Computer Networks 167 (2020) 106975

Contents lists available at ScienceDirect

Computer Networks

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

KeySFC: Traffic steering using strict source routing for dynamic and efficient network orchestration

Cristina K. Dominicini[a,*], Gilmar L. Vassoler[a], Rodolfo Valentim[b], Rodolfo S. Villaca[b], Moisés R.N. Ribeiro[b], Magnos Martinello[b], Eduardo Zambon[b]

[a] Federal Institute of Education, Science and Technology of Espírito Santo (IFES), Brazil
[b] Federal University of Espírito Santo (UFES), Brazil

ARTICLE INFO

Article history:
Received 2 March 2019
Revised 7 October 2019
Accepted 28 October 2019
Available online 12 November 2019

Keywords:
SFC
NFV
SDN
Traffic steering
Source routing
Orchestration,

ABSTRACT

Current service function chaining (SFC) solutions are cumbersome adaptations of classic routing mechanisms, which lack flexibility and agility to cope with new dynamic services required by network functions virtualization (NFV). These SFC solutions require modification of forwarding tables in both physical and virtual elements in the path when updating a chain. In addition, current SFC implementations restrict traffic engineering to sub-optimal resource allocation solutions, because limited switch table sizes prevent the consideration of all possible traffic paths. Moreover, overlay chaining decisions are usually decoupled from the actual underlay routing of packets across service functions. To tackle these issues, we propose KeySFC, a traffic steering scheme that uses software-defined networking (SDN) and strict source routing. KeySFC exploits the fabric network concept in data center networks (DCNs) in two ways: (i) edge software switches classify, encapsulate, forward, and decapsulate flows with SFC labels; and (ii) core tableless switches forward packets based on simple modulo operations over these labels. Thus, the modification of a small number of flow entries in edge elements allows changing of SFC labels and effectively stitching paths via SDN. An OpenStack-based prototype demonstrates that the traffic steering scheme provided by KeySFC has the potential to enable efficient traffic engineering and to provide agile path migration per SFC segment.

1. Introduction

Emerging trends such as 5G, Internet of Things, Smart Cities, and Industry 4.0 require the dynamic composition of end-to-end services in data centers (DCs) that can meet stringent performance, cost, and flexibility requirements. In this scenario, one of the main challenges in network functions virtualization (NFV) is how to enable the dynamic composition of highly customized services by steering a large number of flows across a set of service functions (SFs) to provide service function chaining (SFC) [1].

It is worth noting that SFC changes the traditional end-to-end routing paradigm in DC networks (DCNs), because it requires the dynamic insertion of virtualized nodes between source and destination: packets from source are steered to a host and delivered to a virtualized SF to be processed; then, packets return to the network and are routed to the next SF, repeating these steps until they reach the destination [2]. This is a far more complex challenge than routing between two end nodes, since it involves capturing, classifying, and steering the traffic for each virtualized SF.

To provide composite services, a service overlay topology is built "on top" of the existing network underlay topology [3]. To ensure such services achieve maximum performance, the network controller or distributed agents in charge of traffic engineering should be able to select among all possible paths in the underlay network according to dynamic demands [4], when deciding the routing paths between SFs.

However, there are fundamental problems in traditional SFC approaches that compromise efficient network orchestration. Firstly, most of the current works perceive the network as a mere way to provide connectivity for the overlay. Thus, these SFC solutions are frequently decoupled from the routing decisions on the underlay [5], and leave to the routing mechanisms the responsibility of deciding how to deliver packets between SFC segments. Secondly, routing mechanisms of traditional DCNs are not designed for these dynamic and composite service requests, because they are usually complex, rigid, and subject to large propagation delays of control information [6–8]. Thirdly, the number of states is limited by the size of forwarding tables in switches [9], and thus traffic

* Corresponding author.
E-mail address: cristina.dominicini@ifes.edu.br (C.K. Dominicini).

https://doi.org/10.1016/j.comnet.2019.106975
1389-1286/

engineering is usually restricted to a set of shortest paths between SFC endpoints. This may prevent the orchestrator from selecting non-shortest paths to avoid congestion or faulty paths [10].

Therefore, the traffic engineering of SFC requests is restricted to sub-optimal solutions, because current traffic steering mechanisms do not allow: (i) to explore all network capacities of the underlay topology when mapping the service overlay; and (ii) to explicitly select from all existing paths per chain segment and to agilely modify these paths for variable demands.

To tackle these issues, we propose a new traffic steering scheme, named KeySFC. To make optimal use of network capacity for dynamic SFC demands, we argue that the solution is to replace tables on the routing stage by strict source routing (SSR). In this paradigm, the source (or ingress node) can specify all network elements in the forwarding path to the destination, and each node in the path can forward packets based on some information inserted by the source in the packet header [11]. Therefore, SSR decreases forwarding table sizes, facilitates maintenance, reduces the control plane overhead to configure paths, and supports optimal throughput performance when compared to per-hop table-based approaches [4,6,7,9,12–15].

KeySFC extends the idea of network fabric [6,16] to intra-DC server-based networking, creating a clear separation between: (i) edge elements that provide programmable SFC classification, and (ii) core elements that execute efficient packet transport using a topology-independent SSR mechanism based on the Residue Number System (RNS) [6]. In this scheme, SDN-enabled software switches classify flows for SFC and embed an identifier in the packet header that encodes the information about all forwarding elements in the path. At core switches, the forwarding engine executes a *modulo* operation over the path identifier to discover the output port.

The contributions of this paper can be summarized as follows:

- A novel traffic steering scheme that offers the following features to network orchestration: (i) selection of any available path, (ii) specification of all forwarding elements in the path, and (iii) agile path migration. As a result, network orchestration can integrate chaining and routing decisions to make optimal use of networking and computing resources in response to dynamic demands.
- Support for *any* topology by the traffic steering scheme, including server-centric and hybrid DCNs [17]. This contrasts to traditional solutions that focus only on network-centric topologies.
- Implementation of a proof-of-concept testbed orchestrated by OpenStack, validating the KeySFC proposal and demonstrating its feasibility in production DCs. Results indicate that the selected RNS-based SSR mechanism can provide efficient packet transport.

This paper is organized as follows. Section 2 defines the traffic steering problem. Then, Section 3 discusses how RNS-based SSR can be applied to traffic steering, and Section 4 presents the KeySFC proposal. Afterwards, we describe our proof-of-concept implementation and its evaluation in Section 5. Then, Section 6 compares our solution with related works. Finally, we outline the conclusions and future works in Section 7.

2. The traffic steering problem

This section describes the SFC workflow, the problem of embedding SFC requests in the network infrastructure, and the existing gaps in traffic steering mechanisms of the underlay network to deploy resource allocation solutions. Afterwards, we define the research question investigated by this work.

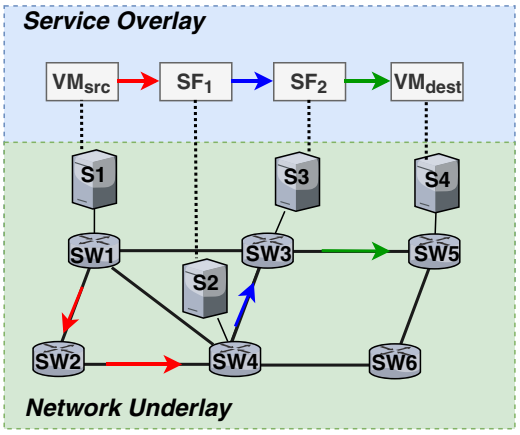


Fig. 1. SFC embedding: service overlay mapped into underlay network.

2.1. SFC workflow

Fig. 1 shows the embedding of an SFC request in the NFV Infrastructure using the concept of service *overlay* layer and network *underlay* layer.

The overlay is represented by a graph that defines the logical connections between virtual machines (VMs) in SFC. This graph is known as the *virtual network function forwarding graph* (VNF-FG), defining a connection chain of SFs in which the order is important. The underlay is represented by a topology graph with links between physical nodes. We consider that each hop in the overlay is an SFC segment, which can be mapped to zero or more hops in the underlay. Thus, an end-to-end network service can be described as a VNF-FG interconnected by a network infrastructure underlay [18].

For example, Fig. 1 shows a VNF-FG that has three SFC segments: $VM_{src} \rightarrow SF_1$, $SF_1 \rightarrow SF_2$, and $SF_2 \rightarrow VM_{dest}$. The VMs of this chain are placed into servers S1, S2, S3, and S4, respectively. The first segment is mapped to two hops in the underlay network (SW1 → SW2 and SW2 → SW4), whereas the second and third segments are mapped to a single hop: (SW4 → SW3), and (SW3 → SW5), respectively.

In the ETSI NFV reference architecture [18], one of the main blocks is the Management and Orchestration (NFV MANO), which has the following components: NFV Orchestrator (NFVO), VNF Manager (VNFM), and Virtualized Infrastructure Manager (VIM). The NFV MANO block communicates with an SDN Controller. Fig. 2 shows how these main components interact in the SFC workflow [19].

Initially, the NFVO receives service requests from customers and the service modeling generates SFC description models for the VNF-FGs. After that, the NFVO gathers information about the physical infrastructure, including server resource usage and network status, to define the available capacity of processing nodes and the network. Based on this information, the NFVO makes two decisions: (i) placement – deciding the optimal location of SFs in physical servers, considering service requirements and resource constraints [20]; and (ii) SFC – deciding how to steer traffic flows across an ordered set of SFs that compose the service [1].

These decisions are then distributed to the VIM and VNFM for resource allocation, which includes the placement of SFs and determination of paths between them. In the next step, SFC descriptions and the selected paths are distributed to the SDN controller, which translates the policy requirements into rules for traffic steering in the data plane. Finally, the service is fully provisioned to the customers.

The traffic steering mechanisms (highlighted in blue in Fig. 2) are the focus of this work, and refer to the operations involved

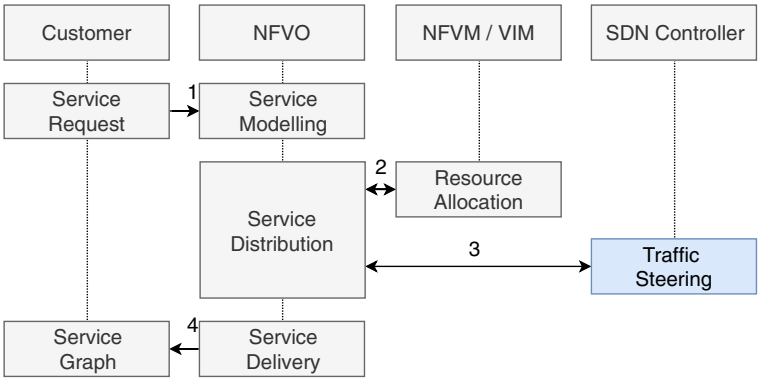


Fig. 2. SFC workflow. Adapted from [19].

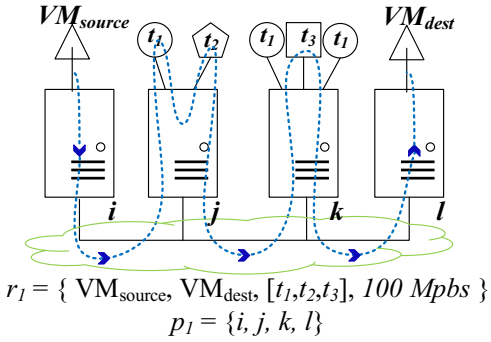


Fig. 3. VNF-FGE example.

in directing the traffic to reach the intermediate SFs of a specific chain [21].

2.2. VNF-FG embedding and deployment gaps

The embedding of SFC requests in the network infrastructure (as just described in previous section) is known in the literature as the VNF-FG embedding (VNF-FGE) problem, which is $\mathcal{NP}$-hard [22,23]. Such problem consists in allocating SF instances in physical hosts (placement decision) and defining paths (SFC decision) for a set of service requests in order to find an optimal solution according to a specific objective function, e.g., minimize the number of SF instances [24], and minimize OPEX [25].

An example of the VNF-FGE problem is shown in Fig. 3, for a topology composed by four interconnected server nodes: *i, j, k*, and *l*. The figure abstracts the interconnection topology between servers (green cloud). Service request r_1 defines a 100 Mbps traffic originated in virtual node VM_{source} (hosted at server *i*) and addressed to virtual node VM_{dest} (hosted at server *l*). This traffic must pass through SF instances of types t_1, t_2, and t_3, respectively, before reaching the final destination. In this example, instances of t_1 and t_2 (both allocated in server *j*), and an instance of t_3 (allocated in server *k*) will serve this request. Thus, in this example, the NFVO chooses path $p_1 = \langle i, j, k, l \rangle$, considering the servers hosting the SF instances.

The NFVO runs a VNF-FGE solving algorithm that makes embedding decisions, according to the defined optimization objectives [22]. One of the orchestrator responsibilities is to perform traffic engineering, which selects the paths (or routes) for each SFC segment for efficient resource usage. Fig. 4 illustrates a typical traffic engineering workflow, consisting of a control loop with the following steps [10]: (i) monitoring and evaluation of metrics of interest; (ii) computation of an optimal resource usage solution; (iii) deployment on the network infrastructure.

