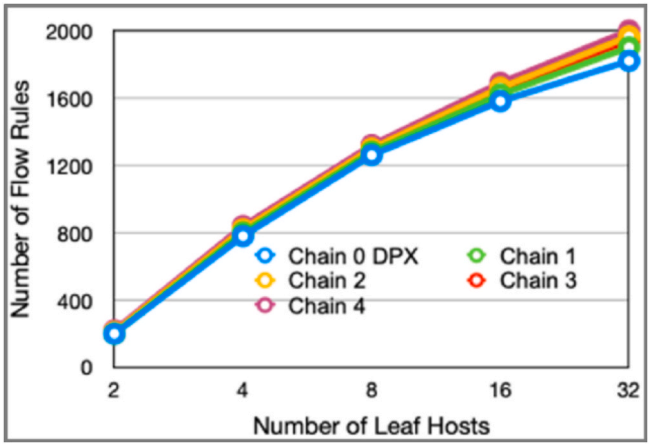

Fig. 36. Flow Table update load.

suspicious cases. (See Fig. 38 c). The dataplane is responsible for the majority of the classification processing workload. Additionally, only necessary traffic will be forwarded to the controller and the ratio varies depending on the traffic composition.

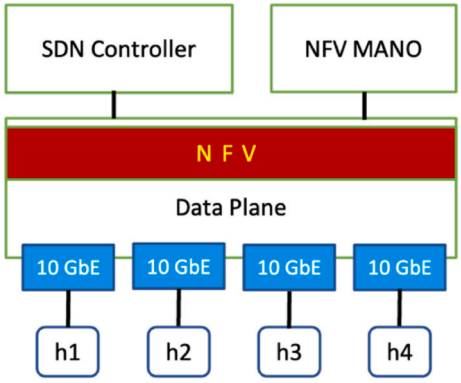

Fig. 37. NFV platform in OpenPATH.

7.8. SFC orchestration in large simulated network

To better explain packet steering (routing), we compare it to two use cases: 1. Data Center deployments, 2. Research networks. We mainly focus on two data center architectures: a leaf-spine and a fat-tree (referred to as a Clos topology here). We evaluate the convergence time for the routing algorithm, end-to-end overhead for new path setup and the control operation overhead due to the flow table installation across the switches in the data plane. We used Mininet (Lantz et al., 2010) to simulate large network topologies. We emulate a typical 4-pod Fat-tree network topology (Fig. 39), the elements in the topology are termed hierarchically as: core, aggregation and edge switches. In a Fat-tree topology, all core switches connect to all pods. In our network, we set up 16 servers, four pods (each with four switches), and four core switches. Thus, there is a total of 20 switches and 16 end hosts (for larger clusters, the number of switches will be smaller than the number of hosts).

We built and compared NFV implementation under following three platforms:

1) Quagga's (Quagga Routing Suite, 2019) OSPF routing protocol suite with virtual Linux switch(es)
2) Floodlight (Big Switch Networks, 2016) as a conventional SDN stack with traditional stateless OpenvSwitch dataplane
3) OpenPATH SDN stack with controller and vSwitch nodes with modified OVS replacing the Pods. The pods act as 'vSwitch' (consolidating aggregate and edge switches), host the NFV applications and service function chains. The nodes (equivalent to *pods* in OSPF setup) have capabilities for light-weight applications and deep-packet inspection logic. ii) high-speed control channel with the controller. iii) sufficient memory (TCAM) for maintaining OpenFlow group tables and stateful data plane pipeline. iv) backup/fail-safe for controller failures. The Controller offloads these Core switches certain functions and delegates flow table management and routing.

7.8.1. network convergence time for topology construction

We evaluate the scalability of the routing protocols with respect to network topology and size. The convergence principle is different for traditional OSPF routing protocol and in SDN flow establishment. For

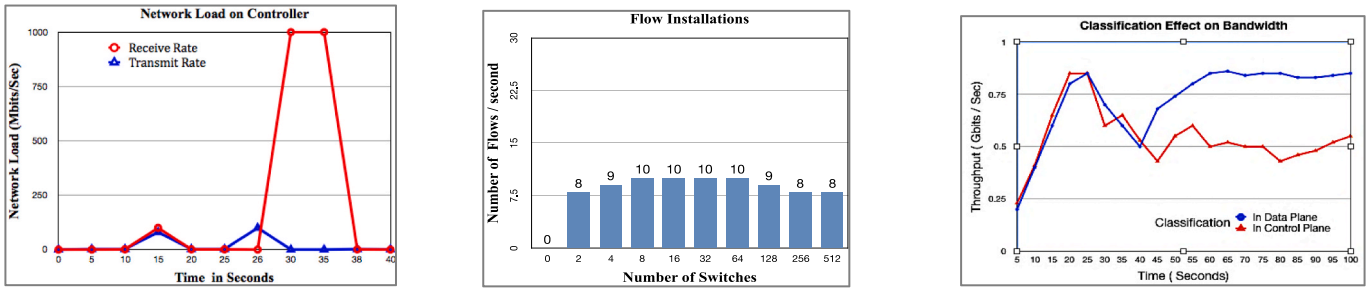

Fig. 38. (a) Network Load on controller, (b) Flow Installation Scaling, (c) Southbound Interface Optimization.

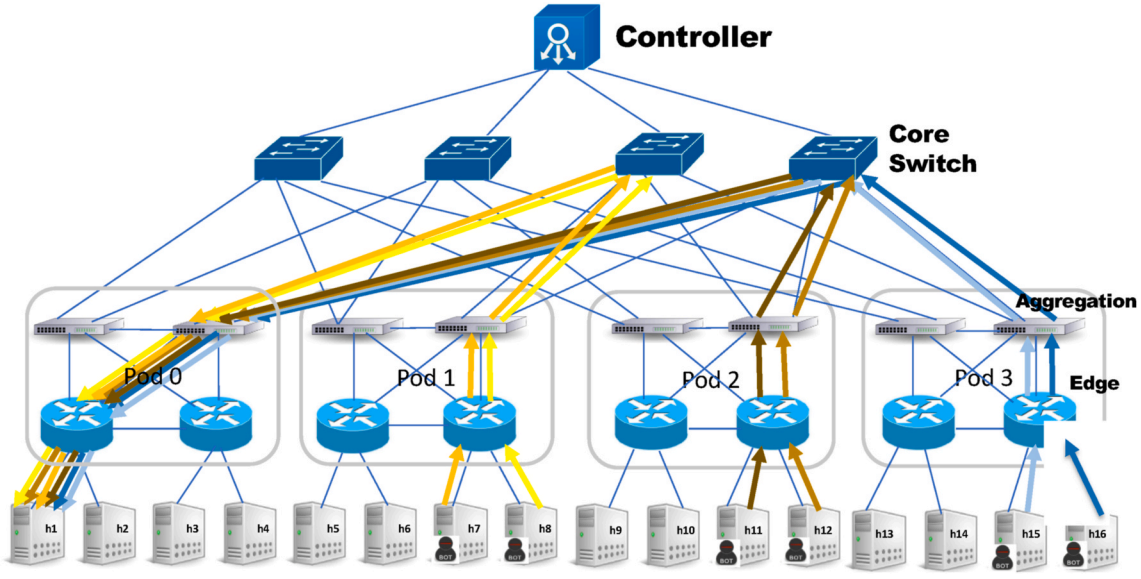

Fig. 39. Testbed network Architecture with Fat Tree topology (Clos).

the Quagga's distributed OSPF configuration-the time for all hosts in the network to receive packets from all other hosts (i.e.," routing tables in all the switches in all paths are determined and populated") in the network. For Floodlight SDN, as part of the OpenFlow principle, the initial topology across all the switches is discovered by the controller through LLDP protocol. The convergence time is measured as the time taken by the controller to install flow tables for all the devices in the network (including the switches). In OpenPATH which has stateful data plane, through the out-of-band control connection with the controller and core switches, the bi-directional flow rules are delegated to the vSwitch nodes. So, the convergence is considered complete when the first packet (new flow) is received by the destination host. Fig. 40 shows that for the network topology (K = 4/8/16), the convergence in OpenPATH is faster than OSPF/LLDP- Floodlight. For OpenPATH, the main overhead for convergence is for establishing the OpenFlow out-of-band secure connection between the control plane and data plane switches and delegating the group of flow-tables to the switches, all of these incurs lesser overhead. For Quagga's-OSPF & Floodlight-SDN, the time for convergence increases with the network size, as more routing protocol messages are shared between switches and controllers ("e.g., LSAs/DBs in OSPF and Floodlight LLDP"). Also, the topology detection mechanism is a heavy weight protocol and has high demands from SDN controllers, which makes this a bottleneck for the network startup, when compared to the OpenPATH which is built on the same SDN principle but with the stateful notion.

