Dynamic In-Network Classification for Service Function Chaining ready SDN networks

Marco Polverini*, Jaime Galán-Jiménez†, Francesco G. Lavacca‡, Antonio Cianfrani* and Vincenzo Eramo*
*DIET Department, University of Rome "Sapienza", Rome, Italy. [name.surname]@uniroma1.it
†Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain. jaime@unex.es
‡Fondazione Ugo Bordoni, Italy. fglavacca@fub.it

***Abstract*—Service Function Chaining (SFC) paradigm consists in steering traffic flows through an ordered set of Service Functions (SFs) so that to realize complex end to end services. SFC architecture introduces all the logical functions that need to be developed in order to provide the required service. The SFC overlay infrastructure can be built on top of many different underlay network technologies. The high flexibility and centrally controlled feature of Software Defined Networking (SDN), make SDN networks to be a perfect underlay to build the SFC architecture. Due to Ternary Content Address Memory (TCAM) limited size, SDN switches have a limitation in the number of flow rules that can be hosted. This constraint is particularly penalizing in case of the SFC classifier function, since it requires to manage a high number of different flows. The limitation imposed by the TCAM size on the SFC classifier can be a bottleneck for the number of SFC requests that the SDN-based SFC architecture can handle. In this paper we define the Dynamic Chain Request Classification Offloading (D-CRCO) problem, as the one of maximizing the number of accepted SFC requests, having the possibility of: i) implement the SFC classifier also in a node that is internal to the SDN-based SFC domain, and ii) install classification rules in a reactive fashion. Furthermore, we propose the Dynamic Nearest Node (DNN) heuristic to solve the D-CRCO problem. Performance evaluation shows that by using DNN heuristic it is possible to triple the number of accepted requests, with respect to existing solutions.**

***Index Terms*—SDN, NFV, SFC, TCAM.**

I. Introduction

Thank to the advent of Network Function Virtualization (NFV) technology, Service Providers can provide complex services to end users, without the need to make strong Capital Expenditure (CAPEX) investments in the hardware infrastructure. In fact, by steering traffic flows through an ordered set of Virtual Network Functions (VNFs), it is possible to realize any kind of service, by only relying on software applications. This paradigm is also known as Service Function Chaining (SFC) [1]. The most appealing feature from the Service Provider perspective is the possibility to scale up or down the involved physical infrastructure, as a function of the actual volume of traffic to serve, with a consistent reduction of the Operational Expenditure (OPEX).

In order to force traffic flows to go through the ordered set of required VNFs, packets must be encapsulated with a Network Service Header (NSH) [2]. This process is accomplished in two steps: i) as first the incoming traffic must be classified, then ii) the NSH has to be inserted on top of each incoming packet. In the reference SFC architecture, this task is accomplished by a single device, named SFC classifier. An SFC classifier is a border router, that receives traffic flows from external networks, and inject them into the SFC domain. Then, the entire classification process is distributed among few devices, that must host specific flow rules in order to steer the incoming traffic through the required chain.

Flow table in network nodes are realized by means of TCAMs. This type of memory is known to have very high performance in terms of time required to access the contents, but also to be limited in storage capacity. TCAMs constitute a bottleneck for different applications, such as Traffic Engineering [3], Green Networking [4], Traffic Measurements [5], etc.

In [6] we have investigated the problem of limited TCAM size in an SDN-based SFC architecture, showing how the classification process can easily become a bottleneck for the number of served SFC requests. To overcome this limitation, we have proposed the classification offloading mechanism. The idea is to let also nodes that are internal to the SFC domain to host classification rules. This simple modification highly increases the number of served requests, having as drawback the increase on the path length (and a consequent bandwidth wastage).

In this work we introduce an enhancement of the model presented in [6] to further increase the number of served requests. Specifically, we include dynamic behaviour in the set of SFC requests, which simply means that a request can be idle or active over time. When the request is idle, it does not inject traffic in the network, and consequently the related classification rule can be removed, creating room in the TCAMs of the network nodes to host more classification rules.

Specifically, the main contributions of this work are:

- we improve the model described in [6], adding the dynamic behaviour to the set of requests;
- we propose an heuristic algorithm to maximize the number of served SFC requests;
- we evaluate the impact of the proposed solution by means of simulation experiments.

The rest of the paper is organized as follows: in Sec. II related works are discussed, in Sec. III we describe the considered SDN-based SFC architecture and detail the classification offloading mechanism, in Sec. IV the system model is presented and the Dynamic Chain Request Classification

978-1-7281-4445-0/19/$31.00 ©2019 IEEE

Offloading (D-CRCO) problem is defined, in Sec. V we introduce the DNN heuristic to solve the D-CRCO problem. Sec. VI contains the performance evaluation, while conclusion and future works are presented in Sec. VII.

II. Related Work

The Dynamic Chain Request Classification Problem (D-CRCO) aims at increasing the number of accepted requests (or reducing the blocking probability) due to the TCAM size limitation at the SFC classifier, in the context of an SDN-based SFC architecture. In order to validate the considered scenario and show the novelty of the studied problem, in the following an overview of the state-of-the-art is presented. In particular, related works are divided in three categories: i) integration of NFV/SDN paradigms with SFC; ii) NFV placement and SFC routing optimization; and iii) TCAM size limitation problem.

A. Integration of NFV/SDN with Service Function Chaining

In [7], a survey on the existing technology for traffic steering in SDN-based SFC domains is presented. Specifically, the programmability feature of SDN is exploited to build an overlay SFC infrastructure on top of an underlay SDN network. Traffic steering types are divided into three main categories: i) header-based methods, such as NSH or Segment Routing (SR) encapsulation , ii) tag-based methods, which exploit existing header fields (VLAN, Source MAC, Type of Service, etc.) to encode SFC related information, and iii) programmable switch-based methods, which aim at re-classifying the traffic at every SFC element (both classifier and Service Function Forwarders (SFF)).