Several works have investigated optimization models and heuristic algorithms to solve the VNF-FGE problem using simulations [8,22], and considering different optimization objectives, e.g., minimize OPEX [25], the number of SF instances [24], or deployment costs. In [24,26], the authors formalized the placement and chaining problem and proposed an Integer Linear Programming (ILP) model and an heuristic to solve it. Other works developed models and heuristics to tackle the scalability of this problem for larger scenarios [23,27,28]. Another relevant approach is the study of the VNF-FGE problem in multi-domain scenarios [29,30]. Moreover, DC traffic presents significant workload variation in short timescales due to the start and end of user requests in a shared infrastructure [10]. To adapt to such dynamic workloads, some works also developed solutions to enable the reallocation of previously determined placement and chaining decisions according to NFV demands [31–33].

These optimization solutions require deployment options that allow the selection of any available resource and enable the exploitation of the rich path redundancy in the underlying DC network. For instance, traffic engineering may need to select a set of paths and load-balance between them, choosing non-shortest paths to avoid congestion or faulty paths, or quickly change paths to adapt to highly variable demands [4,10]. Also, to ensure that the actual resource usage is compatible with the resource allocation solutions, traffic engineering must be able to specify each forwarding element in the path for each SFC segment. In summary, traffic steering mechanisms in the underlay should meet the following requirements.

- **Programmable**: Flow control should be decoupled from hardware and managed by software using application programming interfaces (APIs).
- **Expressive**: It should be possible to select any path between two endpoints, and to specify all the forwarding elements in this path.
- **Scalable**: It should support the encoding of a diverse set of paths between each endpoints, while minimizing control plane overhead.
- **Agile**: It should allow quick changes to the paths, considering the convergence time to apply these changes in all affected nodes.

While many optimization solutions have been proposed in the literature, much less attention has been dedicated to the development of the underlay traffic steering mechanisms that enable the deployment of such resource allocation solutions in the network infrastructures of DCs (highlighted in red in Fig. 4).

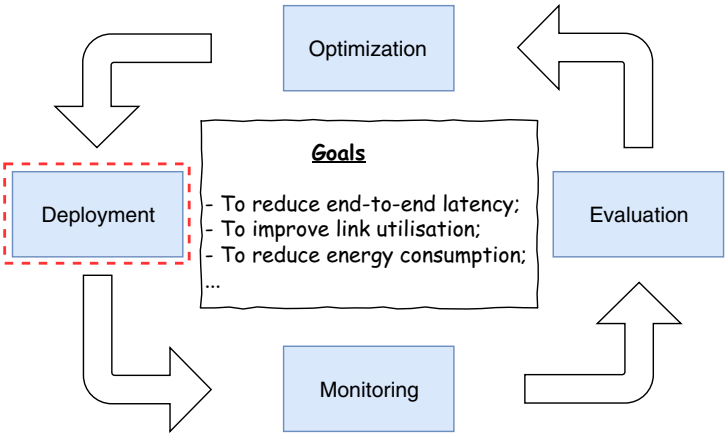


Fig. 4. Traffic engineering workflow (adapted from Tso et al. [10]).

According to [8], the optimization studies for SFC lack practical value and need modifications to be suitable for the SFC architecture, while the problem of deployment of dynamic function chains is an open challenge that needs to be addressed. In [21], the authors present a comprehensive survey of traffic steering techniques for SFC and conclude that current solutions are not efficient enough to be deployed in real-life networks, mainly due to scalability and flexibility limitations. The next section details the problems of current solutions for SFC deployment.

2.3. Problems of current traffic steering solutions

As discussed in the previous section, current traffic steering mechanisms do not offer the levels of programmability, expressiveness, scalability, and agility that are necessary to deploy current state-of-the-art resource allocation solutions in the network underlay. Existing solutions limitations can condensed in four main problems.

The first problem is to consider the network underlay that interconnects the servers in Fig. 3 as a mere way to provide connectivity to the service overlay layer. Proposed solutions normally do not offer adequate mechanisms for traffic engineering to define paths between SFC segments, and sometimes leave the responsibility of selecting paths to routing methods that are separated from the SFC decisions. For instance, in the network service headers (NSH) protocol [5], the decisions for routing (in the underlay) and SFC (in the overlay) are completely decoupled and executed by different mechanisms.

The second problem happens when underlay routing mechanisms have limited scalability and require the insertion of additional constraints in the resource allocation models. For example, traditional flow-based methods for SFC install a large number of flow rules in the switches to steer traffic and the number of states is limited by the size of forwarding tables [4]. Therefore, path options are commonly restricted to a single shortest path or a small set of shortest paths between each pair of segment endpoints. This occurs in the Segment Routing protocol [34], which can enable SFC over a MPLS infrastructure, but restricts the maximum number of nodes that can be specified in a path depending on the MPLS equipment (about 3 or 5 [35]) and delegates the routing between these nodes to the underlying network mechanisms. In this way, traffic engineering algorithms have to take into account specific constraints of path encoding as objective functions [36], which may lead to inefficient traffic distribution and network congestion [35].

The third problem is the difficulty to provide agile path selection in response to dynamic traffic requests. Most traffic steering solutions, such as traditional flow-based methods for SFC and the NSH protocol, rely on rigid table-based routing mechanisms in the underlay. Thus, a path migration may involve changing table entries in all the nodes of that path, which may lead to control plane overhead and long convergence time to configure changes.

The fourth problem is that traffic steering mechanisms normally only support network-centric topologies. Thus, they miss an opportunity to exploit SFC in server-centric or hybrid topologies, where servers are directly interconnected and perform both forwarding and processing tasks [2].

To tackle these problems, this work aims to address the following question:

- How to design a **programmable, expressive, scalable, and agile traffic steering solution** that enables dynamic and efficient orchestration of the underlay of DCNs?

3. Application of RNS-based SSR to traffic steering

In this work, we advocate that an appropriate solution for the traffic steering problem is to use SSR with RNS-based forwarding in the core nodes, and SDN for classification in edge nodes. This section explains the concepts of SSR and RNS, and compares traditional table-based traffic steering with our approach.

3.1. Source routing

Routing methods define the path a packet takes from source to destination, and have great impact on the performance of traffic steering mechanisms. Routing can be classified according to different criteria [37]:

- **Routing decisions**: (i) **source routing (SR)** – the routing path is defined at the source and the path computation only needs to be done once for each packet; and (ii) **per-hop routing** – at each hop enroute to the destination, the packet goes through a routing computation to define the next hop.
- **Routing computations**: (i) **algorithmic routing** – based on current node and destination information, a fixed logic is used to find the output port; and (ii) **table-based routing** – a lookup table returns the appropriate output port based on some field of the packet header, such as the source or the destination.

OpenFlow networks traditionally adopt table-based and per-hop routing, since an SDN Controller distributes the forwarding states in flow tables of the network switches. These tables contain entries that match flows to forwarding actions, and new flows trigger new

state distribution to all devices along the path [7]. The distribution and management of forwarding states in the network bring some limitations, because the control plane signaling generates overhead traffic, and the switches have to wait for a response from the Controller when a new flow arrives [7,15].

SR eliminates tables and complex routing responsibilities from intermediate nodes, placing the responsibility for route selection at the ingress nodes [11]. Also, SR methods can be classified as [9]: (i) **strict SR (SSR)**, if they specify the entire routing path, as in SecondNet [14]; or (ii) **loose SR**, if they determine only some hops that the packet must go through, as in Segment Routing [12]. In this work, we reference as algorithmic SSR the methods that use SSR with algorithmic routing computation.

Port Switching is a traditional method for executing algorithmic SSR that represents the route as a stack of ports or addresses and the forwarding operation as a stack pop. SecondNet [14], Segment Routing [12], and Sourcey [9] are examples of works that explore Port Switching SSR. This work adopts RNS-based SSR, an alternative way of performing algorithmic SSR, as detailed in the next section.

3.2. RNS-based SSR

The residue number system (RNS) has been known as an alternative number system, based on the Chinese remainder theorem (CRT) [6]. The properties of RNS can be explored to provide some special SSR features not covered by Port Switching methods, such as packet forwarding without header modification and fast failure reaction [38,39].

A RNS-based SSR mechanism represents the route as a number according to the RNS (*routeID*) and the node as pairwise co-prime numbers (*nodeID*). Then, the output port in each node is given by the *modulo* (remainder of division) of the *routeID* of the packet by its *nodeID*.

The logic for computing *routeID*s exploits mathematical properties from RNS [6]. Let $S = \{s_1, s_2, \ldots, s_N\}$ be a multiset of the N *nodeID*s of the core nodes on the desired path, in which all elements are pairwise co-prime numbers. In addition, let $P = \{p_1, p_2, \ldots, p_N\}$ be a multiset of N outgoing ports, where p_i is the output port of the packet at the core node s_i. Also, let M be an upper bound value, defined as $M = \prod_{s_i \in S} s_i$.

Then, a natural number $0 \leq R < M$ can be represented by a *residue set* given a basis modulo set S:

$$R \xrightarrow{\text{RNS}} \{p_1, p_2, \ldots, p_N\}_S, \quad \text{where} \quad p_i = R \ \ modulo \ \ s_i \tag{1}$$

To define a route, it is necessary to find out the value of R (the explicit *routeID*), given a modulo set S (the *nodeID*s), and its RNS representation P (the core node output ports). The Chinese Remainder Theorem states that it is possible to reconstruct R through its residues in a RNS as follows, where $< a >_b \equiv \ a \ modulo \ b$ [40]:

$$R = < \sum_{s_i \in S} p_i \cdot M_i \cdot L_i >_M \tag{2}$$

$$M_i = M / s_i \tag{3}$$

$$L_i = < M_i^{-1} >_{s_i} \tag{4}$$

Eq. (4) means that L_i is the modular multiplicative inverse of M_i. In other words, L_i is an integer number such that:

$$< L_i \cdot M_i >_{s_i} = 1 \tag{5}$$

The modular multiplicative inverses are calculated using the Extended Euclidean algorithm [41].

Fig. 5 shows an example of a *routeID* computation based on RNS for a fabric network with 4 core nodes. For the path that binds the output ports $P = \{1, 1, 0\}$ to the nodes $S = \{4, 3, 5\}$, the *routeID* is 25. In fact, $< 25 >_4 = 1$, $< 25 >_3 = 1$, and $< 25 >_5 = 0$. The *routeID* (R) calculation consists of three steps:

1. The parameters M_i are obtained by dividing M by the respective *nodeID* (s_i), where $M = 60$ is the product of all *nodeID*s.
2. The parameter L_i is calculated as the modular multiplicative inverse of M_i ($< L_i \cdot M_i >_{s_i} = 1$). For our example, $< 3 \cdot 15 >_4 = 1$, $< 2 \cdot 20 >_3 = 1$, and $< 3 \cdot 12 >_5 = 1$.
3. The *routeID* is computed by $R = < \sum_{s_i \in S} p_i \cdot M_i \cdot L_i >_M = 25$.

The algorithm complexity for *routeID* computation is $\mathcal{O}\left(len(M)^2\right)$ [41], where $len(M)$ is the number of bits of the binary representation of M. It is important to note that the *routeID* is computed by the SDN Controller only when chains are created or updated, while core nodes only execute one *modulo* operation per packet. Also, in our scheme the *routeID* represents the path between two servers and does not grow with respect to the number of VMs and SFs, which is much larger and dynamic.

Besides, the SDN Controller may proactively compute the *routeID* for the paths between a pair of servers that the NFV Orchestrator has decided to consider, and use this information when necessary to apply orchestration decisions.

As explained in [6], the bit length of the *routeID* depends on: (i) the switch with the largest number of ports; (ii) the size of the core network; and (iii) the number of hops for the selected path. The works in [13,40,42,43] have already performed extensive scalability analysis of the RNS scheme and showed that is possible to use legacy small headers, such as MAC addresses, to encode the *routeID* for DCN topologies of reasonable size. In addition, Ren et al. [13,43] proposed methods to reduce the length of the *routeID*.