7.8.2. End-to-end routing delay for SDN

In this experiment, we tested with the only two platforms (Floodlight and OpenPATH) that use the SDN OpenFlow paradigm. We used different traffic models in data centers to evaluate the Round-Trip-Time (RTT) performance: Ping ("One-to-One, One-to-All, and All-to-All"). Fig. 41 shows that in the three separate traffic modes, the delay for OpenPATH is below that of Quagga's Floodlight SDN. The rationale is that the SFC algorithm for OpenPATH SFC forwarding works better than the Floodlight SDN Dijkstra's Spanning Tree algorithm. The routing paths/forwarding decision are determined on switches locally and with no intervention by the controller.

7.8.3. Control message overhead

During the execution of all the above experiments, we observed the control messages that are exchanged out-of-band on the secure channels between the controller and switches, switches down the tier all the way to the leaf nodes. In OSPF specification, "Message type:xx" and in OpenFlow SDN standards – "Message type: Hello/Packet-In/Packet-Out, Proto Type: LLDP" are some examples. We collected the packet dump on the control connection for a specific sampling period and gathered insight from the packets.

The key findings from the experiments in this section:

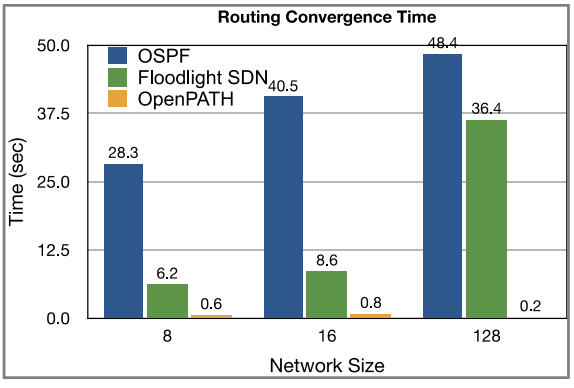

Fig. 40. Topology construction time.

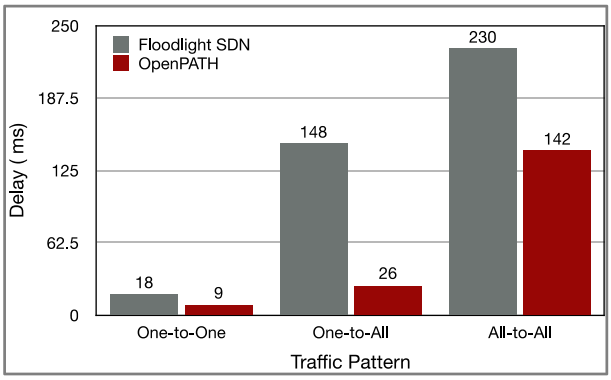

Fig. 41. Flow establishment time.

- OSPF converges in 50 s and Floodlight SDN takes 10 s. Once the network is converged, both Floodlight and OSPF will be steady and quiet. During the whole operation (including convergence) there is a modest increase in the number of maintenance/control messages, but Floodlight shows big spike.
- The frequency of "Hello" messages in OSPF- 10 s, Floodlight's LLDP- 15 s, OpenPATH -every 1 s. The majority of OpenPATH's control communication are "Hello" messages, which is more than others. In OpenPATH, "Hello" traffic is exchanged solely between the switches and is not transmitted to the controller, resulting in no additional burden on the controller. So, switches and controllers' CPU and network resources are thus lowered.
- From a control message overhead viewpoint, OpenPATH generates consistent traffic and seldom encounters bursty traffic. Due to the increased usage of "Hello", OpenPATH is able to detect network topology changes and errors more quickly.

7.9. Application-aware SDNFV architecture

In this experiment, we discuss the benefits of the Flow Analysis function implemented in the SDNFV stack, for monitoring and dynamic applications. The southbound protocols such as OpenFlow and sFlow are standard network monitoring and management wire protocols that are supported across all vendor devices. The flow analyzer engine classifies and tags the flows for priority queuing, with minimal impact on throughput for large flows, while greatly improving latency for small flows. The NF classifies the flows based on their phase change, burst intervals, packet sizes, and dynamically adjusts paths through *Change-Path* message.

Flow-based QoS Engine: This Engine monitors the live flows to determine their traffic behavior. It divides traffic flows into different buckets by measuring the size, rate, and interval in packets arrivals. A Traffic shaping/slicing mechanism will classify elephant flows and latency-sensitive Mice flows traffic. After the detection, the SFC flow-tables output "action" is reconfigured to provide a faster-processing path for Ant/Mice and higher bandwidth for elephant flows. We used *Pktgen*-DPDK to emulate different flows and plotted the flows recorded in Fig. 42. By rapid classification of traffic and detection of anomalies in the network, the dynamic flow-analyzer engine installs an action to optimize the latencies for Ants/Mice flows, as shown in Fig. 42(a). At the same time, the bandwidth for the Elephant Flows is increased due to lesser traffic contention as shown in Fig. 42(b).

Dynamic DDoS Detector: We evaluated the effectiveness of the OpenPATH flow-analyzer and dynamic NF insertion for SFC, in an emulated network that is under mixed attacks. Multiple traffic generator applications are run from within the network and through the gateway, flooding the network with mixed protocol attacks targeting the servers and just filling the pipe. The Smart Monitoring mechanisms that are built natively in the OpenPATH dataplane will find anomalies and

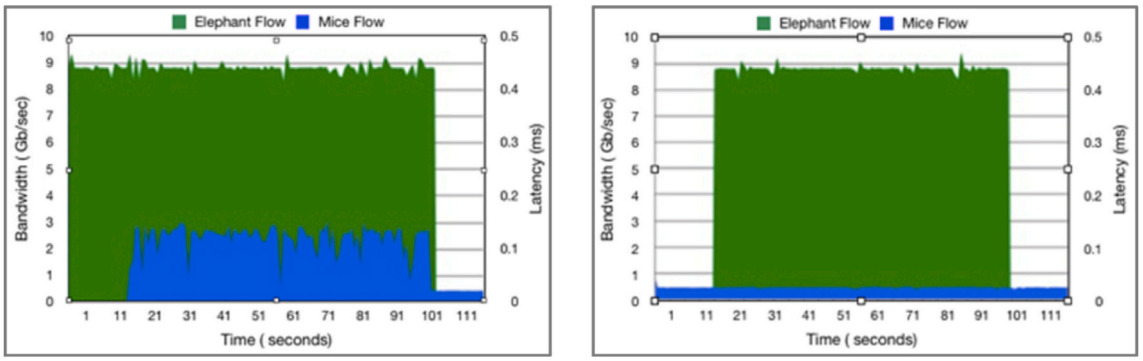

Fig. 42. Performance of Flow Analyzer (a) Actual plot of flows (b) Plot with Flow-based bandwidth shaping.

malicious traffic patterns in the flows and will raise alarm events to the NF manager. The IPS and Defensive firewall appliances are installed with policy filters and dynamic policies/thresholds for reacting to network attacks at run-time. We also run traffic from normal benign clients to measure the impact of malware attacks on the network and how OpenPATH responds to the incidents. Fig. 43 plots the network throughput for the traffic. The attack begins around 10 s into the experiment, and we can see that the OpenPATH detects and responds to the attack in 35 s and the normal service is restored. While we observe small differences in reaction times to different types of attacks (e.g., TCP, UDP, DDoS, etc.), the overall recovery time is still reasonably short.