An example of SDN-based SFC domain is OpenSCaaS, an open platform for service chain as a service [8]. OpenSCaaS uses a tag-based traffic steering method, by encoding the Service Chain Identifier (SC-ID) in the Source MAC to guarantee scalability. The SDN controller can install SFC classification rules both proactively or in a reactive fashion.

Another example of SDN-based SFC architecture is presented in [9], where programmable OpenFlow-based switches are used to realize SFCs, leveraging simple L2 forwarding, thus avoiding the creation of an overlay network by means of packet encapsulation.

B. Efficient resource allocation

In NFV environment, there are some open optimization problems focusing on energy-efficient resource usage maximization. Authors in [10] face the problem by proposing a power-aware VNF placement and routing approach, while in [11] it is proposed a novel resource architecture which enables an energy efficient SFC that keeps into account the Quality of Service (QoS). In [12], the problem is faced by applying a joint approach of VNF consolidation and migration to use the less number of servers in each time interval; likewise authors in [13] propose to dynamically place VNFs in a distributed computing environment to obtain an efficient SFC.

C. TCAM size limitation

Flow tables are implemented by means of TCAMs, which are the fastest and most scalable packet classification solutions thanks to their parallel search capabilities. However, the high speed offered by TCAMs for classifying packets has several associated drawbacks, such as large power consumption and high cost [14]. Consequently, these two factors have an impact on the relatively small number of rules a TCAM is able to store. In particular, commercial switches only support a small fraction of the total number of rules required to operate networks (from 2k to 20k rules, as reported in [15]). Therefore, this limitation on the size of TCAMs is a challenging problem to be considered when proposing solutions on SDN-based networks.

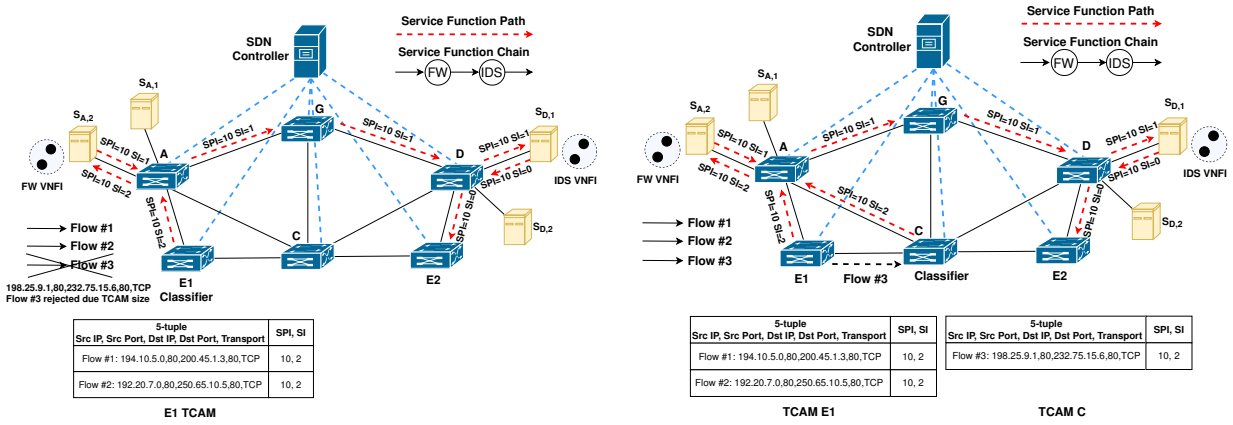
A large number of solutions are devoted to optimize the usage of the TCAMs in SDN switches [16], [17]. The aim of these proposals is to reduce, as much as possible, the memory space required to store a set of flow rules. This is achieved by re-writing the flow rules using a different semantic, trying to aggregate as many of them as possible. Here we want to highlight that the problem of shrinking the flow table of an SDN switch is different from the service chain request classification problem under TCAMs size constraint we aim to solve. In fact, while in the former the set of flow rules is considered to be an input, in the latter it is actually the output, which basically means that compression techniques can be seen as a post-processing of the flow tables obtained as a result of the SFC requests classification procedure.

The present work extends the model considered in [6], where SFC classification rules are proactively installed, by including the possibility to install/remove classification rules in a reactive way.

III. SFC Classification Offloading

IETF has standardized the architecture for SFC [1]. In this architecture, packets are classified at ingress nodes of the SFC-enabled domain by inserting, in the NSH, the required set of Service Functions (SFs). Next, each packet is forwarded through the set of SFs for processing. The network operator instantiates Service Function Paths (SFPs), i.e., constrained paths that allow to cross the SFs of a specific SFC. An SFP is characterized by a Service Path Identification (SPI). The SPI is inserted in the NSH together with the Service Index (SI); the SI is initialized to a value equal to the number of SFs to be executed and decreased by one in each VNF Instance (VNFI). An example of an IETF SFC-enabled network is shown in Fig. 1. The network is composed by 6 SDN switches and 4 servers, hosting the SFs instantiated by the VNFIs. As an example, let us consider that an SFC composed by a Firewall (FW) and an Intrusion Detection System (IDS) is available. The network provider then instantiates the SFP involving the switches $E1$, A, G, D, $E2$, and the servers $S_{A,2}$ and $S_{D,1}$ where FW and IDS VNFIs are respectively activated. The tuple (10,2) is assigned as SPI and SI to the SFP. Finally, notice how the classification of the traffic is performed by a classifier placed in switch $E1$. Packet flows are therefore assigned an SFP by

978-1-7281-4445-0/19/$31.00 ©2019 IEEE



(a) Traditional classification procedure.