3.3. Comparison with traditional traffic steering solutions

One of the most important standardization effort for SFC is RFC 7665 [1], which proposes an SFC reference architecture. An

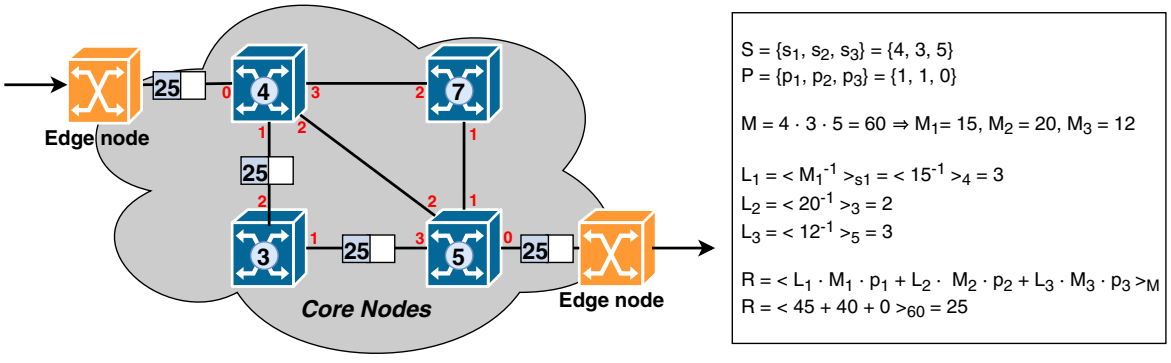
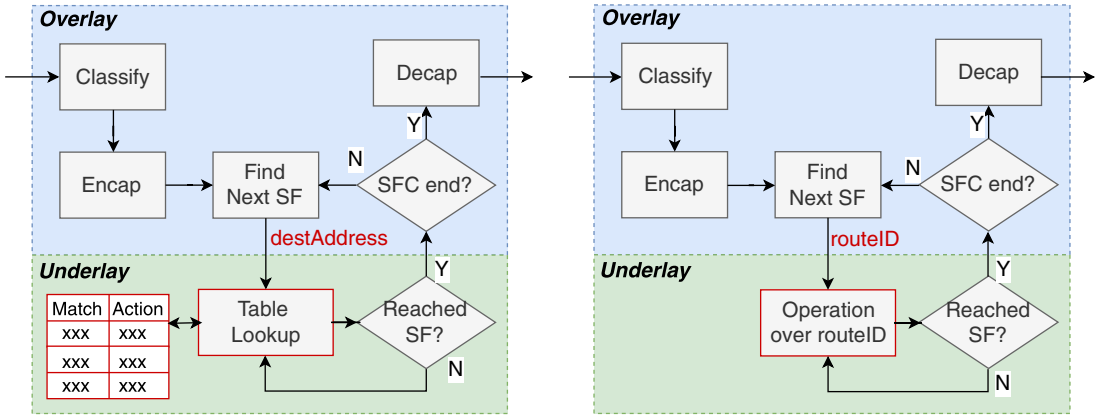


Fig. 5. Example of *routeID* computation using RNS. Adapted from [6].



(a) Workflow on current SFC approaches.

(b) Workflow using strict source routing.

Fig. 6. Traffic steering problem. Adapted from [44].

essential concept in this architecture is the Service Function Path (SFP), a configuration of where packets assigned to a specific SFC must go. In this work, we use the term SFP to represent all network nodes and SFs in the forwarding path.

In this architecture, when packets enter the network, they reach a Classifier. In this step, a packet is matched against a set of preconfigured rules and a SFP is chosen. An SFC encapsulation containing the corresponding path information is added to the packet and it is sent to a service function forwarder (SFF). SFFs use SFC encapsulation to forward packets to the next SF in the chain, and also handle chain termination. A SF may be SFC-encapsulation aware, which receives data with SFC encapsulation, or unaware, which receives data without SFC encapsulation. To support SFC-unaware SFs, it is necessary to include a Proxy between the SF and the SFF.

Fig. 6(a) shows how most of the current SFC solutions perform traffic steering based on RFC 7665. After classification, the SFC encapsulation carries information used to determine the network address of the SF to be executed in each SFC segment. The SFC encapsulation may create a new header, as in NSH [5] and Segment Routing proposals [34], or overload an existing header. This SF address is often encapsulated in an outermost header that is used for routing to the next SF. After determining the address of the current SF, the packet is delivered to the underlying routing mechanism (e.g., MPLS, SDN, or IP), where the forwarding is executed using per-hop table lookups based on the destination address until it reaches the SF. At this moment the encapsulation information is checked again to find the address of the next SF and this loop proceeds until the chain terminates, when the packet is decapsulated and delivered to destination.

The problems with this traditional SFC approach are threefold: (i) the commonly adopted underlying routing mechanisms cannot represent all the possible paths between two endpoints due to limited capacity of switch forwarding tables; (ii) as they are based on per-hop table lookup, the overhead to dynamically change a path is high, because it may involve modifications in all the nodes in the path; and (iii) traffic engineering decisions are made in a decoupled way considering placement, chaining, and routing.

Our traffic steering solution uses SSR in the core nodes with algorithmic routing computation, and SDN-enabled edge nodes in the servers for SFC classification and encapsulation. Fig. 6(b) illustrates how our proposal uses SSR for determining the specific path of each SFC segment. The path information is inserted in the encapsulation stage in the form of a *routeID*. This identifier is passed to the underlying routing layer, where each hop can take forwarding decisions based on a simple operation over the *routeID*.

Our solution presents several benefits over traditional approaches. Firstly, as already proved by other works [4], SSR allows traffic engineering to exploit all existing paths, and, consequently, achieve maximum throughput. Secondly, it enables traffic engineering to specify the entire path for each SFC segment. Besides, we eliminate tables in core nodes in order to reduce latency, jitter, and control plane signaling [6]. Finally, if the path needs to change to fit any dynamic demand, the source only needs to encapsulate a new *routeID*, and intermediary nodes will continue to operate over the received identifier without any modification.

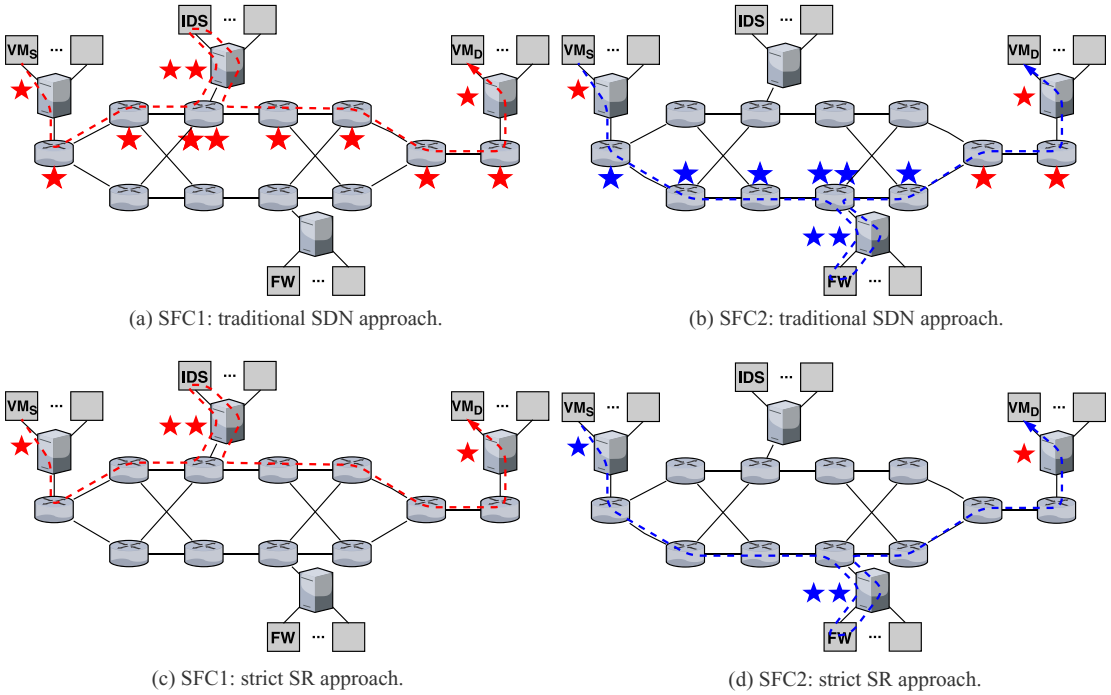
These benefits are particularly important when we consider the dynamic characteristics of the NFV traffic: SFC requests constantly start and terminate; the sequence of SFs in a chain may change during its life cycle; or traffic engineering may need to migrate paths and SFs to guarantee optimal resource usage. Fig. 7 illustrates an example scenario of dynamic chain migration due to security demands. Initially, the NFVO provisions SFC1 ($VM_S \rightarrow IDS \rightarrow VM_D$) for a specific flow, considering overall resource utilization and the need to steer the traffic through an IDS (intrusion detection system). Then, when an attack is detected, the flow is directed to a firewall (FW) and migrated to SFC2 ($VM_S \rightarrow FW \rightarrow VM_D$).

Fig. 7(a) shows how a traditional SDN-based approach installs flow entries (represented as red stars) in all the elements in the forwarding path to guarantee the full specification of an SFP for SFC1. When the SFC is migrated, Fig. 7(b) shows that some of these entries need to be removed and new ones (represented as blue stars) need to be added to steer the traffic through SFC2. On the other hand, Fig. 7(c) shows that our strict SR approach only installs flow entries (represented as red stars) in the SFC segments' endpoints, in order to classify the flow and include the *routeID*s for SFC1. When the SFC is migrated, Fig. 7(d) shows that a small number of flow entries have to be modified (represented as blue stars) to include the *routeID*s for SFC2. Thus, our approach requires the installation of fewer flow entries and, consequently, can react faster to path changes.

Other traditional SFC approaches, like NSH and Segment Routing, usually do not specify all elements of the SFP. Instead, they encapsulate the address of the next SF and rely on underlying routing methods to deliver packets between SFC endpoints. More details about related works are given in Section 6.2.

4. KeySFC proposal

This section provides a complete description of KeySFC, specifying its architecture and presenting a complete example of a service provisioning in this novel proposal.



(a) SFC1: traditional SDN approach.

(b) SFC2: traditional SDN approach.

(c) SFC1: strict SR approach.

(d) SFC2: strict SR approach.

Fig. 7. Comparison between flow entries (represented by stars) in traditional SDN and strict SR approaches for dynamic migration from SFC1 ($VM_S \rightarrow IDS \rightarrow VM_D$) to SFC2 ($VM_S \rightarrow FW \rightarrow VM_D$).

4.1. Architecture

According to ETSI and other references from literature [45–47] a typical NFV architecture is composed by at least three layers: management, control and data planes. Fig. 8 depicts these planes for KeySFC.

In the management plane, applications such as network optimization, traffic monitoring, prediction, and policy definition are responsible to generate management decisions to the control plane about SFC configuration. The SFC problem can be divided in three main phases: service modeling, resource allocation, and traffic steering [19] (as presented in Fig. 2). When service requests arrive, the service modelling and resource allocation tasks are performed by the management plane. As a result, it generates the respective VNF-FG and SFPs for the traffic steering mechanisms in the control plane, which, in turn, configures the data plane.

The scope of our proposal lies on these two lower planes (control and data planes). More specifically, KeySFC replaces tables on the routing stage with RNS-based SSR, using flow entries only for SFC classification. Thus, it is possible to position KeySFC as a traffic steering scheme that enables the control and data planes to support a RNS-based SSR architecture for SFC.

The control plane is responsible for topology discovery, SFC management, and data plane configuration (more details in Section 4.3). Control plane actions are supported by two databases: forwarding information and SFC configuration. According to the SFC decisions received from the management plane (VNF-FG and SFPs), the KeySFC control plane calculates and stores three identifiers in the SFC configuration database: SFC segment identifiers (*segID*), route identifiers (*routeID*), and node identifiers (*nodeID*).

The logic for computing *routeID*s and *nodeID*s exploits mathematical properties from RNS [6], as explained Section 3.2. The set of *nodeID*s are assigned by the control plane to each core switch of the topology in a network configuration phase. The *routeID* and the *segID* of each SFC segment are combined in a identifier called VMAC (more details in Section 4.2). For each segment in the SFC configuration database, the KeySFC control plane application computes table entries that will be installed in forwarding nodes for classifying SFC flows and embedding the VMACs in the packet header. These flow entries are stored in the forwarding information database.

In the data plane, the forwarding in each node is defined by the rest of the division of the *routeID* of the packet by the *nodeID* of the core switch, which gives the output port. The adopted scheme can be applied in any topology (network-centric, server-centric, or hybrid).

Fig. 9 shows how the KeySFC architecture complies with the ETSI NFV standard [18]. Hardware resources are managed by a MANO block that is part of the management plane, formed by a VIM, VNFM, and a NFVO. The NFV Infrastructure block is composed by COTS servers and switches, both programmable by an SDN Controller. A KeySFC-enabled server is composed by four elements (Fig. 9):

1. VMs that act as sources and destinations of packets.
2. VMs that host SFs.
3. An edge switch (in yellow), called *Eswitch*: a software switch that serves as both ingress and egress element for traffic entering and leaving VMs. It supports the OpenFlow protocol, and is managed by the SDN Controller, which installs forwarding rules to tag and steer SFC traffic.
4. A core switch (in green), named *Cswitch*, which connects to physical network interfaces, and handles communication between VMs that reside in different servers by using a RNS-based SSR mechanism. It can be implemented as a software or a hardware switch.

Fig. 9 also shows that KeySFC fits various DCN architectures designs. For architectures based on hardware switches (i.e., network-centric, and hybrid DCNs), these switches are implemented as core elements (e.g. Switches A and B). In server-centric architectures, servers are directly interconnected, and the server itself contains a *Cswitch* element for routing.