Dynamic SFC Expansion: In the following test case we demonstrate how OpenPATH makes modifications to the service-graph at run-time to respond to the situation. The dataplane defines packet sequence-based flows but also has historical data for correlating, drawing insight, and finding anomalous patterns in the network traffic. We configured an NFV chain for an Edge Protection/Firewall system (Fig. 12) in which there is a Volumetric DoS detection IDS, filters packets based on the count of packets(threshold) carrying similar IP segment. (Indicating the origin of a DoS attack). The VNF alerts NF-Manager to ADD new VNF in the chain when packet drops increase over the threshold. Consequently, the new DDoS Sandbox VNF is added in the chain, which re-routes/detours the packets from the DDoS VNF after breaching the threshold to the Sandbox for further scrubbing. The Sandbox VNF inspects the packets for attacks and decides to mark benign/forward or malign/drop the packets. This experiment demonstrates the efficiency and autonomous operation of the NFV system in the OpenPATH dataplane, saving all the complexity and overhead on the control plane. Fig. 44 plots the traffic for a certain period in the network. We generate benign traffic @10 Gbps. At time#10s, we begin the attack (mixed volumetric traffic with the same IP-Prefix) @1 Gbps and gradually ramped up to heavily

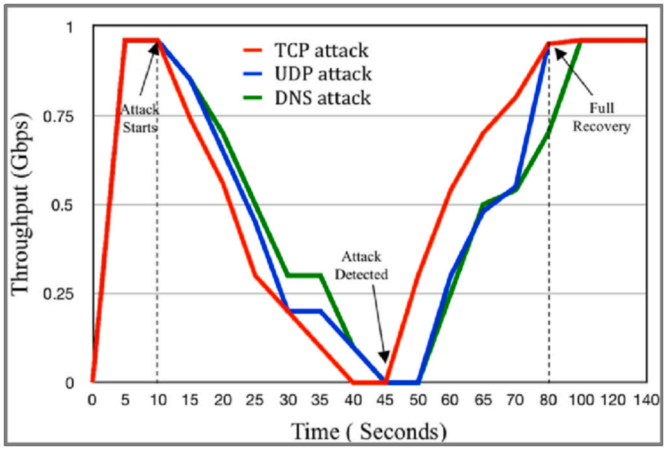

Fig. 43. Dynamic DoS detection.

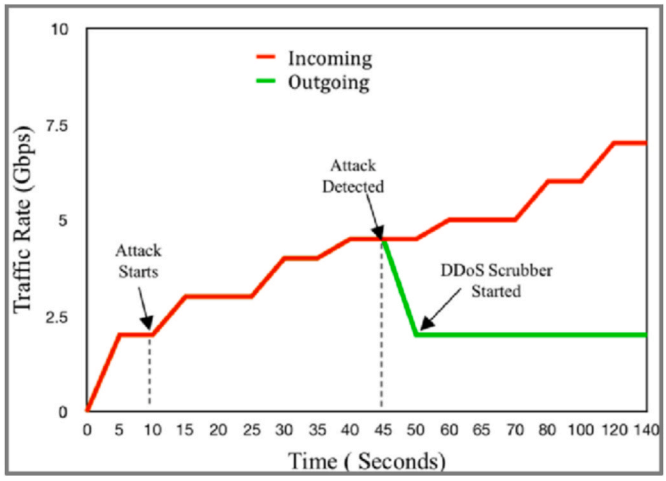

Fig. 44. Dynamic SFC expansion.

hit the network. The DDoS Detector NF monitors the flow patterns in the packets passing through the chain and records the suspicious flows. Using the simple threshold-based IDS, when the traffic reaches (4.8 Gbps), the IDS raises an alarm to mark those flows as malicious and detoured for further scrubbing. The IDS VNF raises the alarm to NF Manager and NFV-MANO for any change in action for the particular VNF. The NFV-MANO installs a command to insert a Sandbox VNF in the SFC to handle the detoured suspicious packets. This VNF was booted in under 4 s (could be improved by having redundant VNFs in sleep mode and woken up at run time quickly). It can be observed from Fig. 44 at time#30s, the outbound traffic rate is restored (Sandbox dropped the attack packets), even as traffic increases inbound.

7.10. Comparison of SDNFV software switch performance

To test NFV configurations, there are four different scenarios employed, the "physical to physical (p2p), physical to virtual (p2v), virtual to virtual (v2v), and loopback. All of the four case studies are depicted in Fig. 45. We believe the performance indicators and scenarios outlined above and these four use cases are crucial for NFV. When integrated with SDN/NFV, an in-depth understanding of the performance characteristics of any network virtualization solution is possible. The latest software-based NFV platforms use "kernel bypass" and zero-copy I/O ("NETMAP, DPDK"). The state-of-the-art software NFV platforms utilizing software switches, that implements kernel bypass and high-speed I/O mechanism on DPDK, NETMAP. BESS and OvS-DPDK aim to provide the benefits of SDN (i.e., separation of data and control planes) with the flexibility of a software solution and highly optimized data paths. We have chosen to do a comparison study of software switch

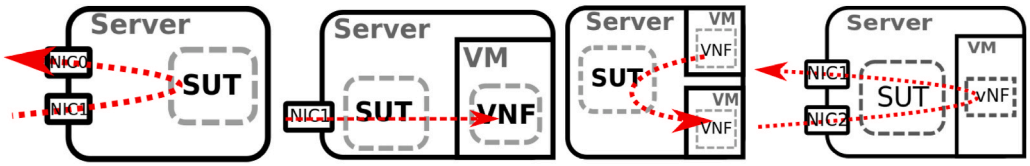

Fig. 45. Test scenarios - physical-to-physical (p2p), physical-to-virtual (p2v), virtual-to-virtual (v2v), loopback.

based NFV platforms - OvS-DPDK (Open, 2019), FastClick (Barbette et al., 2015), BESS (Han et al., 2015b) and VPP (VPP FD.io, 2016) with our solution OpenPATH.

The aim of this experiment is to demonstrate and discuss the throughput sustained by software switches in four test scenarios involving simulated bi-directional traffic and packet sizes of 64, 256, and 1024 Bytes (Fig. 46).

(a) **p2p**: BESS, FastClick, and VPP all surpass 5 Gbps for 64B packets. BESS exceeds 7.5 Gbps when performing very basic tasks such as data collection. FastClick extracts and updates packet header fields, while VPP performs a variety of checks. Due to the overhead imposed by its match/action pipeline, OvS-DPDK achieves a lower cost. Due to the fact that the synthetic traffic is composed of identical packets that correspond to a single flow, OvS-flow DPDK's *cache* is ineffective.

(b) **v2p**: BESS and FastClick are able to maintain line rate for 256B and 1024B packets, but the effect of *vhost-user* is noticeable on the other switches. We observe a small degradation in throughput for VPP, as it incurs a performance penalty when receiving packets from *vhost-user* ports.

(c) **v2v**: In contrast to other cases, v2v is restricted only by memory bandwidth, illustrating the upper limit of inter-VM communication via software switches and packet copying via *virtio* rings for the switches.

(d) **Loopback**: OpenPATH achieves the highest throughput rate for all packet-sizes, even as the NF-chain length increases (default is 1-VNF) due to distributed optimized packet steering and zero-copy shared memory for I/O processing. BESS gives high rate for the single-VNF chain. However, as the service chain lengthens due to the addition of VMs, BESS must conduct an increasing number of packet copies and packet detouring/trips through the central switch. FastClick and VPP operate at a slower rate due to the bottlenecks and packet-copying in the "*vhost-user*".