(b) Proposed classification procedure.

Figure 1. Example of IETF SFC-enabled network in which traditional (a) and proposed (b) classification procedures are considered.

encapsulating them with an NSH reporting the corresponding SPI [2]. A network operator configures and dimensions the bandwidth allocated to the SFP according to traffic forecast [18].

As previously introduced, in this work we face the problem of TCAMs limited size at SFC classifiers. Even if the SFP bandwidth is sufficient to support all the packet flows, it could happen that not all of them could be classified due to the TCAM limited size of the SFC classifier. This situation is illustrated in Fig. 1(a), where three packet flows characterized by the 5-tuples (194.10.5.0, 80, 200.45.1.3, 80, TCP), (192.20.7.0, 80, 250.65.10.5, 80, TCP) and (198.25.9.1, 80, 232.75.15.6, 80, TCP) need a classification to be carried out on the SFP characterized by the SPI equal to 10. Unfortunately, in the case in which the TCAM memory size of the classifier is two entries, only two ((194.10.5.0, 80, 200.45.1.3, 80, TCP), (192.20.7.0, 80, 250.65.10.5, 80, TCP)) of the three flows can be supported, while the third one (198.25.9.1, 80, 232.75.15.6, 80, TCP) must be rejected.

To solve the problem, in [6] we have proposed a solution in which the classification is offloaded to internal switches for these packet flows that otherwise could not be supported. The only requirement on the node to offload the classification to, is that it has to belong to the Shortest Path between the Ingress Node (IN) and the Egress Node (EN) of the considered SFC request. This is because, before that the packets belonging to the SFC request are NSH-encapsulated, they are forwarded according to their destination IP address.

The solution is illustrated in Fig. 1(b), in which we show how the packet flow characterized by the 5-tuple (198.25.9.1, 80, 232.75.15.6, 80, TCP) is classified by switch C and the SFP is updated. The application of the solution requires that the network operator, according to the number of packet flows composing the offered traffic, re-configures the classifiers.Obviously, the use of internal switches for classification may lead to SFPs length increase and, consequently, to side effects such as bandwidth wasting.

IV. The Dynamic Chain Request Classification Offloading Problem

The considered scenario is the one of an Internet Service Provider (ISP) network, which is implemented using the SDN technology. The network topology is represented by means of a directed graph $\mathcal{G}(\mathcal{N}^{\text{SW}}, \mathcal{L})$, where $\mathcal{N}^{\text{SW}}$ is the set of SDN switches and $\mathcal{L}$ represents the set of links. Considering SDN node n_i^{SW}, its flow table maximum size is referred to as α_i, while the set of installed flow rules is represented as $\mathcal{F}_i$. Clearly, the condition $|\mathcal{F}_i| \leq \alpha_i$ must hold for each SDN switch. The available capacity of link l is expressed as b_l^{LINK}. The ISP offers SFC services by steering traffic flows through a set of VNFIs. VNFIs are instantiated at the Data Centers, which are connected to the core network by means of SFFs. The set $\mathcal{N}^{\text{VNF}}$ contains all the VNFIs defined in the considered scenario. The physical resources available at VNFI n_i^{VNF} are represented by the vector $\mathbf{b}_i^{\text{VNF}}$, whose j-th component expresses the amount of the physical resource j (e.g., CPU, storage, etc.) available. We assume that $\mathcal{K}$ different types of SFC services are available in the network, and for each type k of SFC there are I_k instances already configured. It implies that the flow rules needed to process NSH encapsulated packets are already installed in the flow tables of the SFFs. All the SFC instances are designed by using, for instance, the approach proposed in [18]. The j-th instance of SFC of type k is referred to as $\mathcal{I}_j^k$ and consists of all the involved VNFIs, SFFs, and network links. Moreover, $\mathcal{I}_j^k[1]$ refers to the first SFF of the SFC instance, while $\mathcal{I}_j^k[\text{LAST}]$ indicates the last one.

When a user requires an SFC service, it has to sign a Service Level Agreement (SLA) with the ISP. Such SLA contains the following information: i) the Ingress Node (IN) and the Egress Node (EN), ii) a flow classification rule, iii) the type of requested SFC, iv) the minimum requested throughput, v) the amount of required cloud resources, and vi) QoS parameters. An SFC request r is described by the tuple $(IN(r), EN(r), match_r, k(r), T_r^{\min}, \mathbf{Z}_r, \mathbf{q}_r)$, where $\mathbf{Z}_r$ is a vector describing the required amount of each cloud physical

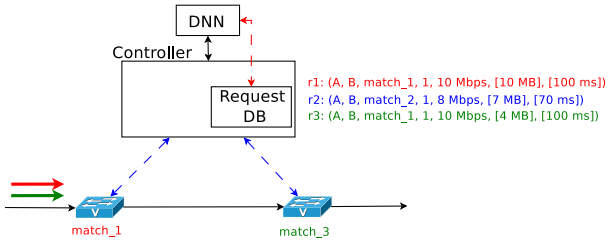
resource (storage, CPU, etc.), while $\mathbf{q}_r$ is a vector that specifies the QoS parameters requested by r (e.g., maximum delay, maximum path length, etc). The full set of SLA is referred to as $\mathcal{R}$ and it is stored at the controller

In the considered scenario, the ISP has to manage two different types of traffic flows, i.e., *SFC requesting traffic* and *normal IP flows*. While the firsts need to be steered through an SFC, the latter can be directly forwarded toward the proper egress node. IP-like flow rules are computed on the basis of a shortest path criterion (e.g., by using the Dijkstra algorithm). We refer to the path between nodes n_i^{SW} and n_j^{SW} as p_{ij}. Then, the SDN switches host two types of flow rules: i) *classification rules*, with a complex matching pattern (based on the value of many header fields) and requiring multiple actions to be performed (encapsulation and forwarding), and ii) *IP-like flow rules*, which are only focused on destination IP addresses and apply a single action (i.e., forward).