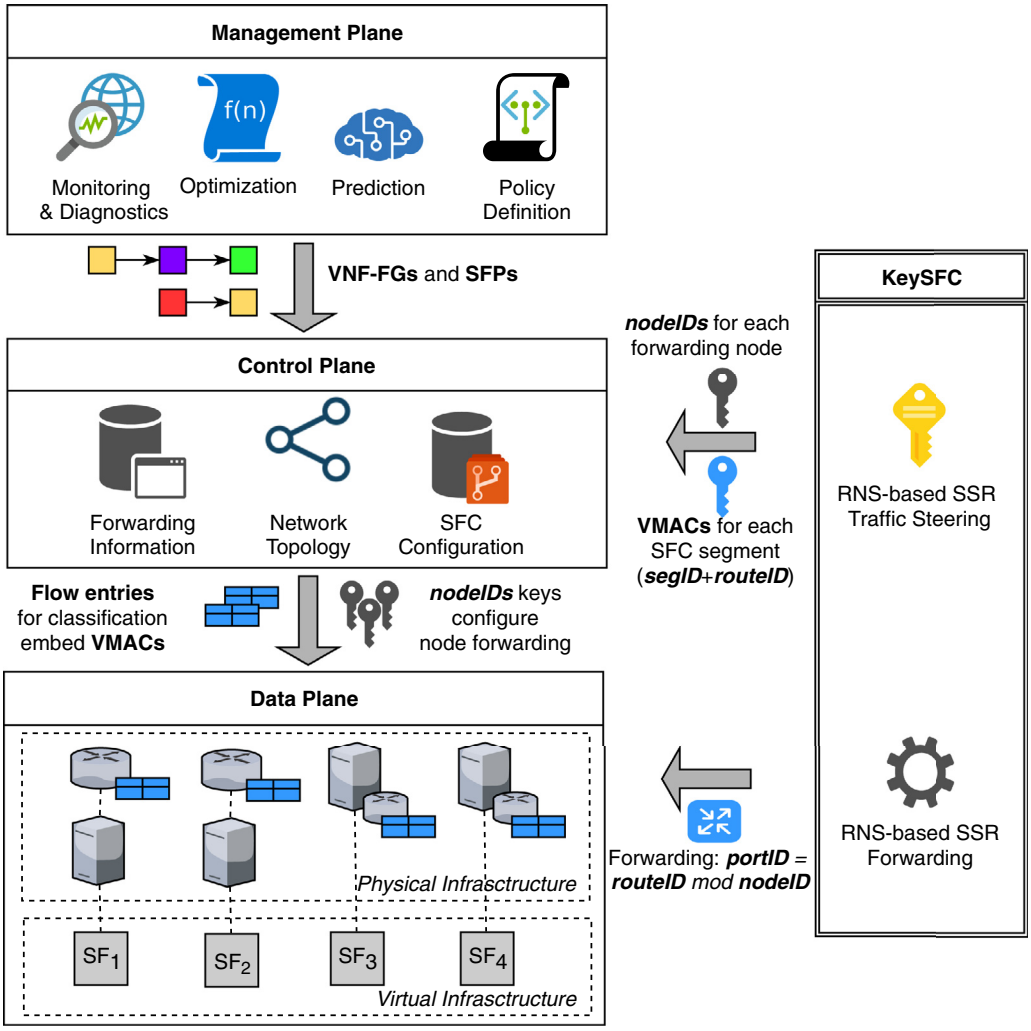


Fig. 8. KeySFC architecture: management, control, and data planes.

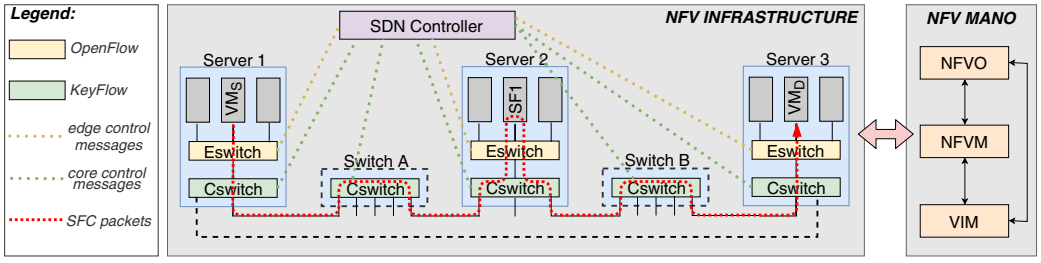


Fig. 9. KeySFC and ETSI NFV standard: example of SFC $VM_S \rightarrow SF1 \rightarrow VM_D$.

4.2. Data plane

The main idea behind our proposal is that any traffic flow that traverses edge switches, and is part of a chain, will be classified by matching pre-installed flow rules. Such rules are installed by the SDN Controller and rewrite Ethernet MAC addresses in order to give a new meaning for that set of bits, called virtual MAC (VMAC) in our approach.

The VMAC can use the 48 bits of the Destination MAC (DstMAC) address in the Ethernet frame, or combine the 96 bits of DstMAC and Source MAC (SrcMAC) addresses. The choice of whether to use one or two MAC addresses is coupled with the maximum bit length of the *routeID* (see Section 3.2), and depends on design choices when deploying KeySFC, such as number of nodes, number of chains, and number of segments per chain.

Fig. 10 shows the format of a VMAC address. It is divided in two parts: the most significant 16 bits represent the *segID*, and the remaining bits (32, if using only DstMAC, or 80, if using DstMAC and SrcMAC) represent the *routeID*. The *segID* uniquely identifies the current chain segment, and, when the segment ends, it is used to match a flow entry that sets the next segment. The *routeID* is used by each forwarding node to define the next hop.

It is important to highlight that the use of the MAC address field to encapsulate the SFC information was a *design choice*, and the same approach could be adapted to use other kinds of encapsulation, such as NSH or MPLS headers, or even a new header. The main advantages of using existing headers are the reduction

Fig. 10. VMAC address format.

of header overhead and the compatibility with legacy forwarders and SFs. On the other hand, some SFs may require to read or modify the MAC addresses. This can be solved with the addition of a Proxy between the SFFs and SFs, as defined in RFC 7665 [1].

When receiving SFC decisions from the NFVO, the SDN Controller splits each chain in segments, and calculates the VMAC for each segment. Then, it installs flow entries in *Eswitches* to tag packets with the corresponding VMACs. Initially, the packet leaves the source VM, and reaches the *Eswitch* of the host server. There, it matches a flow entry that performs classification based on 5-tuple, and tags the first VMAC into the packet. If the next SF in the chain is located in the same server, the packet is directly sent to that SF. Otherwise, the packet is sent to the *Cswitch*. At each hop, the forwarding mechanism in the *Cswitches* performs a *modulo* operation over the packet's *routeID* to find the next hop. Thus, forwarding nodes perform a simple operation over L2 headers to find the output port.

These stateless operations take place along the path until the packet reaches the server hosting the endpoint SF of that SFC segment, and is forwarded to the *Eswitch* of that server. There, the packet's VMAC matches a flow entry that redirects the packet to the SF. After processing, the SF sends the packet back to the *Eswitch*, where the current VMAC matches another flow entry that tags the VMAC of the next segment into the packet.

These steps are repeated for all segments, until the packet arrives at the host of the destination VM. Finally, a flow entry at the *Eswitch* changes the VMAC field to the original MAC addresses, and the packet is forwarded to the destination VM. It is important to note that the packet that reaches destination is exactly equal to the packet that left source. Section 4.4 exemplifies this approach with a complete SFC scenario.

The implementation of *Cswitches* can be done by modifying the datapath of devices based on Open vSwitch (OvS), by using NetFPGAs [40], or by exploring devices that support the P4 language. Besides, KeySFC allows performance optimization in the *Cswitches* of servers. Although the basic approach uses software switches (as implemented in the prototype of Section 5), it can be extended by offloading forwarding tasks to specialized devices (e.g., a networking co-processor SmartNIC).

4.3. Control plane

The control plane has five fundamental roles in KeySFC: (i) topology discovery and monitoring; (ii) configuration of *nodeIDs* for *Cswitches*; (iii) management of chain segments; (iv) installation of flow rules in *Eswitches* according to SFC decisions; and (v) ARP proxy.

To perform these tasks, the SDN Controller has to have full knowledge of the network topology, the SFC requests, the SFs that serve these requests, and the virtual ports where each SF is attached in each physical server. The information about the SFCs and the SFs is obtained via communication with the NFVO. The discovery of the topology uses the LLDP protocol.

Based on its centralized view of the infrastructure, the SDN Controller is responsible for a network configuration phase, where it assigns a set of co-prime numbers to the *nodeIDs* of servers and hardware switches. This is accomplished by communicating with *Cswitches* via OpenFlow protocol.

Afterwards, the Controller calculates two identifiers for all segments of each chain: the *routeID* and the *segID*. The correlation between each chain and its respective segment identifiers is persisted in the SFC configuration database. Then, it installs forwarding rules in the *Eswitches* using the OpenFlow protocol. These rules classify flows and guarantee that, when entering in each chain segment, the flow is tagged with its respective identifiers.

Finally, the SDN Controller also acts as an ARP Proxy in order to reduce the impact of broadcast. When a VM_S at Server 1 wants to send packets to a VM_D hosted in other server, VM_S sends an ARP request broadcast message with the IP of VM_D. This message is intercepted by the *Eswitch* of Server 1 and forwarded to the SDN controller that resolves the address and sends an ARP reply message to the *Eswitch*, which, in turn, delivers the message to VM_S. Then, VM_S can send data to VM_D using the discovered MAC.

One advantage of KeySFC is the significant reduction of control plane signaling. Considering routing, the SDN Controller only answers ARP queries, since *Cswitches* execute forwarding in a decentralized way without control plane communication. Other steps, such as topology management and configuration of *nodeIDs* can happen in a proactive manner. So, considering SFC, only the update or creation of chains generate control plane communication.

4.4. Example: A day in the life of an SFC request

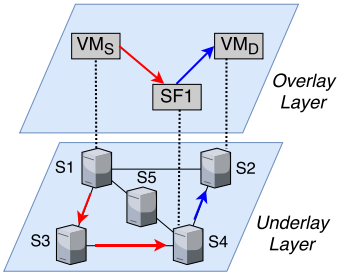
This section provides a step-by-step example of how KeySFC executes an SFC request. Consider the illustrative scenario shown in Fig. 11: a 5-node server-centric topology that provisions the SFC $VM_S \rightarrow SF1 \rightarrow VM_D$. We have two segments: $VM_S \rightarrow SF1$, and $SF1 \rightarrow VM_D$. The first segment is mapped to two hops in the underlay ($S1 \rightarrow S3$, and $S3 \rightarrow S4$), and the second segment is mapped to a single hop ($S4 \rightarrow S2$).

Initially, the controller assigns the *nodeIDs* to servers (9, 11, 13, 17, 19), splits the chain in two segments, and calculates the identifiers: VMAC1 (*routeID1* = 4051, *segID1* = 10) for segment $VM_S \rightarrow SF1$, and VMAC2 (*routeID2* = 30, *segID2* = 100) for segment $SF1 \rightarrow VM_D$. Then, it installs flow entries at *Eswitches*, as shown in Table 1. When VM_S sends traffic to VM_D the following events happen:

1. **At S1**: A packet leaves VM_S and reaches the *Eswitch*, where the flow matches **rule 1**, causing the DstMAC to be changed to VMAC1 and the packet to be forwarded to the *Cswitch*. In the *Cswitch*, the *modulo* operation of *routeID1* by the *nodeID* of S1 gives $< 4051 >_{19} = 4$. Thus, the output port is 4, and the packet is forwarded to host S3.
2. **At S3**: When the packet reaches the *Cswitch*, the *modulo* operation of *routeID1* by the *nodeID* of S3 gives $< 4051 >_{17} = 5$. Therefore, the packet is forwarded to port 5, which is connected to S4. The packet does not go up to the *Eswitch*, because S3 only forwards traffic.
3. **At S4**: When the packet reaches the *Cswitch*, the *modulo* operation of *routeID1* by the *nodeID* of S4 gives $< 4051 >_{13} = 8$. Thus, it forwards the packet to the *Eswitch*, where it matches **rule 2**, causing the packet to be sent to SF1. SF1 processes the packet and forwards it back to the same interface. The returning packet reaches the *Eswitch* and matches **rule 3**, causing the DstMAC to be changed to VMAC2, with the packet being sent to the *Cswitch*. There, computing the *modulo* of *routeID2* by the *nodeID* of S4 gives $< 30 >_{13} = 4$. Therefore, the packet is sent to S2 via port 4.

Table 1
Simplified flow tables at *Eswitches*.