In all the scenarios, the performance for the compared NFV platforms is affected mainly due to the non-optimal I/O, packets copying, across the stack and buffers, multiple contexts switching within the node and as the number of NFs increase the packet touring back and forth between the VMs through the switch. Even for the OVS-DPDK switch, it incurs detouring and concurrent processing overhead (two streams) which lowers the throughput. In OpenPATH, with the NFs already embedded in the single node, we save all the detouring overheads and the other resource optimization techniques such as zero-copy have played key role in achieving maximum practical sustained peak throughput in all scenarios. FastClick and VPP work well in all cases due to *VirtIO* mechanism. BESS performs well in all cases except in multi-VNFs loopback

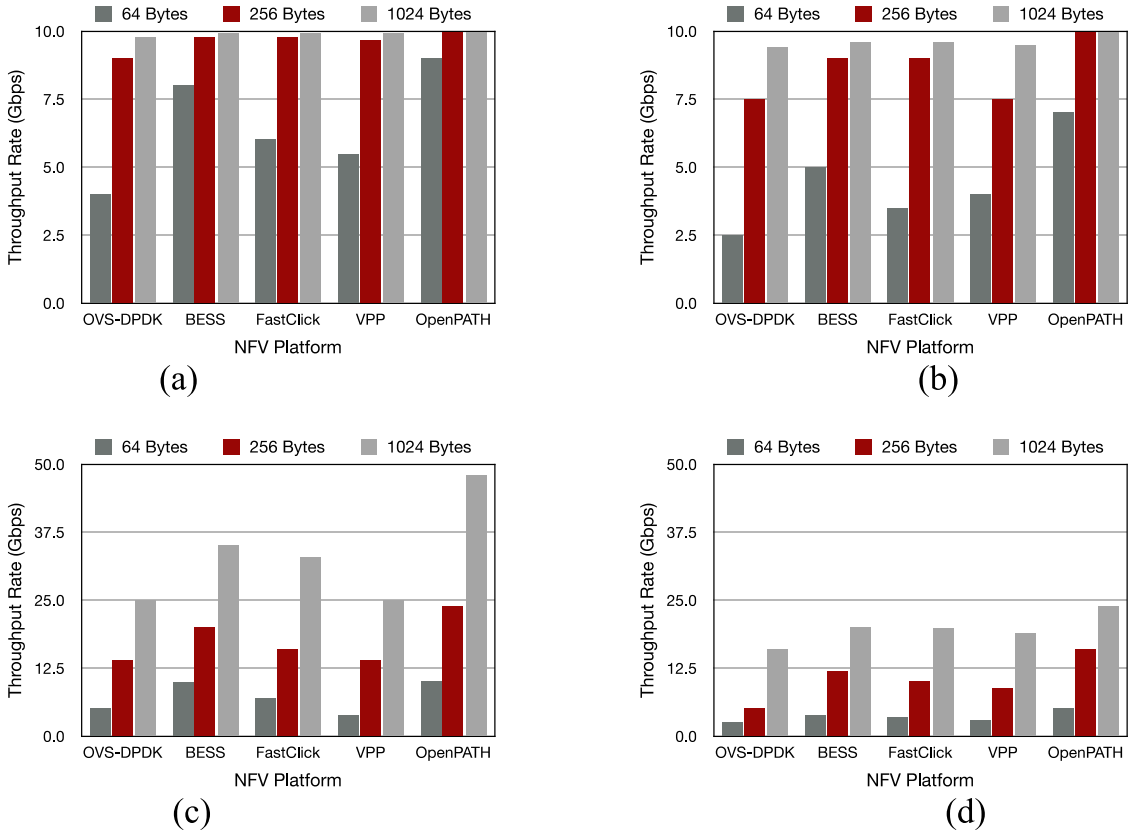

Fig. 46. Throughput in (a) p2p, (b) p2v, (c) v2v and (d) loopback scenarios.

Table 8
Software switches use cases summary.

	Best at	Remarks
OvS-DPDK (Open, 2019)	Stateless SDN deployments	Supports OpenFlow protocol and P4, works with third-party SDN controllers and newly introduced protocols
BESS (Han et al., 2015b)	Forwarding between physical NICs and one or multiple paralleled VMs	Natively provides scheduling capabilities, Incompatible with newer versions of QEMU and scalability issues with VMs
FastClick (Barbette et al., 2015)	VNF chaining, linear and parallel NFV environments with reasonable trade-offs.	Supports live migration, high latency at low workload, Modular high-speed router design
VPP (VPP FD.io, 2016)	VNF chaining, linear and parallel NFV environments with lowest latencies	Supports live migration, For Fully featured software network function (e.g., switch, router, or security Appliance)
OpenPATH	Intent based VNF Chaining, high-performance linear and parallel NFV environments, line rate NF (middlebox) application and SFC processing.	Supports OpenFlow protocol and P4, works with third-party SDN controllers and newly introduced protocols, preferable when some state is required (e.g., for a firewall). All programmable SDN/NFV infrastructure.

configuration. We have summarized the salient aspects of some of the popular software switches in Table 8.

7.11. Comparison of server-based NFV platform performance

We compare OpenPATH's end-to-end performance against customized, high-performance NFV platforms OpenNetVM, NFVNice and Metron. We explore how OpenPATH's performs when running an NF chain of various lengths and varying function complexity.

About the compared NFV Platforms:

- NFVNice (Kulkarni et al., 2017) – A VM based NFV platform that supports isolation with packet copying technique to the master module that acts as the message bus for transmitting packets between processes. has similar performance (95%) with a single-NF chain, but as chain length increases its throughput decreases despite its extra CPU cores for transmitting packets.
- Metron (Katsikas et al., 2018) - high performance NFV platform that compiles NFs into a single process and runs-to-completion each chain as a thread. It implements a runtime and scaling algorithm. Each Metron runtime is a single thread process that runs an NF chain. Packets are transmitted between NFs with no isolation mechanisms.
- OpenNetVM (Zhang et al., 2016a)- employs fast and zero-copy I/O through shared-memory between NFs. VNFs are run as independent processes on VMs. It runs the central software switch (bridge) on a dedicated CPU core for packet forwarding.

7.11.1. Throughput Effect with varying SFC length

We focus on a per-packet processing throughput for 64 B packets with a single server for OpenPATH and server-based NFs. For simplicity, we use a test NF (a firewall IDS module with 200 rules) and run chains with a sequence of the same NF. Fig. 47 shows the throughput of different NFV approaches for different length chains. OpenPATH achieves more than 95% of the peak rate across all packet sizes, for NFV processing. The key findings are:

- OpenPATH throughput remains steady regardless of the chain length.
- In legacy NFV platforms (NFs on VMs) the processing rate and throughput decrease as we increase the NFs in the chain. The bad performance is due to the non-optimal I/O, packets copying, across the stack and buffers, multiple contexts switching within the node and as the number of NFs increase the packet touring back and forth between the VMs through the switch. The legacy OVS switch involves detouring and concurrent processing overhead (two streams) which lowers the throughput.
- In OpenPATH switch, with the NFs already embedded in the single node, we save all the detouring overheads and the other resource optimization techniques such as zero-copy have played key role in achieving maximum practical sustained peak throughput even with scaling of the chain.
- OpenPATH outperforms all other server-based NFV platforms with line-rate throughput for all types of traffic.