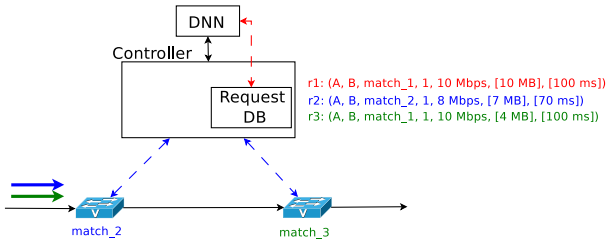
Due to the structure of the related flow entries, classification rules are in general, much higher in number than the IP-like ones. Consequently, the installation of classification rules in a proactive fashion would lead to a low number of served requests. This is a consequence of the limited size of the TCAMs of the SDN switches. On the other hand, due to the relative low number, IP-like flow rules can be proactively installed in the SDN switches. In [6] we have proposed the classification rule offloading mechanism as a method to increase the number of accepted requests, when these are handled in a proactive way. In the present work we propose to exploit the dynamic behaviour of the SFC requiring traffic, to further increase the number of served requests.

Specifically, considering a request r, this can be *active* or *idle* at any time. When the request is active, the stream of packets belonging to it is present in the network and needs to be served with the required SFC. Consequently, a classification rule must be present in the data path. On the other hand, when the request is idle, it means that currently there are not packets belonging to it in the network. For an idle request, there is no need to keep a classification rule installed in the data path. In Fig. 2 we propose a simple example to show how the active/idle behaviour of the requests can be exploited, in conjunction with the classification offloading mechanism, to increase the number of served requests.

Let us assume that each of the switches reported in the example of Fig. 2 can host a single flow rule. Globally, there are three different requests to be handled at the ISP. If the classical SFC architecture is considered, then classification rules must be pro-actively installed at the ingress node. As a consequence, only one, out of three requests, is served. Assuming that the classification offloading mechanism is exploited, it is possible to serve two out of three requests. For instance, in Fig. 2(a), classification rules for r_1 and r_3 are pro-actively installed at nodes A and B, respectively. Anyway, by exploiting also the dynamic behaviour of the requests, it is possible to serve all of them. In fact, when r_2 is idle, classification rules installed at the data path are needed only for requests r_1 and r_3 (Fig. 2(a)). As soon as the request r_1 becomes idle, the controller



(a) Request 2 is idle.



(b) Request 1 is idle.

Figure 2. Example of classification rules installed in a reactive way.

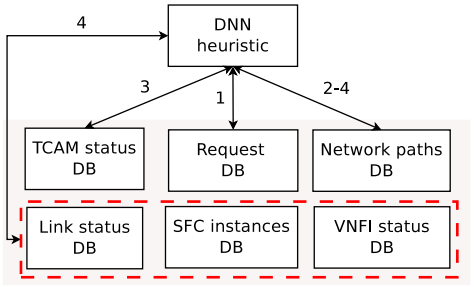


Figure 3. Architecture of the proposed DNN heuristic. This figure shows all the logical components of the proposed solution.

removes the related classification rule at node A, making room for the classification rule for request r_2 that now is active (Fig. 2(b)).

We define D-CRCO as the problem of maximizing, given a set of input requests, the number of served ones, having the possibility to both offload the classification of requests to transit nodes and to exploit the dynamic behaviour of the requests, while satisfying the QoS requirements and the capacity constraints (on links and VNFIs). In the next section we propose an heuristic algorithm that efficiently solves the D-CRCO problem.

V. The Dynamic Nearest Node Heuristic

In this section, the proposed heuristic to solve the D-CRCO problem is described. The main idea behind the Dynamic Nearest Node heuristic (DNN) is to exploit the concept of the nearest node, i.e., the classification rule for the current request is selected according to the distance with respect to the IN. In particular, if a request arriving to the SDN-based SFC domain needs to be classified, this process will be carried out by the node, among the ones belonging to the shortest path between the IN and the EN, which i) is closer to the IN; and ii) has available space in its TCAM to host the classification rule.

Fig. 3 shows the architecture of the DNN heuristic, with all the logical components (a set of data-bases available at the controller) needed to run it. The first logical component used in DNN is the set of SFC requests reaching the ISP network, $\mathcal{R}$. This information is stored in the *Request DB*, and it is known since SFC services must be negotiated in advance with the Service Provider (see Sec. IV). The current status of the flow table of each switch of the SDN-based SFC domain is stored in the *TCAM status DB* of the controller. OpenFlow protocol can be used to gather specific information, by periodically sending OFP_STAT_REQUEST messages. Using a similar approach, the controller can continuously monitor the status of the network links, which is stored in the *Link status DB*. Network paths are calculated by the controller by running the Dijkstra algorithm. The result of this computation is stored in the *Network paths DB*. The set of available SFC instances ($\mathcal{I}_j^k, \forall k \in \mathcal{K} \ \& \ j = 1 \dots I_k$) configured in the network, with all the related information, are available at the controller in the *SFC instances DB*. This DB is periodically updated by interacting with the Network Orchestrator, which computes the SFC instances. Finally, the available resources at each VNFI are stored in the *VNFI status DB*, that is built by interacting with the various Virtualized Infrastructure Managers (VIMs) that belong to the considered SDN-based SFC domain.