#	Server	Match	Action
1	S1	SrcIP = IP VM_S; L4DstPort = P1	DstMAC = VMAC1; Out to *Cswitch*
2	S4	DstMAC = VMAC1	Out to SF1
3	S4	DstMAC = VMAC1; InPort = SF1 port	DstMAC = VMAC2; Out to *Cswitch*
4	S2	DstMAC = VMAC2	DstMAC = MAC VM_D; Out to VM_D

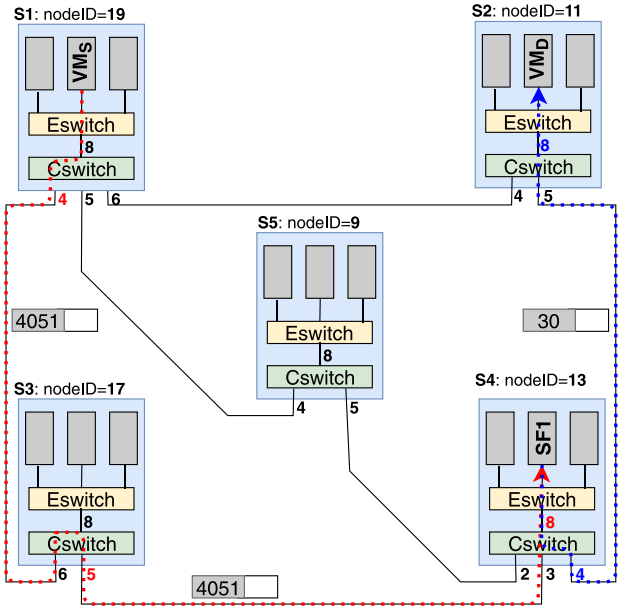


(a) Overlay and underlay.

Segment1 (VM_S-SF1): VMAC1
segID1= 10 | routeID1= 4051
S1: $<4051>_{19} = 4$
S3: $<4051>_{17} = 5$
S4: $<4051>_{13} = 8$

Segment2 (VNF-VM_D): VMAC2
segID2= 100 | routeID2= 30
S4: $<30>_{13} = 4$
S2: $<30>_{11} = 8$

(b) Forwarding operations.



(c) Detailed scenario.

Fig. 11. KeySFC example ($VM_S \rightarrow SF1 \rightarrow VM_D$).

4. **At S2**: When the packet reaches the *Cswitch*, the *modulo* operation of the *routeID2* by the *nodeID* of S2 gives $< 30 >_{11} = 8$. Thus, the packet is sent to the *Eswitch*, where the flow matches **rule 4**, causing the DstMAC to be changed to the MAC VM_D, and the packet to be delivered to VM_D.

5. Implementation and evaluation

This section evaluates a proof-of-concept implementation of the KeySFC scheme. Firstly, we present reference SFC scenarios that can capture some trade-offs for SF placement and chaining in Section 5.2. Then, in Section 5.3, we execute functional tests to check the correct behavior of our traffic steering scheme. Afterwards, Section 5.4 evaluates the performance of KeySFC by measuring latency and jitter on an end-to-end service. Finally, in Section 5.5, we test some traffic engineering scenarios that explore steering functionalities offered by KeySFC.

5.1. Proof-of-concept prototype

We implemented a proof-of-concept prototype of the KeySFC scheme for the 5-node server-centric topology of Fig. 12. It implements all the KeySFC architectural components, as presented in Fig. 9.

The *OpenStack* platform (release *Icehouse*) was selected as our VIM and VNFM because its server networking architecture is organized in two layers, implemented as OvS bridges, that could be adapted to our scheme. To this end, the KeySFC RNS forwarding algorithm was implemented in the OvS datapath module of *Cswitches*. The OpenStack deployment includes a Cloud and a Network Controller [48]. The SDN Controller was implemented using the *Ryu* framework (version *2.13*). The NFVO is a Python application that has a CLI interface to receive SFC requests, and integrates with the OpenStack Controllers and the SDN Controller.

The testbed is composed by 9 physical servers running Linux Ubuntu (12.04.5 LTS, kernel 3.5.0), as shown in Fig. 12: 5 server nodes (S1-S5), one NFVO, one SDN Controller, one OpenStack Cloud Controller, and one OpenStack Network Controller. Each server node has one Intel Xeon E5-2620 2.4 GHz processor, 16GB of memory, and four or five 1Gbps Ethernet NICs: one is connected to the OpenStack data network, one is connected to the OpenStack management network, and the remaining are connected to other servers to build the server-centric topology of Fig. 12.

The basic behavior of a SF is to receive packets, to perform some processing that may or not modify packets, and to send packets to the next hop. The time a SF spends processing packets and the nature of the modification it performs in the packets is deeply related to the specific application. In our proof-of-concept, to isolate the effects of the SFC mechanism itself from the performance limitations of specific SFs, all the SFs in our test scenarios implement a *forwarding function*. In this way, we can state that our performance evaluation is influenced only by the KeySFC scheme and not caused by the behavior of one or more specific SF. To implement this forward function, we create an OvS bridge inside the VM of the SF and install a single flow entry to redirect the traffic back to *Eswitch* when it reaches the SF.

5.2. Reference SFC scenarios

One SFC segment is the path between two VMs, which can be of three types: source, SF, or destination. When installing SFC rules, it is necessary to verify the following cases for each segment:

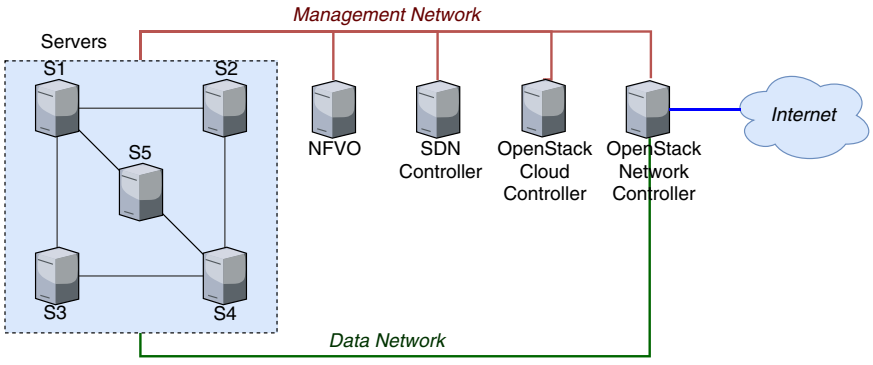
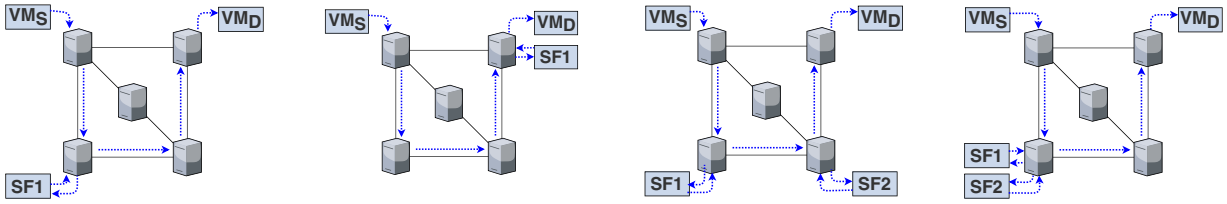


Fig. 12. KeySFC prototype testbed.



(a) SC1. (b) SC2. (c) SC3. (d) SC4.

Fig. 13. Reference scenarios exploring different placement strategies:(a) in **SC1**, 1 SF and all VMs in different servers; (b) in **SC2**, 1 SF allocated with VM_D; (c) in **SC3**, 2 SFs and all VMs in different servers; and (d) in **SC4**, 2 SFs allocated in the same server.

1. Source and destination endpoints of that segment are SFs;
2. Source endpoint of that segment is also the source of the chain; or
3. Destination endpoint of that segment is also destination of the chain.

For each case, it is necessary to verify the following subcases: (a) VMs are allocated in the same server, and (b) VMs are allocated in different servers.

When the two endpoints of a segment are in different servers, a *routeID* must be generated for routing in the physical network. Also, a flow entry in the *Eswitch* of the source endpoint should include this identifier in the packet header, and send it to the *Cswitch*. If the endpoints are in the same server, the flow remains in the *Eswitch*, but is redirected to the input port of the VM that is the destination of that segment. Moreover, for Case 3, where the VM is also the destination of the chain it is necessary to replace the VMAC by the real MAC of the destination to terminate the chain.

To cover different combinations between these cases and subcases, we designed four reference scenarios presented in Fig. 13. Scenario 1 (SC1) and Scenario 2 (SC2) have one single SF, but the later covers the case where the SF is placed in the same server of destination. Besides, SC1 involves segments with 1 and 2 hops in the physical network, while SC2 involves segments with 0 and 3 hops in the physical network. On the other hand, both Scenario 3 (SC3) and Scenario 4 (SC4) have two SFs, but, in SC4, both SFs are placed in the same server. In addition, SC3 only involves segments with 1 hop in the physical network, while SC4 involves segments with 0, 1, and 2 hops in the physical network, diversifying the routing combinations to test KeySFC. The case where the source VM is allocated in the same server as a SF was also tested, but results were omitted because they were similar to the case where the destination is allocated with a SF.

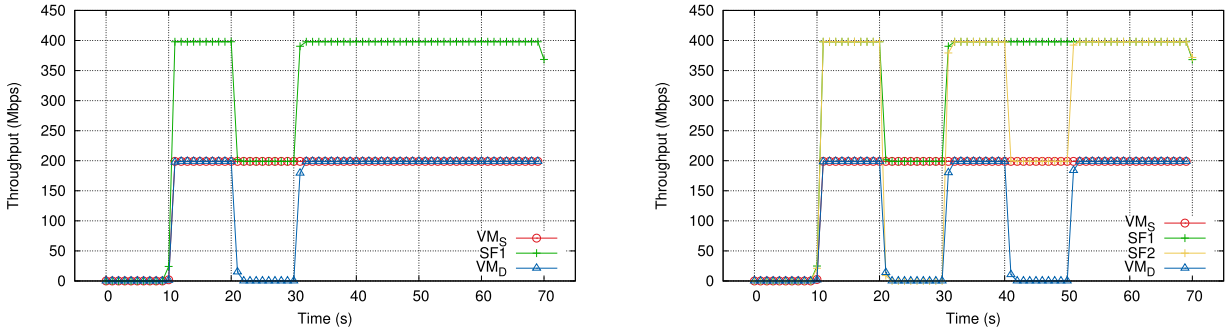
We compare the results of the reference scenarios with a baseline scenario (BL) presented in Fig. 15(a), when traffic goes from VM_S to VM_D without going through any SF. To keep consistency, the end-to-end path in the underlay is always the same for all scenarios.

5.3. Functional test

This first test is intended to verify the correct operation of KeySFC in our reference scenarios (Fig. 13). We want to show that all traffic that leaves the source is intercepted and steered through the specified SFs in the right order, and arrives at destination. To this end, we turn off one SF of the chain at a time, and check that nodes positioned after such SF stops to receive traffic while the SF is not forwarding.

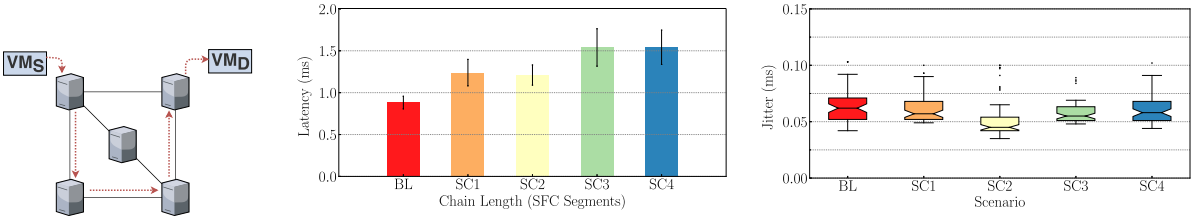
Fig. 14 shows the functional tests for scenarios SC1 and SC3, as specified in Fig. 13. Results for SC2 and SC4 were omitted, because they follow the same behavior as SC1 and SC3, respectively. In Fig. 14, at instant 10*s*, VM_S starts to send a 200 Mbps UDP traffic using the `iperf` tool to VM_D, and this traffic must be steered through one or two SFs, depending on the scenario. From 10 to 20 s, all the SFs are forwarding traffic. SFs perceive a total bandwidth that is twice the bandwidth sent by VM_S (200 Mbps), because they receive traffic and send it back to the same network interface. VM_D receives 200 Mbps of traffic during this interval. From 20 s to 30 s, the forwarding function at SF1 is turned off, so traffic drops from 400 Mbps to 200 Mbps at SF1, because traffic enters in network interface, reaches SF1, but does not return. Besides, traffic drops to zero in VM_D and SF2 (for scenarios with two SFs), because it was interrupted at SF1. At instant 30*s*, SF1 is turned on again, and traffic returns to initial conditions, proving that it was indeed passing through SF1.

In scenarios with two SFs, we repeat the same procedure for SF2, turning it off during a 10s interval, and we see that VM_D stops receiving traffic, but nodes in a previous position in the chain (i.e., VM_S and SF1) continue to receive traffic normally. These tests show that, for all scenarios, packets were indeed steered through each SF in the correct order, and that VM_D correctly receives traffic sent by VM_S after the SFC.



(a) SC1 ($VM_S \rightarrow SF1 \rightarrow VM_D$). (b) SC3 ($VM_S \rightarrow SF1 \rightarrow SF2 \rightarrow VM_D$).

Fig. 14. Throughput results for functional tests: (a) in **SC1**, SF1 is off from 20*s* to 30*s*; and (b) in **SC3**, SF1 is off from 20*s* to 30*s* and SF2 is off from 40*s* to 50*s*.