7.11.2. Latency Effect with varying NF complexity

We compared the processing overhead with 2 instances of the same NFs configured in a chain. The testbed uses 7 VNFs in NFV nodes. Each test case runs traffic through 2 VNFs instances and we expect the overhead to vary across the test cases, depending on the complexity of the application logic. The key findings (See Fig. 48) are:

- Latency benefits due to NF parallelism in OpenPATH increases with NF complexity. For the least complex NF (300 cycles) to most complex NF (3000 cycles)

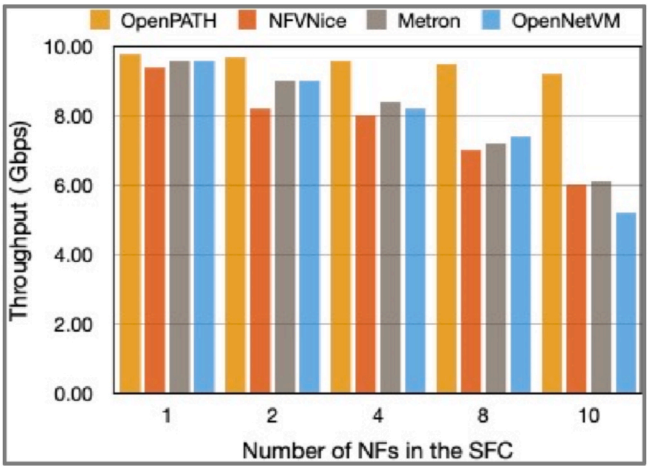

Fig. 47. Throughput vs chain length.

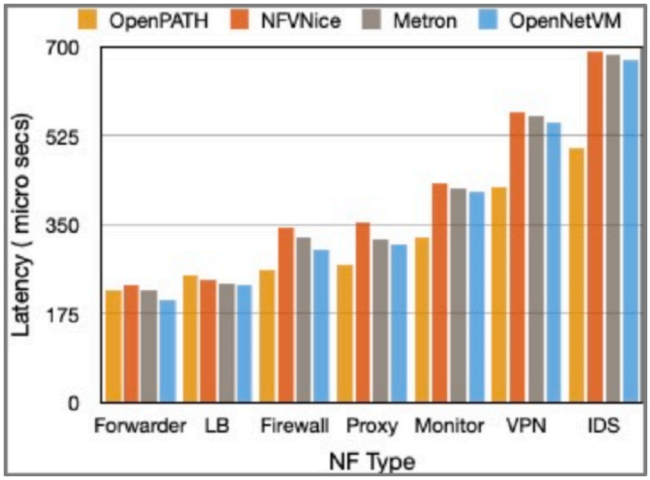

Fig. 48. Latency vs. NF complexity.

- The optimization level increases with more complex network functions and processing. e.g., the Forwarder NF has simplified instruction cycles and overhead is less, whereas IDS/VPN NFs consist of heavier processing, instructions.
- OpenPATH's optimization effect is profound due to NF parallelism and zero-copy/minimal packet steering overhead in the pipeline and brings about 45% latency reduction compared with other platforms.

7.12. Result discussion

Network functions (NFs), or middleboxes, are widely common in large-scale networks. These appliances are usually very expensive, hard to manage, to scale, and to provision. They usually perform complex packet processing tasks, and thus, play a major role in the overall network performance. SDN and NFV work together to build a highly scalable network with a lower operational cost for data centers. The current way of connecting network functions to an existing network, however, introduces new problems such as duplicating operations between various network functions, overlapping packet data sharing in packet processing, and unnecessary network traffic with co-existing service chains. An NFV-enabled SDN architecture is proposed here to solve these challenges by merging duplicated processes, the fast packet I/O, parallel processing, and avoiding traffic detours in the software-defined networks. Table 9 presents the evaluation strategy we followed to test various design aspects and key findings from our experiments under micro benchmarks, comparative test cases, and application scenarios, on SDN/NFV testbed infrastructure.

8. Future work

The modern virtualized/SDNFV data centers demand an agile/programmable switching platform that can encompass the complete distributed network for high-speed packet processing and routing in the complex network function/service chains. Network functions and middleboxes are an active area of research. The inherent limitations of current packet processing technologies, and the growth in bandwidth requirements, require a rethinking of network functions in all aspects, from traffic engineering aspects such as placement and steering, through operational aspects as scaling, provisioning, and security, to functional aspects such as programming abstractions, runtime I/O packet processing frameworks, and improved algorithms for specific tasks. Addressing some of these issues, OpenPATH enables NFV with the SDN architecture. In this research direction, we are pursuing a few open issues and a sketch of future work are listed here.

- Enriching the scope of the policy definition templates for more complicated service graphs in SFC, also tools to inspect and verify policies. Expanding the OpenPATH to *container* platforms as they facilitate portability, elastic scaling, and fast boot-up of instances. Applying *Graph partitioning* techniques and sub-graphs, *map/reduce* computing model. To evaluate the NF dependencies and conflicts when we transform into sub-graphs for parallelism, we must develop an automated formal method to define diverse functionalities and complex policies (Sekar et al., 2012).
- The heterogeneity of network functions is a motivating factor for hardware acceleration and this aspect is not dealt in this article. We will design outsourcing of NFs to hardware GPU and NIC accelerators by providing an OpenCL-like interface for COTS hardware (e.g., GPU, Intel Phi, Netronome AgilioTM CX, NetFPGA, Intel FlexPipe, Barefoot Tofino). Some work is in progress to implement the architecture presented in a proof-of-concept hardware prototype using an FPGA platform. The hardware prototype is being designed using as target development board NetFPGA SUME (Zilberman et al., 2014), an x8 Gen3 PCIe adapter card incorporating a Xilinx Virtex-7 690T FPGA.

Table 9
Summary of the key findings.

ASPECTS EVALUATED	DISCUSSION
NF placement	The processing rate or throughput is higher in OpenPATH compared to the unoptimized "back-and-forth" policy. The output is almost the same as the single-node case (as the fabric connection is not crossed multiple times). Similarly, the production of easy forwarding tests is typically flat at around 9.4 Mb/s, suggesting that adding NF Manager has limited effects on the efficiency of medium to heavy NFs.
NF Scalability	OpenPATH consolidates NFs in a service graph inside one server to optimize resource overhead. Load balancing with two NF nodes is identical to load balancing with a single node. When adding replicas, we use a different node. The throughput increases linearly up to the rate of the input line (10 Gb/s). With a number of instances ranging from one to ten, throughput is just marginally slower than with a single server, despite the addition of the chaining server, fabric connections, and a separate NF placement.
Flow Table Management	The flow entries can have compact representation called *synapse* of packet length and inter-time packet arrival distributions which is small yet provides enough information to perform accurate application-specific traffic classification. This provides a significant reduction in the size of the flow entries for OpenFlow communication, reducing the bandwidth usage, less memory space for flow tables, which in turn yields a reduction in power dissipation for core switches.
Packet Steering management	In OpenPATH, for steering the packets across the VNFs in the chain the NF Manager delegates the packets to NF handlers attached to them. The packets are accessed by using references to shared memory buffers and alleviates the hot spot in packet forwarding.
Algorithmic Complexity	The smart placement single-node algorithm (See Fig. 7) consumes O(n) operations to select optimal nodes. So, the complexity can be O ($2n + n \log n$). The multi-node placement (See Fig. 9) O(Ln) operations (length L as a constant) to traverse the graph. The worst-case complexity takes $O(Kn^2)$. Thus, we can infer the approach has reasonable scalability even for large complex service graphs/SFC.
Resource overhead (Flow table updates, packet copying overhead)	To minimize copying overhead, OpenPATH makes use of DPDK's zero-copy interfaces. For a complex NF, it can incur an average of 100 micro secs of latency and a negligible throughput penalty while still achieving a 35% latency reduction over sequential composition. The latency overhead percentage of copying and merging will be further reduced with longer chains and more complex NFs (e.g., VPN). This framework provides throughput improvements such as 67%, and reduces latency by 35%, in the scenarios tested in NF parallel configurations (copying and no-copy, degrees and packet sizes.
Modularity	OpenPATH decomposes NFs into independent building blocks. This characteristic is exploited for both monolithic and modular SFC, for block-level parallelism, to further reduce latency.
Processing Rate	In the Conventional method, the throughput is bounded by the slowest NF in the pipeline. With OpenPATH, the SDNFV orchestrator merges the NFs into subgraphs and executes in parallel. The overall improvement in throughput is 2x and the latency is 50%, as packets are processed parallelly by one of the vSwitch nodes.
State Management	As the NF state is distributed across multiple nodes, the SDN controller initializes the tables across the nodes and periodically synchronizes the state tables. This problem and the management overhead are not unique to OpenPATH, and we solve this issue efficiently through "Statefulness" in the data plane.