When a new request r arrives, DNN heuristic performs the following steps (represented with numbers in Fig. 3): i) it first inspects the *Request DB* in order to get all the required information, then ii) it searches the shortest path between $IN(r)$ and $EN(r)$ in the *Network paths DB*, after that iii) it determines the node to offload the classification rule to, by inspecting the *TCAM status DB* and by executing the *select_classifier* function, finally, iv) it looks for an SFC instance with enough resources (link bandwidth and cloud resources) to serve the current request, by collecting all the required inputs (link and VNFIs status, plus SFC instances paths) and passing them to the *steer_traffic* function. If all the steps can be successfully performed, then the classification rule for the current request can be installed in the identified node. Furthermore, the controller sets an *idle_timeout* for this rule in the SFC classifier. In this way, as soon as the stream of packets related to the current request stops, then it is considered to be idle and the classification rule is removed. Clearly, the action to decide the value of the *idle_timeout* is critical. If it is set too high, then the time needed to detect that the request is idle will be higher, with a consequent waste of TCAM space. On the other hand, if it is set too low, the request might be considered as idle, while it is actually active. As a consequence, the same rule will be re-installed many times, with a consequent increase of the signaling overhead and degradation of QoS experienced by the request.

In the next subsection we provide an insight of the main functions used in the DNN heuristic.

A. DNN Technical Details

The first function used in the DNN heuristic that we introduce is the *select_classifier*, whose pseudo code is reported in Alg. 1. Given an incoming SFC request, r, as input, this function returns the node in the shortest path between $IN(r)$ and $EN(r)$ (line 1) which i) is closer to the $IN(r)$; and ii) has available space in its TCAM to host the classification rule (line 3). For the ease of readability, we refer to the node selected as classifier for the current request r as n^r.

Algorithm 1 *select_classifier* function pseudo code.

Require: An incoming SFC request: r.
1: $p_{IN(r),EN(r)} \leftarrow$ *Network paths DB*
2: **for all** $n_i^{\text{SW}} \in p_{IN(r),EN(r)}$ **do**
3: **if** $|\mathcal{F}_i| \leq \alpha_i$ **then**
4: **return** n_i^{SW}
5: **end if**
6: **end for**
7: **return** $\emptyset$

Algorithm 2 *steer_traffic* pseudo code.

Require: Current request: r, the set of SFC instances of the required type: $\mathcal{I}_j^{k(r)}, j = 1 \dots I_{k(r)}$, the classifier node: n^r
1: $U^{\text{MAX}} = \infty$, $\mathcal{I}^r \leftarrow \emptyset$
2: **for all** $\mathcal{I}_j^{k(r)} \in \mathcal{I}^{k(r)}$ **do** ▷ Evaluate each SFC instance of type $k(r)$
3: $p^{\text{e2e}} = p_{\text{IN}(r),n^r} \cup p_{n^r,\mathcal{I}_j^{k(r)}[1]} \cup \mathcal{I}_j^{k(r)} \cup p_{\mathcal{I}_j^{k(r)}[\text{LAST}],\text{EN}(r)}$
4: $U_l \leftarrow$*check_link_constraints*$(p^{\text{e2e}}, r,$ *Link status DB*$)$
5: **if** $U_l < 100$ **then**
6: $U_{\text{VNF}} \leftarrow$*check_VNF_constraints*$(\mathcal{I}_j^k, r,$ *VNFI status DB*$)$
7: **if** $U_{\text{VNF}} < 100$ **then**
8: $\text{tmp} \leftarrow \min(U_l, U_{\text{VNF}})$
9: **if** $\text{tmp} < U^{\text{MAX}}$ **then**
10: $U^{\text{MAX}} \leftarrow \text{tmp}$
11: $\mathcal{I}^{k(r)} \leftarrow \mathcal{I}_j^{k(r)}$
12: **end if**
13: **end if**
14: **end if**
15: **end for**

The second function that is defined in the DNN heuristic is the *steer_traffic*, whose pseudo code is specified in Alg. 2. It is executed only if the *select_classifier* function has produced a non empty output. The main purpose of the *steer_traffic* function is to find a feasible SFC instance, of the required type, to serve the current request r. This task is accomplished in two main steps: i) evaluate the feasibility in terms of link bandwidth and QoS requirements, and ii) evaluate the feasibility in terms of cloud resources.

At first, the *steer_traffic* function (line 1) initializes two variables, named U^{MAX} and $\mathcal{I}^r$ respectively. The first one is used to store the maximum resource utilization achieved by the best SFC instance checked so far, while the latter is used to return the selected SFC instance as output. After that, it iterative inspects each of the available SFC instances of the requested type (line 2). In order to do that, it is needed to reconstruct the overall end to end path (p^{e2e}) followed by the request r in case it is assigned to the SFC instance under test (line 3). The end-to-end path is composed of, at most, four paths: i) the shortest path between ingress node of the request ($\text{IN}(r)$) and the classifier (n^r); ii) the shortest path between the classifier and the first SFF of the SFC instance

under test ($\mathcal{I}_j^{k(r)}[1]$); iii) the corresponding path defined by the SFC instance; and iv) the shortest path between the last SFF of the SFC instance ($\mathcal{I}_j^{k(r)}[\text{LAST}]$) and the egress node ($\text{EN}(r)$). In case the request r is classified at the ingress node, then the first path is empty.