(a) Baseline (BL). (b) Latency over time for SFC scenarios. (c) Box plot of jitter for SFC scenarios.

Fig. 15. Jitter and latency results comparing reference scenarios with baseline (BL).

5.4. Performance tests

5.4.1. Latency and jitter on reference scenarios

In this first set of performance tests, we measure end-to-end latency and jitter in the reference scenarios (Fig. 13). Each test was run for 60 s and it was run 30 times, graphs present the mean of the values in the interval, and bars represent standard deviation. The confidence interval is 95%.

The first test (Fig. 15(b)) aims to understand SFC impact on latency, and uses `ping` to send an ICMP message from VM_S to VM_D. It needs SFC rules in both directions to guarantee that both ICMP request and reply messages pass through the same SFs, but in reverse order. The results show that latency increases from approximately 0.8 ms in the baseline scenario to an average of 1.2 ms in scenarios with one SF, and to an average of 1.5 ms in scenarios with two SFs. This impact in latency is expected, because SFC scenarios add extra hops when passing through the SFs.

The second test, shown in Fig. 15(c), compares jitter in the reference scenarios to the baseline. VM_S sends a 200 Mbps UDP traffic using `iperf` to VM_D, and SFs forward traffic without interruption. Our first observation is that there are small differences in the median: 0.05 ms for the baseline against approximately up to 0.06 ms in all SFC scenarios. Even when one or two hops are added in the chain, there is no considerable degradation in jitter, keeping it quite small. Also, the box plot shows that the distributions are well behaved, varying from 0.04 to 0.07 ms for the 75 percentis, with jitter reaching 0.1 ms only in the worst case of SC1. Thus, these results indicate that the KeySFC scheme has no strong impact on jitter, which remains very small independently of the number of SFs for the reference scenarios. Also, the distributions can be considered statistically equivalent for all scenarios, which can be later explored by jitter sensitive applications.

5.4.2. The impact of chain length in latency

This test investigates how the chain length impacts end-to-end latency in KeySFC. To this end, we analyze latency growth by increasing chain length from 1 ($VM_S \rightarrow SF1 \rightarrow VM_D$) to 10 ($VM_S \rightarrow SF1 \rightarrow SF2 \rightarrow SF3 \rightarrow SF4 \rightarrow SF5 \rightarrow SF6 \rightarrow SF7 \rightarrow SF8 \rightarrow SF9 \rightarrow SF10 \rightarrow VM_D$). In each of these steps, we add one extra SF between VM_S and VM_D.

We analyze two placement scenarios: when all SFs are allocated in the same server (Fig. 16(a)), and when consecutive SFs are allocated in alternating servers (Fig. 16(b)). In the first scenario, the hops between SFs are executed in the *Eswitch* of the same server, so packets do not need to be routed in the physical network. On the other hand, in the second scenario, each segment in the overlay is mapped to one hop in the physical layer, passing through the *Eswitches* and *Cswitches* of two servers. The second scenario is also relevant to prove that KeySFC is able to represent loops in the physical network, considering the end-to-end chain.

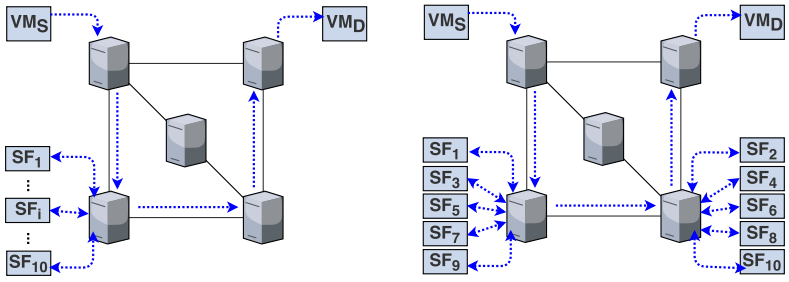
Fig. 17 shows average latency values (95% confidence interval) measured using ICMP in different chain lengths for both scenarios. The bars represent the standard deviation of our measurements, and each test was run 60 times. Both charts show that latency grows in a close-to-linear fashion for up to 10 SFs in the chain. As the second scenario (Fig. 16(b)) involves routing in the physical network, we note an overall increase in the latency measurement. Nevertheless, this growth goes only from 3.2*ms* in the first scenario to 4.3*ms* in the second scenario, even in the worst case when the chain length is 10, showing the routing in the physical network is efficient.

5.5. Traffic engineering enabled by KeySFC

One of the main contributions of KeySFC is to provide the appropriate SFC mechanisms to the NFVO, so traffic engineering can select any path and take optimal placement and chaining decisions. This section explores some traffic engineering scenarios that can be enabled by KeySFC.

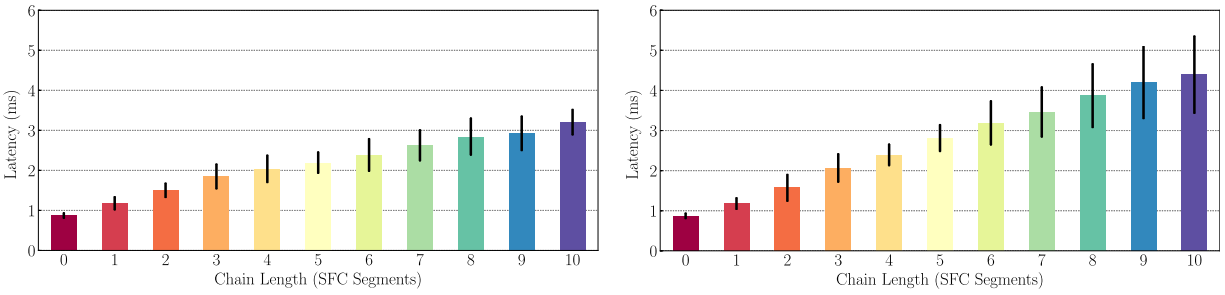
5.5.1. Allocation of maximum bandwidth with path migration

Consider the NFVO has to embed the following SFC request in our test topology: $VM_S \rightarrow SF1 \rightarrow VM_D$. The goal is to dynamically provide the maximum bandwidth to a TCP flow in an SFC.



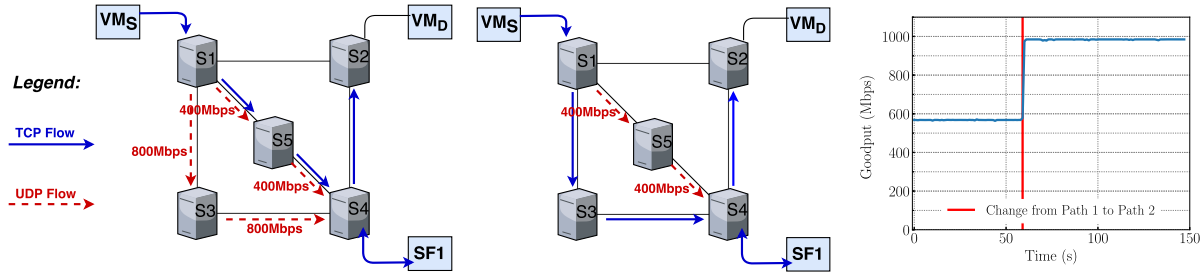
(a) All SFs in the same server. (b) Consecutive SFs in alternating server.

Fig. 16. Scenarios for evaluating the impact of SFC length.



(a) Scenario of Fig. 16(a). (b) Scenario of Fig. 16(b).

Fig. 17. Latency results when chain length varies from 1 to 10.



(a) Path1: concurrent 400Mbps UDP traffic. (b) Path2: no concurrent traffic. (c) Throughput at VM_D.

Fig. 18. Traffic engineering for migration from Path 1 to Path 2.

Fig. 18(a) shows VM_S and VM_D are allocated in servers S1 and S2, respectively. Besides, there are two different paths with length 2 that connect VM_S to SF1. Initially, these paths present the following background traffic:

- Path 1: S1 → S5 → S4: with 400 Mbps UDP traffic.
- Path 2: S1 → S3 → S4: with 800 Mbps UDP traffic.

At the beginning of this experiment, the SFC request is embedded in Path 1, as shown in Fig. 18(a), because it is the best choice. We send a TCP flow using `iperf` that will use all the available bandwidth in the SFC. Fig. 18(c) shows that the throughput at VM_D starts at approximately 560 Mbps.

As NFV demands are very dynamic, at any moment, the NFVO may discover there is an alternative to improve the bandwidth utilization for the TCP flow. This happens at instant 60 s, when Path 2 becomes idle and the NFVO migrates this TCP flow from Path 1 to Path 2, causing the throughput at VM_D to increase to 980 Mbps, as shown in Fig. 18(b) and (c).

To perform the described path migration, the NFVO communicates with the SDN Controller, which only have to execute a very simple `flow mod` operation to modify a single flow entry at the *Eswitch* of S1. Table 2 shows the original flow entry that embeds VMAC1 into the packet, which contains the *routeID* for Path 1. For selecting Path 2, the only field that has to change is the action "DstMAC = VMAC2", shown in the second line of Table 2, as VMAC2 embeds the new route through Path 2. Even for longer paths, no other changes would be required in the hops along the path, thanks to strict SR. Once this single flow mod operation is executed, all the packets that leave VM_S will be tagged with the *routeID* of the new path.

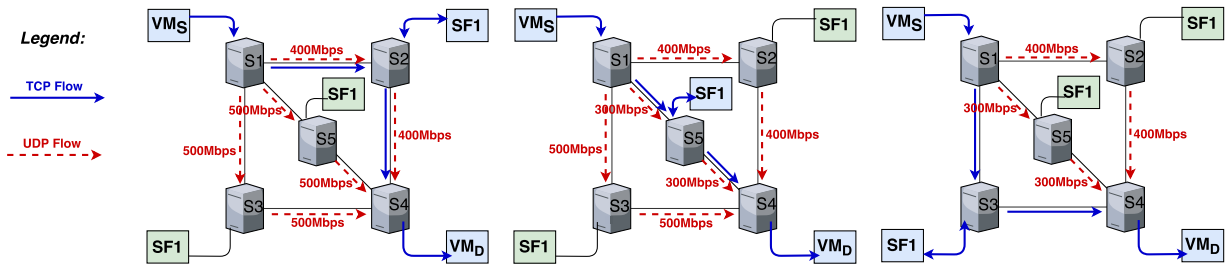
Comparing KeySFC with traditional SFC schemes based on SDN approaches, the path migration would involve changing flow entries in all the hops along the old and in the new paths. Depending on the path length, this can represent a long delay to update a large number of flow entries in switches that might be distributed, leading to consistency problems and packet loss in a TCP flow. This time to reconfigure a path becomes even more significant for chains with a large number of segments.

5.5.2. Agile path selection with replication of SFs

Apart from seamless path migration, the KeySFC mechanism allows to use replication of distributed SFs instances that might be selected on demand. In this way, the NFVO component can offer an extra degree of freedom by allowing the migration to a path that uses a replicated SF instance. The goal is still to dynamically

Table 2
Flow entries at S1 for paths 1 and 2.

Path	Server	Match	Action
1	S1	SrcIP = IP VM_S; L4DstPort = P1	DstMAC = VMAC1; Output to *Cswitch*
2	S1	SrcIP = IP VM_S; L4DstPort = P1	DstMAC = VMAC2; Output to *Cswitch*



(a) Path 1: 600Mbps UDP traffic. (b) Path 2: 400Mbps UDP traffic. (c) Path 2: no concurrent traffic.

Fig. 19. Traffic engineering scenarios for different paths using SF redundancy.

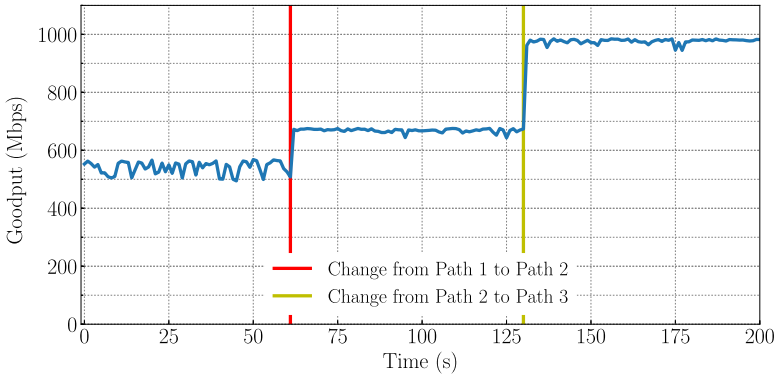


Fig. 20. Throughput results when SFC migrates from Path 1 to Path 2, and from Path 2 to Path 3 (see Fig. 19).

achieve the maximum available bandwidth for a TCP flow that crosses $VM_S \rightarrow SF1 \rightarrow VM_D$.

Fig. 19(a) shows VM_S and VM_D are allocated in servers S1 and S4, respectively. Besides, there are three instances of the SF of type SF1 allocated in servers S2, S3, and S5. Also, there are three different paths with length 3 that connect VM_S to VM_D passing through one instance of SF1. In the initial condition, these paths present the following background traffic:

- Path 1: S1 → S2 → S4: with 400 Mbps UDP traffic.
- Path 2: S1 → S5 → S4: with 500 Mbps UDP traffic.
- Path 3: S1 → S3 → S4: with 500 Mbps UDP traffic.