- The NFV deployment solution has to ensure security (i.e., authenticity, integrity of VNFs) and reliability (i.e., order of services) in the SDN infrastructures. Despite the advantages of NFV/SDN for SFC implementation, major protection and reliability problems exist such as i) Difference among high-level SFC policies and their data plane compliance. ii)ensuring that the authentic VNFs are assembled in the proper and optimal order when the actual traffic flows through the SFC.
- The Consistency issues in SFC, analysis of attack models to deviate the SFC path (e.g., flow redirection) are still a major concern and relatively neglected by the research community. The issue is how do we ensure that packet flows linked to a particular SFC are steered and traverse through appropriate and legitimate VNFs in relation to the predefined policies.
- We are examining the protection and explain how security signature aggregation techniques can preserves the security, reliability and resistance to the attacks defined in the common threat models for the SDN/NFV enabled data centers.
- We have to investigate light-weight attestation mechanisms to verify the behavior of packet traversals across the SFC chain and resistance to attacks like flow diversion, adversarial rerouting and unauthorized path de-touring.

While we did our best to make the study as comprehensive and thorough as possible, we expect to discover only a small part of the iceberg. Even the findings presented here should inspire further investigation of OpenFlow and SDN assumptions for NFV.

9. Summary and conclusions

SDN/NFV technologies are converging, evolving, maturing and new players are entering the market and we have proposed a flexible and high-performance SDN/NFV switching framework. OpenPATH aims at completely leveraging virtualizing-based network features in enhancing network capabilities through an integrated hierarchical control and stateful flow management system to resolve the constraints of a stateless dataplane that are over-simplified in the current SDNs. We've maintained the spirit of the SDN paradigm's simplified-programmable model by keeping the concept flexible and extensible. The proposed architecture can be used to implement NFV/middlebox functions as well as stateful applications. Our findings demonstrate that OpenPATH provides superior performance over other switching architectures in NFV services infrastructure. More significantly, OpenPATH is developed to be conformant to the OpenFlow standards and can be integrated as a datapath (without recompile) in the existing SDN-enabled environment. OpenPATH provides an effective intuitive and expressive interface to the operators for semantic representation of the policies, priorities in SFC, sequential or parallel NFV composition intent. Our experiment results show that OpenPATH based NFV deployments can reduce latency by about 55% and increase throughput by up to 70%. With the stateful dataplane design, OpenPATH infrastructure provides a robust SDN switching stack, without overloading the control channel. The added intelligence in dataplane ensues far more dynamism and accuracy when comparing to the legacy approaches in SDN which employ proactive rules and static flow management. Through this work, we have established the need for new research directions in software switching and discussed the design strategies associated with hosting the NFV services in the next generation converged SDNFV architectures.

Authors credit statement

The following is the statement of contributions for this research paper: **Prabhakar Krishnan** devised the project, the main conceptual ideas, Methodology, Software and worked out almost all of the technical details, design, implementation of the research and performed the experiments. **Subhasri Duttagupta** helped supervise the project critical feedback and helped shape the research, analysis and contributed to the interpretation of the results. **Prabhakar Krishnan** took the lead in writing the manuscript in consultation with **Subhasri Duttagupta. Rajkumar Buyya** provided critical feedback and helped shape the research, analysis and manuscript. All authors discussed the results and contributed to the final manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Alim, Abdul, Clegg, Richard G., Luo, Mai, Rupprecht, Lukas, Seck-ler, Eric, Costa, Paolo, Peter, Pietzuch, Wolf, Alexander L., Sultana, Nik, Crowcroft, Jon, et al., 2016. Flick: developing and running application-specific network services. In: 2016 USENIX Annual Technical Conference (USENIX ATC 16).

Anwer, Bilal, Benson, Theophilus, Feamster, Nick, Levin, Dave, Rexford, Jennifer, 2013. A slick control plane for network middleboxes. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13. ACM, New York, NY, USA, pp. 147–148.

Arashloo, Mina Tahmasbi, Koral, Yaron, Greenberg, Michael, Rexford, Jennifer, Walker, David, 2016. SNAP: Stateful Network-wide Abstractions for Packet Processing. SIGCOMM.

Barbette, Tom, Soldani, Cyril, Mathy, Laurent, 2015. Fast userspace packet processing. In: 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, pp. 5–16.

Bianchi, G., Bonola, M., Capone, A., Cascone, C., 2014. OpenState: programming platform-independent stateful OpenFlow applications inside the switch. Comput. Commun. Rev. 44 (2), 44–51.

Bianchi, G., Bonola, M., Pontarelli, S., Sanvito, D., Capone, A., Cascone, C., 2016. Open Packet Processor: a Programmable Architecture for Wire Speed Platform-independent Stateful In-Network Processing. arXiv preprint arXiv:1605.01977.

Bifulco, R., Rétvári, G., 2018. A survey on the programmable data plane: abstractions, architectures, and open problems. In: 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), Bucharest, Romania, pp. 1–7.

Big Switch Networks, 2016. Project Floodlight - Floodlight OpenFlow Controller. http://www.projectfloodlight.org/floodlight/. (Accessed 17 September 2016).

Bosshart, P., et al., 2014. P4: programming protocol-independent packet pro- cessors. Comput. Commun. Rev. 44 (3), 87–95.

Bray, T., 2017. The Java Script Object Notation(JSON) Data Interchange Format (RFC8259). https://tools.ietf.org/html/rfc8259.

Bremler-Barr, A., Harchol, Y., Hay, D., 2016. OpenBox: A software- defined framework for developing, deploying, and managing net- work functions. In: Proc. SIGCOMM.

Cai, J., Huang, Z., Luo, J., Liu, Y., Zhao, H., Liao, L., 2020. Composing and deploying parallelized service function chains. J. Netw. Comput. Appl. 163, 102637.

Csikor, L., Szalay, M., Sonkoly, B., Toka, L., 2015. NFPA: network function performance analyzer. In: IEEE NFV-SDN. Demo Track, pp. 17–19.

Dong, Y., Yang, X., Li, X., Li, J., Tian, K., Guan, H., 2010. High performance network virtualization with SR-IOV. In: HPCA - 16 2010 the Sixteenth International Symposium on High-Performance Computer Architecture, pp. 1–10.

European Telecommunications Standards Institute(ETSI), 2014. Network Functions Virtualisation (Nfv). White Paper.

Fang, Vivian, Lvai, T., Han, Sangjin, Ratnasamy, Sylvia, Raghavan, Barath, Sherry, Justine, 2018. Evaluating software switches: hard or hopeless?. In: EECS Department, University of California, Berkeley Tech. Rep. UCB/EECS-2018-136 (2018).

Fayaz, S.K., Tobioka, Y., Sekar, V., Bailey, M., 2015. Bohatei: flexible and elastic ddos defense. In: 24th USENIX Security Symposium (USENIX Security 15). USENIX Association, Washington, D.C., pp. 817–832

Fayazbakhsh, Seyed Kaveh, Sekar, Vyas, Yu, Minlan, Mogul, Jeffrey C., 2013. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13. ACM, pp. 19–24.

Feamster, Nick, Rexford, Jennifer, Zegura, Ellen, 2013. The road to sdn. Queue 11 (12), 20:20–20:40.