Next, the feasibility of the overall end to end path is checked (line 4). Specifically, the function *check_link_constraints* is invoked, having as input the end to end path (p^{e2e}), the request (r) and the *Link status DB*, which contains all the information related to links (latency, residual bandwidth, etc). The *check_link_constraints* function verifies that the QoS requirements ($\mathbf{q}_r$) of the request r are satisfied. It also checks if all the links in the end to end path have enough residual capacity to steer the request ($T_r^{\min} \leq b_l^{\text{LINK}}, \forall l \in p^{\text{e2e}}$). If both the performed checks are successfully verified, then the *check_link_constraints* function returns as output the utilization of the most loaded link (U_l), otherwise it returns the value $U_l = 101$.

Similarly, in line 6, the capacity constraint on the VNFI is checked. This task is accomplished by the function *check_VNF_constraints*, which takes as input the SFC instance under test ($\mathcal{I}_j^{k(r)}$), the request r, and the residual capacity of the VNFI ($\mathbf{b}_i^{\text{VNF}}$), which is retrieved from the *VNFI status DB*. The *check_VNF_constraints* function verifies the validity of the following conditions: $\mathbf{Z}_r \leq \mathbf{b}_i^{\text{VNF}}, \forall i \in \mathcal{I}_j^{k(r)}$. Finally, it returns as output the utilization of the most loaded VNFI (U_{VNF}). In case the test fails, it returns $U_{\text{VNF}} = 101$.

Once all the SFC instances have been checked, the *steer_traffic* has to select one to be used to serve the current request. The criterion adopted to select the best SFC instance is the one with the lowest global utilization value. This selection process is implemented in lines $8-12$. The global utilization is defined as the minimum between the most loaded link in the path and the most loaded VNFI in the SFC instance (line 8).

If there is a feasible end-to-end path (and its associated SFC instance) to handle the request, the traffic demand of request r is steered through the selected path.

VI. Performance Evaluation

In this section, the potential benefits achieved by the proposed DNN heuristic, as well as possible drawbacks, are evaluated. To do that, a performance evaluation is carried out over a real network topology. In particular, four different analyses are performed to show i) the impact of the selection of the idle timeout; ii) the impact of the inter-arrival requests time; iii) the impact of the number of incoming requests to the system; and iv) the penalty that must be paid in terms of number of times a rule must be re-installed due to idle timeouts expiration.

Performance evaluation is carried out over NSFNET network, which is composed of 14 SDN switches and 46 links [18], and is reported in Fig. 4. It is assumed that all the SDN switches can host the same number of rules, i.e., they have the same TCAM size. Since real TCAMs are able to store a range between 2000 and 4000 rules [16], TCAM size is set to 2000

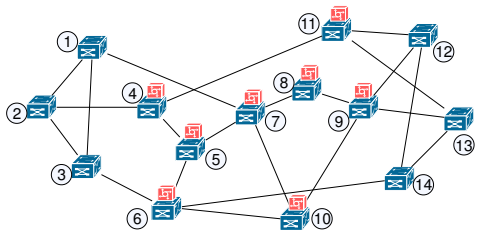


Figure 4. NSFNET topology with 14 SDN switches and 8 Cloud Infrastructures.

rules in the simulations. Four SFs are considered: Firewall (FW), Intrusion Detection System (IDS), Network Address Translator (NAT) and Proxy. With this set of SFs, four types of SFCs are used: 1) FW; 2) FW→IDS; 3) FW→IDS→NAT; 4) FW→IDS→NAT→Proxy. Finally, 20 SFC instances are deployed in the network, using the approach described in [18].

Concerning the creation of the set of requests, the following procedure has been used: i) at first, an IE TM (aggregated flow) is generated, in which each entry is characterized by a bandwidth value chosen in the set [10Gbps, 15Gbps, 20Gbps, 25Gbps, 30Gbps] according to a Zipf distribution [18], then ii) each aggregated flow is divided into m micro-flows of the same size (in terms of requested bandwidth).

DNN, S-CRCO [6] and *alwaysIN* are coded in Matlab and run on a dual-core Intel-based machine at 3.1 GHz with 16 GB of RAM. All the analyses are repeated 25 times and the average values are considered.

The first analysis we propose aims at evaluating the impact of the rules' idle timeout on the number of accepted requests. For this analysis, a set of 50000 SFC requests is used. Moreover, the duration of each request is set according to an exponential distribution with average time of $\mu = 5s$. for all the performed analyses. Finally, inter-arrival time between requests is set to $\delta = 1s$. Fig. 5 reports the results obtained by the *alwaysIN* approach (in which the classification procedure is always performed at the IN), by the S-CRCO (classification rules are installed proactively), and by the proposed DNN heuristic as a function of the idle timeout. For the DNN solution, an incoming request is accepted only if it is classified every time it becomes active. Clearly, the dynamic behaviour exploited by DNN obtains much better results compared to *alwaysIN* and S-CRCO, especially for low values of idle timeout. The fact of removing rules after an idle timeout allows to handle and classify more incoming requests. In case the idle timeout is highly increased, the number of accepted requests is decreased. This is due to the fact that rules remain much more time installed in the flow tables of the nodes even though the last matching packet occurred long time ago and there is less space to accommodate new classification rules. Anyway, in the worst-case scenario (idle timeout equal to $10^5 s.$), DNN is able to classify 1000 requests more than the *alwaysIN*, achieving similar results compared to S-CRCO.