Initially, the SFC request is embedded in Path 1, as shown in Fig. 19(a), and, as in the previous experiment, we send a TCP flow using `iperf` to use all the available bandwidth. Fig. 20 shows the flow is able to achieve approximately 530 Mbps at VM_D.

At instant 60 s, the NFVO detects the utilization of Path 2 decreased to 300 Mbps, and decides to migrate the SFC from Path 1 to Path 2 using an extra SF1 instance that was already provisioned, as shown in Fig. 19(b). In a similar way, at instant 130 s, Path 3 becomes idle, and the SFC migrates from Path 2 to Path 3, as shown in Fig. 19(c). Fig. 20 presents throughput results when NFVO migrates the SFC from Path 1 to Path 2, and from Path 2 to Path 3. In this test, we can see that the first migration was able to increase the bandwidth from about 530 Mbps to about 660 Mbps, and the second migration was able to increase the bandwidth to about 980 Mbps.

The cost of performing these operations in terms of flow entry modifications is the same as the previous example: a *flow mod* operation. We highlight here one of the main contributions of our KeySFC scheme: NFVO takes placement, routing, and chaining decisions in an integrated way. This integration allows traffic engineering tasks to make optimal use of networking and processing resources. In most of current SFC approaches, routing is a separate responsibility of the data plane, so SFC decisions are not integrated to routing decisions. Also, data plane routing frequently restricts the NFVO to choose one from a set of shortest paths, and, consequently, prevents traffic engineering to find an optimal path for an SFC.

6. Related works

6.1. RNS-based SSR

The idea of using RNS for packet forwarding was firstly explored by Wessing et al. [38] and applied to optical packet-switched networks to avoid header rewriting and label distribution protocols. This idea was further integrated with SDN in core packet-switched networks by KeyFlow [6], which builds a fabric model that replaces the table lookup in the forwarding engine by elementary operations relying on RNS. Then, other works, such as KeySet [43] and KAR [39] explored additional properties of RNS to extend KeyFlow. KeySet explored techniques to reduce the length of the forwarding label, while KAR deviates packets from faulty links with routing deflections and guides them to their original destination due to resilient paths added to the forwarding label. However, all these works applied RNS to routing in core networks and did not explore how to use these concepts for DCNs where

virtualized SFs need to be chained to provide a service. In turn, the works in [40,42] evaluate the scalability of a RNS-based routing system for DCNs with 2-tier Clos topologies, but they did not explore a solution for SFC.

The first work to suggest the application of RNS in SFC was [49], but it does not give any details on how to implement the proposal with virtualized SFs in a DC. In CRT-Chain [13], the authors proposed a more elaborate scheme for SFC using two RNS labels: one for routing in the physical layer, and one for routing between SFs in an overlay layer. However, their simulated results focused only on reducing the labels length, and they did not provide any implementation and validation for data and control planes. Moreover, their theoretical proposal presents some critical limitations for real-world scenarios: (i) in contrast to our approach that performs one modulo operation per core node, their data plane is composed by a loop that performs a modulo operation for each SF attached to each forwarder in the path (even the SFs that are not part of the chain), which will add a huge delay in the end-to-end service latency; (ii) it does not allow loops in the physical layer, which is not acceptable if we consider that a traffic may be steered back and forward in the physical network to traverse different SFs; and (iii) it does not allow more than one SF of the same type in the same forwarder. Thus, to the best of our knowledge, KeySFC is the first work that proposes a complete SFC scheme using RNS that is implementable and efficient for DCNs.

6.2. Traffic steering

Based on the RFC 4665 architecture [1], many SFC solutions have been proposed. Several of them employ NFV and SDN to enable network services, as surveyed in [21,50]. In [51], the authors classify SFC approaches in four types, based on their forwarding methods: (1) flow identifiable information; (2) stacked headers; (3) service chain identifiers with extra header; and (4) service chain identifiers with overload of existing address field.

In this section, we use this classification to compare our work with related works that also present traffic steering schemes. To this end, we select the main representatives of each of these four methods to compare the details of these solutions with KeySFC. Table 3 summarizes this comparison, classifying then according to the techniques used for SFC, routing, and control. Extensive surveys on traffic steering methods can be found on [8,21,50].

Method 1 consists of classifying packets based on flow identifiable information, such as a 5-tuple (e.g., source IP and port, destination IP and port, and L4 protocol), at SFF. A central entity configures each SFF with flow rules to forward packets to the next SF in the chain depending on information contained in packet headers. This method does not demand changes either to the network or the original packet format, because it is based on flow tables and the SDN control plane. On the other hand, when we analyze scalability, it is the least suitable for large networks with a high number of flows, and requires per-hop table lookups. An example of this approach is StEERING [52]: the SDN controller installs flow entries into the switches and handles the service placement, and OpenFlow-enabled switches are responsible for traffic classification and steering. It allows for fine granular application policies and uses multiple forwarding tables to reduce the number of flow rules, but it still suffers from scalability and agility issues because of the large number of flow rules distributed in the switches [21].

In **Method 2**, the Classifier analyzes each packet entering the SFC domain, and stacks a series of packet headers over the packet. Each header includes a network address (e.g., using MPLS or IPv6), and the outermost header addresses the next destination. After receiving the packet, each SF or SFF processes it, removes the outermost header, and uses the next header to find the next SF. This process repeats until all stacked headers are removed and the packet reaches its final destination. This approach may present MTU, fragmentation, and scalability issues when the number of SFs is high.

One of the most important works that uses Method 2 is Segment Routing, which is a flexible source routing method that can enable SFC [34]. It can be implemented by using a new type of IPv6 header that encodes segments as a list of 128-bit IPv6 addresses, but this approach present large overhead [35]. Also, it can be implemented using MPLS with no modification, since an ordered list of segments can be encoded as a stack of MPLS labels. The next segment to be process is popped from the top of the stack after the completion of a segment, and a lookup operation in the forwarding table is performed in each hop. Differently to traditional MPLS networks, Segment Routing with MPLS distributes segment labels using simple extensions to current IGP protocols, and LDP and RSVP-TE are no longer required for populating the forwarding tables [53]. Besides, scalability is improved, because only source nodes maintain state information and the size of forwarding table remains constant regardless of the number of paths [36].

The paths can be derived from a IGP Shortest Path First (SPF) algorithm, which allows a packet to be forwarded along the Equal Cost Multi Path (ECMP)-aware shortest path, or from a Segment Routing Traffic Engineered (SR-TE) path that allows a packet to be steered along an explicit SSR path by using a combination of one or more shortest segments [35]. However, this flexibility comes at the expense of increased packet overhead and increased label processing in the network [53]. Moreover, the source node needs to push large segments list in order to realize long explicit paths, but

Table 3
Comparison between related works.

SFC proposal	Proto- type	SFC method	SFC encap.	Routing decision	Routing computation	Topology	Control Plane
StEERING [52]	X	(1) Flow identifi- able information	None	Per-hop routing	Table-based	Network-centric	OpenFlow
Segment Routing [34]	X	(2) Stacked headers	MPLS or IPv6	Loose source routing	Table-based & Algorithmic (list pop)	Network-centric	IS-IS, BGP or OSPF
NSH [5]	X	(3) Service chain identifier, extra header	VXLAN, GRE, or Ethernet	Per-hop routing	Table-based	Network-centric	BGP or NSH Extension for OpenFlow
NetFloc [57]	X	(4) Service chain identifier, existing header	MAC address	Per-hop routing	Table-based	Network-centric	OpenFlow
VirtPhy [2]	X	(4) Service chain identifier, existing header	MAC address	Per-hop routing	Algorithmic (XOR)	Server- centric	OpenFlow
CRT- Chain [13]		(4) Service chain identifier, existing header	MPLS or IPv6	Strict source routing	Algorithmic (modulo)	Network-centric	Not described
KeySFC	X	(4) Service chain identifier, existing header	MAC address	Strict source routing	Algorithmic (modulo)	Any	OpenFlow

most MPLS equipments can support a limited label stack depth (about 3 to 5 labels), which may lead to inefficient traffic distribution and network congestion [35]. In this way, the traffic engineering algorithms have to take into account specific constraints of path encoding as additional objective functions [36]. Thus, although Segment Routing can be used for enabling SSR, it commonly uses a loose SR scheme that specifies a list with some SFFs and SFs that the packet must go through, and delegates the routing between these elements to the underlying network that employs ECMP shortest paths.

In contrast to Segment Routing, KeySFC does not restrict the path selection by traffic engineering, eliminates forwarding tables, and performs simple operations over path identifiers for computing the output port in each forwarding element. KeySFC approach (as any other SSR approach that encodes the path in the packet header) also has to consider the scalability of the routing header, as discussed in Section 3.2. To the best of our knowledge, KeySFC is the first work to propose and implement a traffic steering solution that uses algorithmic SSR and can be applied to any topology.

In **Methods 3 and 4**, each chain receives a unique service chain identifier from Classifier. Also, SFFs receive forwarding configuration, and consult this information to discover the next hop based on the packet's chain identifier. In this way, flow entries are defined in a per-chain basis, instead of in a per-flow basis, using less flow entries when compared to Method 1.

This chain identifier can be included in a new specific header, as adopted in **Method 3** and implemented in network service headers (NSH) solutions [5]. The NSH header carries two chain identifiers: a 3-byte Service Path Identifier (SPI), and an 1-byte Service Index (SI), the former indicating the selected chain, and the latter the current position in the chain. The SI is decremented at each step of execution by each SF (or by a Proxy). The SFFs in the path uses the SPI for performing a table lookup and deciding which SF should be executed at any given time for each chain. Then, the address of the next networking element is encapsulated in an outermost header, relying on the underlying routing methods to deliver the packet between these elements. At least two works implement SFC solutions using NSH [54,55].

Although the NSH protocol presents great flexibility, it is necessary to change all the forwarding elements and control mechanisms in the infrastructure to support it. In addition, attachment of headers expands packet size, causing an increase of traffic, and potential problems with MTU to handle fragmentation. Also, there is a strong separation between the service overlay layer and the underlay network layer. In other words, the SFC mechanism does not give any instruction about how to forward packets in the network (only the order of services that must be traversed) [5]. Therefore, the routing decisions are decoupled from the chaining decisions and executed by different mechanisms. Furthermore, this scheme uses per-hop tables both for discovering the next SF and routing packets in each SFC segment, which leads to scalability and agility issues.

Other alternative, used by **Method 4**, is to include the chain identifier in existing packet fields, like MAC address [56]. In [57], the authors propose an SFC mechanism named Netfloc, where packets entering a service chain invoke new flows in the first and the last bridges of that service chain, which rewrite the original MAC address to a virtual address. Traffic steering relies on L2-based OpenFlow rules, processed by SDN switches that are configured by OpenDayLight SDN controller [21]. However, their approach relies on various per-hop table lookups and presents limited scalability and agility.

KeySFC and CRT-Chain [2,13] also rewrite existing headers to include chain identification, but they can achieve better scalability and agility by replacing per-hop table lookups with algorithmic SSR using *modulo* operations. VirtPhy [2] also explores algorithmic forwarding and MAC rewrite for SFC with the use of a high performance forwarding mechanism based on XOR operations, but it does not allow the specification of underlay paths.

Aside from KeySFC and VirtPhy, all the described related works are tailored to network-centric topologies. VirtPhy specifies how commodity servers can be directly interconnected in server-centric topologies to provide SFC. KeySFC, on the other hand, supports any DCN topology.

Regarding the control plane, StEERING, NetFloc, VirtPhy, NSH, and KeySFC support OpenFlow. In Segment Routing, segments are allocated and signaled by IS-IS, OSPF or BGP. NSH also supports BGP in the control plane. Except for CRT-Chain, all described works present implementations of their mechanisms. Especially, NSH and Segment Routing received a lot of attention from academia and industry with several real-world deployments.

7. Conclusions and future work

NFV challenges network operators to compose services by steering dynamic flows across a set of SFs, with demanding performance requirements. Traditional SFC solutions lack flexibility and efficiency due to the fact that they are adaptations of the classical source-to-destination routing paradigm. As a result, one may not be able to use all the existing paths for chaining specific segments, either because the SFC decisions are completely decoupled from the routing decisions, or because the path choice is constrained by the routing mechanism to a small set of shortest paths. In addition, dynamic modification of chain paths may impact a large set of nodes, leading to large operational complexities even for simple service modification tasks.

This paper proposed, implemented, and evaluated KeySFC: a traffic steering scheme built up over strict source routing designed as a fabric network, separating edge and core switching elements. With our proof-of-concept prototype orchestrated by OpenStack we demonstrated that KeySFC is a feasible solution to the dynamic SFC traffic steering problem in production DCs. While current solutions are tailored to network centric DCNs, we showed that it is possible to deploy SFC in any DCN topology, extending the infrastructures that can be used to provide SFC. In addition, performance results indicated that the selected SSR mechanism based on RNS can provide efficient packet transport with low latency and low jitter.