Fei, Xincai, Liu, Fangming, Zhang, Qixia, Jin, Hai, Hu, Hongxin, 2020. Paving the Way for NFV acceleration: a taxonomy, survey and future directions. ACM Comput. Surv. 53 (4), 42. https://doi.org/10.1145/3397022. Article 73 (August 2020).

Gallenmüller, S., Emmerich, P., Wohlfart, F., Raumer, D., Carle, G., 2015. Comparison of frameworks for high-performance packet IO. In: Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS), Oakland, CA, USA, pp. 29–38.

Gember, A., Krishnamurthy, A., John, S.S., Grandl, R., Gao, X., Anand, A., Benson, T., Akella, A., Sekar, V., 2013. Stratos: A network-aware orches- tration layer for middleboxes in the cloud. CoRR abs/1305, 209.

Gember-Jacobson, Aaron, Viswanathan, Raajay, Prakash, Chaithan, Grandl, Robert, Khalid, Junaid, Das, Sourav, Akella, Aditya, 2014. OpenNF: enabling innovation in network function control. In: ACM Conference on Special Interest Group on Data Communication (SIGCOMM), Chicago, IL.

Han, S., Jang, K., Park, K., Moon, S., 2010. Packetshader: A gpu-accelerated software router. In: Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM '10. ACM, New York, NY, USA, pp. 195–206.
Han, B., Gopalakrishnan, V., Ji, L., Lee, S., 2015a. Network function virtualization: challenges and opportunities for innovations. IEEE Commun. Mag. 53 (2), 90–97. https://doi.org/10.1109/MCOM.2015.7045396.
Han, Sangjin, Jang, Keon, Panda, Aurojit, Palkar, Shoumik, Han, Dongsu, Ratnasamy, Sylvia, 2015b. BESS SoftNIC: A Software NIC to Augment Hardware. Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA. Tech. Rep. UCB/EECS-2015-155 (2015).
H wang, Jinho, Ramakrishnan, K.K., Wood, Timothy, 2014. Netvm:High performance and flexible networking using virtualization on commodity platforms. In: Proceedings of the 11th USENIX Conference on Net- Worked Systems Design and Implementation, NSDI'14. USENIX Association, pp. 445–458.
Han, W., et al., 2016. State-aware network access management for software- defined networks. In: Proc. 21st ACM Symp. Access Control Models Technol. SACMAT, Shanghai, China, pp. 1–11.
Honda, Michio, Huici, Felipe, Lettieri, Giuseppe, Rizzo, Luigi, 2015. mSwitch: a highly-scalable, modular software switch. In: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. ACM, 1.
Indigo. Indigo - open source OpenFlow switches [Online]. Available: http://www.projectfloodlight.org/indigo/.
Iperf. https://iperf.fr/.
Jackson, E.J., Walls, M., Panda, A., Pettit, J., Pfaff, B., Rajahalme, J., Koponen, T., Shenker, S., 2016. SoftFlow: a middlebox architecture for Open vSwitch. In: Proc. USENIX ATC.
Joseph, D.A., Tavakoli, A., Stoica, I., 2008. A policy-aware switching layer for data centers. In: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, ser. SIGCOMM '08. ACM, New York, NY, USA, pp. 51–62.
Kaljic, Enio, Maric, Almir, Begovic, Pamela, Hadzialic, Mesud, 2019. A survey on data plane flexibility and programmability in software-defined networking. IEEE Access 7, 47804–47840. https://doi.org/10.1109/ACCESS.2019.2910140.
Katsikas, Georgios P., Barbette, Tom, Kostíc, Dejan, Steinert, Rebecca, Maguire Jr., Gerald Q., 2018. Metron: NFV service chains at the true speed of the underlying hardware. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). USENIX Association, pp. 171–186.
Kim, H., Feamster, N., 2013. Improving network management with software defined networking. IEEE Commun. Mag. 51 (2), 114–119.
Kim, J., Huh, S., Jang, K., Park, K., Moon, S., 2012. The power of batching in the click modular router. In: Proceedings of the Asia-Pacific Workshop on Systems, APSYS '12. ACM, New York, NY, USA, pp. 14:1–14:6.
Kohler, Eddie, Morris, Robert, Chen, Benjie, Jannotti, John, Kaashoek, M Frans, 2000. The Click modular router. ACM Trans. Comput. Syst. 18 (3), 263–297, 2000.
Krishnan, P., Achuthan, K., 2019. Managing network functions in stateful application aware SDN, security in computing and communications. In: SSCC 2018. Communications in Computer and Information Science, vol. 969. Springer, Singapore. https://doi.org/10.1007/978-981-13-5826-5_7.
Krishnan, P., Duttagupta, S., Achuthan, K., 2019a. VARMAN: multi-plane security framework for software defined networks. Comput. Commun. 148, 215–239. https://doi.org/10.1016/j.comcom.2019.09.014.
Krishnan, P., Duttagupta, S., Achuthan, K., 2019b. SDNFV based threat monitoring and security framework for multi-access edge computing infrastructure. Mobile Network. Appl. 24, 1896–1923. https://doi.org/10.1007/s11036-019-01389-2.
Kulkarni, S.G., Zhang, W., Hwang, J., Rajagopalan, S., Ramakrishnan, K., Wood, T., Arumaithurai, M., Fu, X., 2017. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. ACM, pp. 71–84.
Lantz, B., Heller, B., McKeown, N., 2010. Mininet:A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In: Proc. ACM Hot Topics, pp. 19:1–19:6.
Lettieri, G., et al., 2017. A Survey of Fast Packet I/O Technologies for Network Function Virtualization. *ISC Workshops*.
Li, Bojie, Tan, Kun, Luo, Layong Larry, Peng, Yanqing, Luo, Renqian, Xu, Ningyi, Xiong, Yongqiang, Cheng, Peng, 2016. ClickNP: highly flexible and high-performance network processing with reconfigurable hardware. In: Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference. ACM, pp. 1–14.
Liu, A.X., et al., 2010. TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs". IEEE/ACM Trans. Netw. 18 (2).
Martins, Joao, Ahmed, Mohamed, Raiciu, Costin, Olteanu, Vladimir, Honda, Michio, Bifulco, Roberto, Huici, Felipe, 2014. Clickos and the art of network function virtualization. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). USENIX Association, pp. 459–473.
McKenney, P.E., Slingwine, J.D., 1998. Read-copy-update: using execution history to solve concurrency problems. In: Parallel and Distributed Computing and Systems, pp. 509–518.
McKeown, N., et al., 2008. OpenFlow: enabling innovation in campus networks. In: ACM SIGCOMM CCR 38.2.
Medved, J., Varga, R., Tkacik, A., Gray, K., 2014. Opendaylight: towards a model-driven sdn controller architecture. In: A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th Inter-national Symposium on. IEEE, pp. 1–6.
Mekky, H., Hao, F., Mukherjee, S., Lakshman, T., Zhang, Z.L., 2017. Network function virtualization enablement within sdn data plane. In: IEEE INFOCOM, pp. 1–9.
Mekky, Hesham, Fang, Hao, Mukherjee, Sarit, Zhang, Zhi-Li, Lakshman, T.V., 2014. Application-aware data plane processing in sdn. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, pp. 13–18. New York, NY, USA.
Molnár, et al., 2016. ESwitch: Dataplane Specialization for High-Performance OpenFlow Software Switching. ACM SIGCOMM, New York, NY, USA, pp. 539–552. https://doi.org/10.1145/2934872.2934887.
Moshref, M., Bhargava, A., Gupta, A., Yu, M., Govindan, R., 2014. FAST: flow- level state transition as a new switch primitive for SDN. In: Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN), Chicago, IL, USA, pp. 61–66.
Nasr, M., Houmansadr, A., Mazumdar, A., 2017. Compressive traffic analysis: a new paradigm for scalable traffic analysis. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, pp. 2053–2069.
Network service chaining problem statement [Online]. Available: https://tools.ietf.org/html/draft-%20quinn-%20nsc-%20problem-%20statement-%2003.
Network service header [Online]. Available: https://tools.ietf.org/html/draft- quinn- sfc-nsh- 07.
ntop. PFRING. http://www.ntop.org/products/packet-capture/pf_ring.
ONF Solution Brief, 2014. Openflow-enabled Sdn and Network Fatunctions Virtualization.
Open vSwitch with DPDK. http://docs.openvswitch.org/en/latest/intro/install/dpdk/, 2019.
OpenWrt Wiki [Online].Available: https://wiki.openwrt.org/start.
Osiński, T., Tarasiuk, H., Chaignon, P., Kossakowski, M., 2020. P4rt-OVS: programming protocol-independent, runtime extensions for open vSwitch with P4. In: 2020 IFIP Networking Conference (Networking), pp. 413–421.
Palkar, Shioumik, Chang, Lan, Han, Sangjin, Jang, Keon, Panda, Aurojit, Ratnasamy, Sylvia, Rizzo, Luigi, Scott, Shenker, 2015. E2: a framework- work for nfv applications. In: Proceedings of the 25th Symposiumsium on Operating Systems Principles, SOSP '15. ACM, pp. 121–136.
Panda, Aurojit, Han, Sangjin, Jang, Keon, Walls, Melvin, Ratnasamy, Sylvia, Scott, Shenker, 2016. NetBricks: taking the V out of NFV. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), vol. 16. USENIX OSDI, pp. 203–216.
Paolino, Michele, Nikolaev, Nikolay, Fanguede, Jeremy, Raho, Daniel, 2015. Snabb Switch user space virtual switch benchmark and performance optimization for NFV. In: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN). IEEE, pp. 86–92.
Park, Taejune, et al., 2019. DPX: data-plane eXtensions for SDN security service instantiation. DIMVA. https://doi.org/10.1007/978-3-030-22038-9_20.
Pfaff, B., et al., 2015. The design and implementation of open vSwitch. In: USENIX NSDI '15. USENIX Association, USA, pp. 117–130.
Pica8, 2013. Pica8 3920 [Online]. Available: http://www.pica8.org/documents/pica8-datasheet-64x10gbe-p3780-p3920.pdf.
Qazi, Z.A., Tu, C.-C., Chiang, L., Miao, R., Sekar, V., Yu, M., 2013. SIMPLE- fying middlebox policy enforcement using SDN. In: Proc. SIGCOMM.
Quagga Routing Suite. Accessed: Nov. 16, 2019. [Online]. Available: https://www.nongnu.org/quagga/index.html.
Quinn, P., Elzur, U., Sep. 2016. "Network Service Header," Internet Engineering Task Force. Internet-Draft draft-ietf-sfc-nsh-10 work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc- nsh-10.
Rizzo, Luigi, 2012. netmap: a novel framework for fast packet I/O. In: USENIX Annual Technical Conference. USENIX, Berkeley, CA, pp. 101–112.
Rizzo, Luigi, Lettieri, Giuseppe, 2012. Vale, a switched ethernet for virtual machines. In: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies. ACM, pp. 61–72.
Sekar, Vyas, Egi, Norbert, Ratnasamy, Sylvia, Reiter, Michael K., Shi, Guangyu, 2012. Design and implementation of a consolidated middlebox architecture. In: NSDI, pp. 323–336.
Service function chaining general use cases [Online]. Available: https://tools.ietf.org/html/draft- liu- sfc- use- cases- 08.
Service function chaining (sfc) architecture [Online]. Available: https://tools.ietf.org/html/draft- merged- sfc- architecture- 02.
Sherry, J., Ratnasamy, S., 2012. A Survey of Enterprise Midlebox Deploy- Ments. Technical report, Technical Report No. UCB/EECS-2012-24.
Sonchack, J., Smith, J.M., Aviv, A.J., Keller, E., 2016. Enabling practical software-defined networking security applications with OFX. In: NDSS.
Sonchack, J., Michel, O., Aviv, A., Keller, E., Smith, J., 2018. Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow. In: Proceedings of the USENIX Annual Technical Conference, Boston, MA, USA, pp. 823–835.
Srinivasan, V., Suri, S., Varghese, G., 1999. Packet classification using tuple space search. In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '99. ACM, New York, NY, USA, pp. 135–146.
Sun, Chen, Bi, Jun, Zheng, Zhilong, Yu, Heng, Hu, Hongxin, 2017. NFP: enabling network function parallelism in NFV. In: Proceedings of SIGCOMM '17, USA. https://doi.org/10.1145/3098822.3098826.
Tahhan, Maryam, O'Mahony, Billy, Morton, Al, 2017. Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV). RFC 8204. (Sept. 2017). https://doi.org/10.17487/RFC8204.
VPP FD.io, 2016. The Fast Data Project. project website.
Wang, M., Cheng, B., Chen, J., 2020. Joint availability- and traffic-aware placement of parallelized service chain in NFV-enabled data center. In: 2020 IEEE International Conference on Web Services (ICWS), pp. 216–223. https://doi.org/10.1109/ICWS49710.2020.00035.
Xie, Sihao, Ma, Junte, Zhao, Jin, 2021. FlexChain: bridging parallelism and placement for service function chains. IEEE Trans. on Netw. and Serv. Manag 18 (1), 195–208. https://doi.org/10.1109/TNSM.2020.3047834. March 2021.