In the second analysis, the objective is to analyze the impact of the inter-arrival requests time on the number of accepted requests. For this analysis, inter-arrival times for

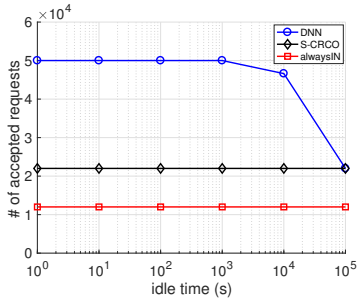


Figure 5. Accepted requests as a function of the idle time with inter-arrival time equal to $\delta = 1s$.

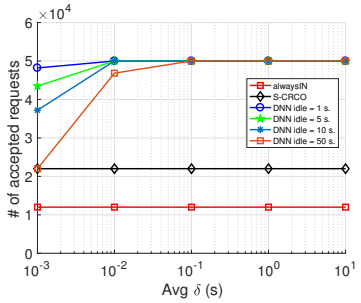


Figure 6. Accepted requests as a function of the average inter-arrival time for different values of idle times.

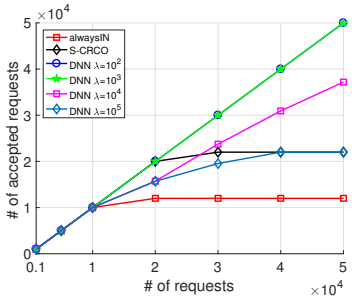


Figure 7. Accepted requests as a function of the number of incoming requests for different values of λ.

the set of requests follow a Poisson distribution with average δs. Looking at Fig. 6, DNN heuristic presents again better results compared with *alwaysIN* and S-CRCO by exploiting the removal of rules after an idle timeout expiration. If we only focus on the different lines reported for the DNN, in which different idle timeouts are considered, it can be seen that they tend to follow a logarithmic function, in which the number of accepted requests increases with δ. It is clear again, as in the analysis reported in Fig. 5, that the idle timeout has a strong impact on the number of requests that can be classified, being high for small values of idle timeout and vice-versa (see, e.g., the values reported for $\delta = 10^{-3}s$.).

Next, an analysis to show the number of accepted requests as a function of the number of incoming ones is shown in Fig. 7. In this case, we define term λ as the ratio between the idle timeout for each request and the average inter-arrival time for the set of requests:

$$\lambda = \frac{idle\_timeout}{\bar{\delta}} \quad (1)$$

By inspecting Fig. 7, it can be seen that all the incoming requests are accepted (and indeed classified) by DNN for values of $\lambda \leq 10^3$. This means that if the difference between the *idle_timeout* and the average inter-arrival time of requests, $\bar{\delta}$, is not quite big, rules are continuously been removed from the flow tables of the nodes and there is free space for installing new classification rules. On the contrary, for values of $\lambda \geq 10^4$, the number of accepted requests decreases up to reaching a minimum of 22000 in the worst-case scenario. In this way, it is necessary that the idle timeout is big enough compared to the average inter-arrival time to not be able to classify, and therefore reject some of the incoming requests. Remarkably, the worst solution obtained by the proposed DNN approach overcomes *alwaysIN*, since it is limited to only classify at the ingress node. It is also interesting to notice that S-CRCO presents better results than DNN when the number of incoming requests belong to the range $[20000, 30000]$. This is caused by the timing component introduced in DNN, which is not considered in the static version.

The last analysis we propose aims at analyzing the impact of the DNN heuristic on the number of times a classification rule must be re-installed due to the expiration of the idle timeout. The setting of the simulation is described as follows. At first, the set of 50000 requests used in the last analysis (see Fig. 7) is selected. Next, we assume the case where all the requests are accepted by exploiting, e.g., DNN with a value of $\lambda = 10^2$ or $\lambda = 10^3$. Then, for each accepted request, we generate a flow pattern following the Variable bitrate (VBR) traffic pattern. In this way, a flow is composed of a set of sending periods (in which the traffic is sent at a constant bitrate) followed by silent periods (in which no traffic is sent). For the flows composition, next considerations are taken into account: i) the number of silent periods is selected using a probabilistic distribution in the range $[1, 5]$; ii) the percentage of aggregated silent time over the total flow duration is selected using a random distribution in the range $[0, 100]$ [%]; iii) the duration of each silent period is then randomly selected respecting the aggregated maximum limit obtained in ii).

Five values for idle timeout are considered, in the range $[1, 5] \in \mathbb{N}$. Results of this last analysis are shown in Fig. 8, in which the number of times a classification rule is installed for the same flow throughout time is reported. Since the dataset used contains 50000 requests and tests have been repeated 25 times, average values are considered for each case. By inspecting Fig. 8, it is clear to remark that DNN has to pay a penalty in terms of the re-installation of rules that are removed due to idle timeout expiration, unlike S-CRCO, where each classification rule is installed once at most. Fortunately, such penalty is not extremely restraining, resulting on average that a rule must be installed 1.6 times in the worst-case scenario, i.e., when the idle timeout is low ($1s$.). Clearly, the penalty decreases with the increase of the idle timeout, since the probability that rules are removed due to timeout expiration is reduced. This situation can be better explored by looking at Fig. 9, in which the number of times it is required to install the same classification rule is reported for each individual incoming request, considering idle timeouts of $1s$. (Fig. 9(a)), and $5s$. (Fig. 9(b)). The smaller the idle time is, the higher the number of times the rule must be re-installed.