Moreover, our results show KeySFC can provide a programmable, expressive, scalable, and agile traffic steering scheme. Using our expressive solution, the traffic engineering can specify any path between two endpoints of an SFC segment, and agilely migrate between paths using SDN in a programmable manner. Furthermore, our solution only needs to install flow rules in the SFC segments' endpoints, significantly improving scalability and agility when compared to traditional table-based approaches. Thus, our solution enables dynamic and efficient orchestration of the underlay of DCNs, and does not restrict the traffic engineering on the selection of paths for SFC.

As future work, we will perform a more extensive performance and scalability comparison with related works to properly quantify the gains obtained with KeySFC. Besides, this work can be extended with the exploitation of RNS properties to support security and resilience use cases. In adittion, offload of forwarding tasks and SFs to a networking co-processor, such as SmartNICs, is planned. Performance tests in large DCN topologies with realistic SFs, and exploitation of P4 language are also in our roadmap. As carrier-grade services can benefit from our mechanisms, the integration of KeySFC with operation and management (OAM) systems will be addressed. Finally, online optimization strategies for resource allocation in multi-domain SFC scenarios [58] are important extensions of this work.

Declaration of Competing Interest

The authors declare that they do not have any financial or non-financial conflict of interests.

Acknowledgments

This work has received funding from CNPq (grant agreements 432787/2016-0, and 428311/2018-0) and FAPES (grant agreements 94/2017, 560/2018, and 269/2019). In addition, it is part of the FUTEBOL project, which has received funding from the European Union's Horizon 2020 for research, technological development, and demonstration under grant agreement no. 688941 (FUTEBOL), as well from the Brazilian Ministry of Science, Technology and Innovation (MCTI) through RNP and CTIC. It was also financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2019.106975.

References

[1] J. Halpern, et al., RFC 7665: Service function chaining (SFC) architecture, Technical Report, RFC Editor, 2015.
[2] C.K. Dominicini, et al., VirtPhy: fully programmable NFV orchestration architecture for edge data centers, IEEE TNSM 14 (4) (2017) 817–830.
[3] P. Quinn, et al., Problem Statement for Service Function Chaining, RFC 7498, RFC Editor, 2015.
[4] S.A. Jyothi, et al., Towards a flexible data center fabric with source routing, in: SOSR, ACM, 2015, pp. 10:1–10:8.
[5] P. Quinn, et al., Network Service Header (NSH), RFC 8300, RFC Editor, 2018.
[6] M. Martinello, et al., KeyFlow: a prototype for evolving SDN toward core network fabrics, IEEE Netw 28 (2) (2014) 12–19.
[7] M. Soliman, et al., Source routed forwarding with software defined control, considerations and implications, in: CoNEXT Student Workshop, ACM, 2012, pp. 43–44.
[8] D. Bhamare, et al., A survey on service function chaining, JNCA 75 (2016) 138–155.
[9] X. Jin, et al., Your data center switch is trying too hard, in: SOSR, ACM, 2016, pp. 12:1–12:6.
[10] F.P. Tso, et al., Network and server resource management strategies for data centre infrastructures: a survey, Comput. Netw. 106 (2016) 209–225.
[11] C.A. Sunshine, Source routing in computer networks, SIGCOMM Comput. Commun. Rev. 7 (1) (1977) 29–33.
[12] C. Filsfils, et al., The segment routing architecture, in: GLOBECOM, IEEE, 2015, pp. 1–6.
[13] Y. Ren, et al., On scalable service function chaining with $\mathcal{O}(1)$ flowtable entries, in: INFOCOM, 2018, pp. 702–710.
[14] C. Guo, et al., Secondnet: a data center network virtualization architecture with bandwidth guarantees, in: Co-NEXT, ACM, 2010, pp. 15:1–15:12.
[15] B. Stephens, et al., A scalability study of enterprise network architectures, in: ANCS, IEEE, 2011, pp. 111–121.
[16] M. Casado, et al., Fabric: a retrospective on evolving SDN, in: HotSDN, ACM, 2012, pp. 85–90.
[17] Z. Li, et al., GBC3: a versatile cube-based server-centric network for data centers, TPDS 27 (10) (2016) 2895–2910.
[18] ETSI ISG, NFV 002 V1.2.1. Network Functions Virtualisation (NFV); Architectural Framework, 2014.
[19] J. Zhang, et al., Enabling efficient service function chaining by integrating NFV and SDN: architecture, challenges and opportunities, IEEE Netw. 32 (2018) 152–159, doi:10.1109/MNET.2018.1700467.
[20] R. Mijumbi, et al., Management and orchestration challenges in network functions virtualization, IEEE Commun. Mag. 54 (2016) 98–105.
[21] H. Hantouti, et al., Traffic steering for service function chaining, IEEE Communications Surveys Tutorials 21 (2019) 487–507, doi:10.1109/COMST.2018.2862404. 1–1
[22] J.G. Herrera, et al., Resource allocation in NFV: a comprehensive survey, IEEE TNSM 13 (3) (2016) 518–532.
[23] M.C. Luizelli, et al., A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining, Comput. Commun. 102 (2017) 67–77.
[24] M.C. Luizelli, et al., Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions, in: IM, IEEE, 2015, pp. 98–106.
[25] F. Bari, et al., On orchestrating virtual network functions, in: CNSM, IEEE, 2015, pp. 50–56.
[26] F. Bari, et al., Orchestrating virtualized network functions, IEEE TNSM 13 (4) (2016) 725–739.
[27] M. Mechtri, et al., A scalable algorithm for the placement of service function chains, IEEE TNSM 13 (3) (2016) 533–546.
[28] M.T. Beck, et al., Scalable and coordinated allocation of service function chains, Comput. Commun. 102 (2017) 78–88.
[29] D. Bhamare, et al., Optimal virtual network function placement in multi–cloud service function chaining architecture, Comput. Commun. 102 (2017) 1–16.
[30] G. Sun, et al., Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks, Future Gener. Comput. Syst. 91 (2019) 347–360.
[31] A. Leivadeas, et al., Optimal virtualized network function allocation for an SDN enabled cloud, Comput. Stand. Interfaces 54 (2017) 266–278.
[32] S. Draxler, et al., JASPER: joint optimization of scaling, placement, and routing of virtual network services, IEEE TNSM 15 (3) (2018) 946–960.
[33] G. Miotto, et al., Adaptive placement & chaining of virtual network functions with NFV-PEAR, JISA 10 (1) (2019) 3.
[34] F. Clad, et al., Segment Routing for Service Chaining, Internet-Draft, IETF, 2018.
[35] Z.N. Abdullah, et al., Segment routing in software defined networks: a survey, IEEE COMST 21 (1) (2019) 464–486.
[36] E. Moreno, et al., Traffic engineering in segment routing networks, Comput. Netw. 114 (2017) 23–31.
[37] D. Abts, et al., High performance datacenter networks: architectures, algorithms, and opportunities, Synth. Lect. Comput. Archit. 6 (1) (2011) 1–115.
[38] H. Wessing, et al., Novel scheme for packet forwarding without header modifications in optical networks, J. Lightwave Technol. 20 (8) (2002) 1277–1283.
[39] R.R. Gomes, et al., KAR: key-for-any-route, a resilient routing system, in: DSN Workshop, 2016, pp. 120–127.
[40] M. Martinello, et al., Programmable residues defined networks for edge data centres, in: CNSM, 2017, pp. 1–9.
[41] V. Shoup, A Computational Introduction to Number Theory and Algebra, Cambridge University Press, 2009.
[42] A. Liberato, et al., RDNA: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters, IEEE Trans. Netw. Serv. Manag. 15 (4) (2018) 1473–1487.
[43] Y. Ren, et al., Flowtable-free routing for data center networks: a software-defined approach, in: GLOBECOM, 2017, pp. 1–6.
[44] C. Dominicini, et al., KeySFC: agile traffic steering using strict source routing, in: SOSR, ACM, 2019, pp. 154–155.
[45] , ETSI, NFV Release 2 Description - v1.9.0, Technical Report, European Telecommunications Standards Institute, 2019.
[46] R. Chaparadza, et al., SDN enablers in the ETSI AFI GANA reference model for autonomic management control and virtualization impact, in: IEEE Globecom Workshops, 2013, pp. 818–823.
[47] H. Hantouti, et al., Traffic steering for service function chaining, IEEE COMST 21 (1) (2019) 487–507.
[48] Openstack, OpenStack Open Source Cloud Computing Software, 2017. https://www.openstack.org/software/.
[49] M. Zhao, et al., An RNS-based forwarding approach to implement SFC, in: ICTC, IEEE, 2017, pp. 679–682.
[50] A.M. Medhat, et al., Service function chaining in next generation networks: state of the art and research challenges, IEEE Commun. Mag. 55 (2) (2017) 216–223.
[51] S. Homma, et al., Analysis on Forwarding Methods for Service Chaining, Internet-Draft, IETF, 2016.
[52] Y. Zhang, et al., StEERING: a software-defined networking for inline service chaining, in: ICNP, IEEE, 2013, pp. 1–10.
[53] R. Bhatia, et al., Optimized network traffic engineering using segment routing, in: INFOCOM, 2015, pp. 657–665.
[54] G. Li, et al., Fuzzy theory based security service chaining for sustainable mobile-edge computing, Mobile Information Systems 2017 (2017).
[55] V. Mehmeri, et al., Optical network as a service for service function chaining across datacenters, in: OFC, 2017, pp. 1–3.
[56] P. Bottorff, et al., Ethernet MAC Chaining, Internet-Draft, IETF, 2017.
[57] I. Trajkovska, et al., SDN-based service function chaining mechanism and service prototype implementation in NFV scenario, in: Computer Standards and Interfaces, 2017, pp. 247–265.
[58] D. Dietrich, et al., Multi-provider service chain embedding with Nestor, IEEE TNSM 14 (1) (2017) 91–105.

Cristina K. Dominicini received her B.Sc. degree in computer engineering from Federal University of Espírito Santo (UFES), Brazil, her M.Sc. degree in electrical engineering from University of Sao Paulo (USP), Brazil, and her PhD degree in Computer Science at Federal University of Espírito Santo (UFES), Brazil, in 2009, 2012, and 2019, respectively. Since 2014, she is an assistant professor in the Department of Informatics at Federal Institute of Education, Science and Technology of Espírito Santo (IFES), Brazil. Her research interests include NFV, SDN, and next generation networks.

Gilmar L. Vassoler received his B.Sc. degree in computer engineering from Federal University of Espírito Santo (UFES), Brazil, his M.Sc. and Ph.D. degree in electrical engineering from Federal University of Espírito Santo (UFES), in 2000, 2003, and 2015, respectively. Since 2006, he is a lecturer the Department of Informatics at Federal Institute of Education, Science and Technology of Espírito Santo (IFES), Brazil. His research interests include DCN, SDN and NFV.

Rodolfo V. Valentim is a Master's student in Computer Science at Federal University of Espírito Santo (UFES), Brazil. He received her B.Sc. degree in computer engineering from Federal University of Espírito Santo (UFES), Brazil. Since 2016 is a member in Group of Studies in Software Defined Networks (NERDS). His research interests are NFV, SDN, cloud computing, embedded systems, and neural networks.

Rodolfo S. Villaca is an assistant professor at the Federal University of Espírito Santo (UFES) in Industrial Technology Department (DTI). Received his Ph.D. in Computer Engineering in 2013 at the University of Campinas (Unicamp). He also holds the Computing Engineering degree and M.Sc. degree in Electrical Engineering from the Federal University of Espírito Santo (UFES). His research Interests are: Computing Systems and Networks.

Moises R. N. Ribeiro received his B.Sc. degree in electrical engineering from the Instituto Nacional de Telecomunicações, Brazil, his M.Sc. degree in telecommunications from the Universidade Estadual de Campinas, Brazil, and his Ph.D. degree from the University of Essex, United Kingdom, in 1992, 1996, and 2002, respectively. In 1995, he joined the Department of Electrical Engineering of Federal University of Espírito Santo. He was a visiting professor at Stanford University in 2010–2011 with the Photonics and Networking Research Laboratory. His research interests include fiber optic communication and sensor devices, systems, and networks.

Magnos Martinello received his B.Sc. degree in computer science from Universidade Federal do Parana (UFPR), his M.Sc. degree in computer systems engineering from Universidade Federal do Rio de Janeiro (UFRJ), and his Ph.D. degree from the Institut National Politechnique de Toulouse (INPT) France in 1998, 2000, and 2005, respectively. In 2008, he joined the Informatics Department at Federal University of Espírito Santo (UFES). He has worked as a visiting researcher at University of Bristol in 2016–2017 within the High Performance Network group. His research interests include SDN, NFV, Cloud Computing and Network Performance Analysis.

Eduardo Zambon is an assistant professor at the Computer Science Department of the Federal University of Espírito Santo (UFES), Brazil. He holds a B.Sc. degree in Computer Engineering and a M.Sc. degree in Computer Science, both obtained from UFES in 2003 and 2006, respectively. In 2013, he received a Ph.D. in Theoretical Computer Science from the University of Twente, in the Netherlands. His research interests include formal methods for system verification, in particular of computer networks.