Zhang, Wei, Liu, Guyue, Zhang, Wenhui, Shah, Neel, Lopreiato, Phil, Todeschi, Gregoire, Ramakrishnan, K.K., Wood, Timothy, 2016a. OpenNetVM: a plat- form for high performance network service chains. In: 2016 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization.

Zhang, Tianzhu, et al., 2019. Comparing the performance of state-of-the-art software switches for NFV. In: The 15th International Con Ference on Emerging Networking EXperiments and Technologies (CoNEXT '19). ACM, pp. 68–81.

Zhang, Yang, Anwer, Bilal, Gopalakrishnan, Vijay, Han, Bo, Reich, Joshua, Shaikh, Aman, Zhang, Zhi-Li, 2017. ParaBox: exploiting parallelism for virtual network functions in service chaining. In: Proceedings of the Symposium on SDN Research. ACM, pp. 143–149.

Zhang, Wei, et al., 2016b. SDNFV: Flexible and Dynamic Software Defined Control of an Application- and Flow-Aware Data Plane, Middleware'16. https://doi.org/10.1145/2988336.2988338.

Zilberman, N., Audzevich, Y., Covington, G., Moore, A., 2014. NetFPGA SUME: Toward 100 Gbps as research commodity. Micro, IEEE 34 (5), 32–41.

Dr. Prabhakar Krishnan is a research scientist and is currently a PhD Scholar at the Department of Cyber Security Systems and Networks at Amrita Vishwa Vidyapeetham, India. He has over two decades of Industry experience in the USA. His current research interests are primarily in Cybersecurity, with special focus in designing network security and architecture, Network Softwarization, SDN/NFV, Cyber Forensics and IoT standardization. He was a visiting adjoint professor with the Department of Computer Science, University of Texas at San Antonio, USA. He is a working member of the core-development group of OpenAirInterface Software Alliance(OSA) 5G Wireless community at Eurecom in Europe. He is a member of the IEEE and the IEEE Computer Society.

Dr. Subhari Duttagupta has received her M. Tech. degree in 1993 in the area of Parallel Computing and received the doctoral degree in 2010 in the area of sensor networks, both from Indian Institute of Technology-Bombay. Between 1994 and 2002, she worked in various organizations such as IBM, Micron and HP in USA. **Areas of interest**: Performance Evaluation And Modelling Of Systems And Networks, Distributed Systems, Real-Life Applications Using Sensor Networks And Analysing IoT Applications, Internet of Things, Performance Engineering, Modelling and Simulation.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored over 725 publications and seven textbooks including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 137, g-index = 304, 100000+ citations). He is a Fellow of the IEEE.