VII. Conclusion and Future Works

In this paper, the problem of TCAM size limitation at the SFC classifier has been studied. In particular, considering an SDN-based SFC domain, the main logical functions of the SFC architecture can be mapped in SDN switches. While the possibility to control the network in a centralized way

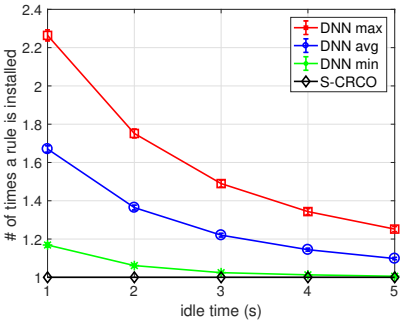
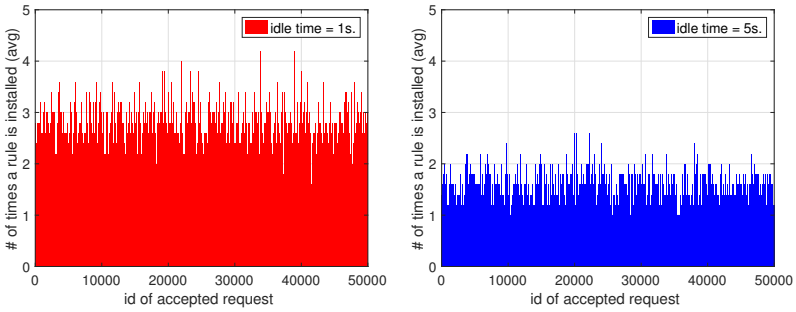


Figure 8. Number of times a classification rule is installed as a function of the idle time.



(a) Idle time = 1s. (b) Idle time = 5s.

Figure 9. Average number of times each classification rule is installed.

in conjunction with the flexibility offered by the generalized packet forwarding, brings SDN into the light as an enabling technology for implementing SFC underlay, the limited size of the TCAM memories turns to be a bottleneck for the number of SFC requests that can be handled by the classifier.

To overcome this hard constraint, the Dynamic Nearest Node heuristic is introduced. DNN exploits the classification offloading mechanism, i.e., it aims to install classification rules also into nodes that are internal to the SDN-based SFC domain. Moreover, DNN also exploits the dynamic behaviour of SFC request flows, to further increase the number of accepted requests. The performance evaluation shows that DNN can increase the number of accepted requests of 354% with respect to the classical *alwaysIN* approach, and of 127% with respect to the static CRCO, which installs classification rules in a proactive way.

As future steps, there is the formalization of the Dynamic Chain Request Classification Offloading problem by means of an Integer Linear Programming formulation, so that to provide an indication of the maximum benefit of the classification offloading mechanism as well as of the reactive classification rule installation. Furthermore, as a future step there is the realization of a working test-bed to validate the feasibility of the proposed approach.

Acknowledgements

This work has been partially funded by the 4IE Project (0045-4IE-4-P) and 4IE+ Project (0499-4IE-PLUS-4-E) funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 program, by the Spanish Ministry of Science, Innovation and Universities (RTI2018-094591-B-I00), by the Department of Economy and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 7665, October 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7665.txt

[2] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," RFC 8300, January 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8300.txt

[3] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *IEEE INFOCOM 2014-IEEE conference on computer communications*. IEEE, 2014, pp. 1734–1742.

[4] J. Galán-Jiménez, M. Polverini, and A. Cianfrani, "Reducing the reconfiguration cost of flow tables in energy-efficient software-defined networks," *Computer Communications*, vol. 128, pp. 95 – 105, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366417312550

[5] M. Polverini, A. Baiocchi, A. Cianfrani, A. Iacovazzi, and M. Listanti, "The power of sdn to improve the estimation of the isp traffic matrix through the flow spread concept," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 6, pp. 1904–1913, 2016.

[6] M. Polverini, J. Galán-Jiménez, F. G. Lavacca, A. Cianfrani, and V. Eramo, "Proposal and investigation of a scalable and offloading-based traffic classification solution in nfv/sdn network architectures," *Under Review at IEEE Access*, 2019.

[7] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi, "Traffic steering for service function chaining," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 487–507, 2018.

[8] W. Ding, W. Qi, J. Wang, and B. Chen, "Openscaas: an open service chain as a service platform toward the integration of sdn and nfv," *IEEE Network*, vol. 29, no. 3, pp. 30–35, 2015.

[9] I. Trajkovska, M.-A. Kourtis, C. Sakkas, D. Baudinot, J. Silva, P. Harsh, G. Xylouris, T. M. Bohnert, and H. Koumaras, "Sdn-based service function chaining mechanism and service prototype implementation in nfv scenario," *Computer Standards & Interfaces*, vol. 54, pp. 247–265, 2017.

[10] A. Varasteh, M. De Andrade, C. M. Machuca, L. Wosinska, and W. Kellerer, "Power-aware virtual network function placement and routing using an abstraction technique," in *2018 IEEE Global Communications Conference (GLOBECOM)*, December 2018, pp. 1–7.

[11] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and QoS-aware path allocation and VNF placement for service function chaining," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, March 2019.

[12] V. Eramo, M. Ammar, and F. G. Lavacca, "Migration energy aware reconfigurations of virtual network function instances in NFV architectures," *IEEE Access*, vol. 5, pp. 4927–4938, 2017.

[13] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[14] C. R. Meiners, A. X. Liu, and E. Torng, "Bit weaving: A non-prefix approach to compressing packet classifiers in tcams," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 488–500, April 2012.

[15] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable ethernet for data centers," in *2012 International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, December 2012, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413183

[16] E. Norige, A. X. Liu, and E. Torng, "A ternary unification framework for optimizing TCAM-based packet classification systems," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 657–670, April 2018.

[17] A. X. Liu, C. R. Meiners, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 2, pp. 490–500, 2010.

[18] V. Eramo and F. G. Lavacca, "Computing and bandwidth resource allocation in multi-provider NFV environment," *IEEE Communications Letters*, vol. 22, no. 10, pp. 2060–2063, October 2018.