Received: 28 June 2016 | Revised: 30 September 2016 | Accepted: 8 December 2016

DOI 10.1002/nem.1965

WILEY

RESEARCH ARTICLE

# CATENAE: A scalable service function chaining system for legacy mobile networks

**Roberto Bifulco** | **Anton Matsiuk** | **Alessio Silvestro**

NEC Laboratories Europe, Heidelberg, 69115, Germany

**Correspondence**
Roberto Bifulco, NEC Laboratories Europe, Heidelberg 69115, Germany.
Email: roberto.bifulco@neclab.eu

**Funding information**
EU in the context of the "FP7 ITN CleanSky" project, Grant/Award Number: 607584

**Summary**

Service function chaining promises to ease the introduction of new services in mobile networks by enabling the dynamic composition of virtualized network functions. However, current implementations rely on new tunneling protocols and on network infrastructure changes, which make deployments in legacy networks difficult. In this paper, we present a system for service function chaining that can be readily deployed in mobile networks, without requiring any protocols or network modifications. Our system, named CATENAE, exploits the specific properties of the targeted deployment environment to overcome scalability issues and efficiently implement traffic steering. Using a centralized controller, CATENAE coordinates software-defined networking software switches to implement a steering method based on MAC address rewriting. Furthermore, we present the design of a hybrid hardware-software software-defined networking switch to realize a scalable network traffic classifiers, which CATENAE employs to assign network flows to corresponding function chains. Using experimental evaluations based on physical testbeds, emulated environments, and simulations, we demonstrate that our approach is ready to be deployed in mobile networks, scales to handle expected mobile networks' workloads, introduces no overhead in the (virtual) network functions, and integrates seamlessly with legacy network management systems.

**KEYWORDS**

management approaches: policy-based management, methods: design, methods: experimental approach, network management: IP networks, network management: LANs, service management: hosting (virtual machines)

## 1 | INTRODUCTION

Network operators deploy network functions to enforce their policies and to provide additional services on top of plain connectivity.[2] Content caching, Network Address Translation (NAT), Transmission Control Protocol (TCP) video transcoding, and HTTP header enrichment are examples of such services. Despite their ubiquitous usage,[3] network functions deployment is still performed by modifying the network topology. That is, network functions are hard-wired on the network traffic's path. The inflexibility and complexity of this approach is not acceptable when network functions are implemented by means of software running in virtual machines, as envisioned in the case of network function virtualization.[4] In fact, hard-wiring would hinder the benefits brought by the possibility of dynamically deploying virtual network functions (VNFs) on general purpose servers. Therefore, there is a growing interest on service function chaining (SFC) systems,[5] which enable the flexible deployment of network functions while guaranteeing their configurable and dynamic chaining.

In general, a SFC system assigns a network flow entering the managed network to a chain of functions and steers

*An earlier version of this work was presented in this 1 study.[1] This new version includes 2 relevant extensions. First, we present a case for deploying an end-of-chain classifier, which generalizes the system to work also in deployments that do not use NATs. Second, we describe a technology to scale the classifiers using commodity hardware SDN switches. This included the design of offloading algorithms for such hybrid hardware-software architectures and their evaluation.

 Copyright © 2017 John Wiley & Sons, Ltd. 

the flow through the functions of such chain, according to the chain's functions ordering.[6] A number of challenges arise when addressing the design of a SFC system. First, assigning a network flow to its chain requires network traffic classification, an operation that is critical for the system scalability since it should be performed for all the handled traffic. Second, traffic forwarding should be performed according to the chain the traffic belongs to, instead of following the typical forwarding approach, e.g., based on IP routing. Third, network flows are usually bidirectional, that is, there is an upstream and a downstream direction and a network function, e.g., a firewall, may need to handle both of them. This requires to perform a coordinated classification of upstream and downstream flows and the enforcement of symmetric paths for the 2 directions. Finally, network functions may have dynamic and opaque behaviors that modify the network traffic in unknown ways, which may introduce a need for traffic reclassification or even make the traffic unclassifiable.[7]

To address these challenges, a number of SFC systems have been already proposed[7–11]. However, they usually target green-field or long-term deployments. In fact, they require a number of changes either in the network hardware[7] or in the network functions,[11] or in both.[8] In other cases, they require modifications to the network architecture.[9] Ready to deploy solutions, which do not require such changes, may instead not handle all the aforementioned challenges. For example, some SFC systems are unable to deal with opaque network functions actions.[7,10] Regardless of the adopted solutions, the proposed systems address SFC in a general way, supporting a broad range of deployment scenarios without considering their specific properties and constraints. That is, they usually adopt a "one-size-fits-all" approach. While we recognize the intrinsic value of such a general solution, we also notice that not all the deployment scenarios share the same set of requirements, with the final result of SFC systems that provide unnecessary features for the specific scenarios in which they are deployed. At the same time, such systems usually fail to satisfy a critical requirement of many today's production deployments, i.e., the SFC solution should introduce a minimum impact on the legacy infrastructures.[12,13]

In this paper, we argue that it is possible to simplify the implementation of a SFC system, by carefully tailoring the SFC solution to its specific deployment scenario. Our main contribution is to demonstrate that this statement holds true for the practical case of implementing SFC in mobile networks. To this aim, we present the design and implementation of CATENAE[†], a system that supports SFC in today's mobile networks without introducing new protocols, without changing the legacy infrastructure, and without changing network functions behavior. CATENAE leverages the unique properties of a mobile network's scenario to provide the desired functions chaining features, including the handling of opaque network functions' actions. Traffic forwarding is performed by rewriting network packets' header to steer network flows from one function to the next one in the chain. Rewriting rules are configured using software-defined networking (SDN) software switches, which are anyway deployed at the servers hosting VNFs.[14] Flow reclassification after a VNFs is done by creating per-VNF VLAN topologies, using an approach conceptually similar to this 1 study.[15] By implementing a proof of concept prototype, we demonstrate that CATENAE does not add perpacket processing overheads; it integrates nicely with legacy network management systems and it is fully compatible with legacy network infrastructures, and functions while supporting millions of network flows.

**Organization**. The paper is structured as follows:

- Section 2 introduces background information on the mobile networks where CATENAE is deployed and introduces related work;
- Section 3 presents the CATENAE's design, describing its architecture and deployment options;
- Section 4 describes the design of the classifiers employed to assign network traffic to service function chains;
- Section 5 presents the CATENAE's traffic steering method and provides a concrete example of the method applied to a network scenario;
- Section 6 reports the results of our prototype evaluation;
- Section 7 discusses our design in the light of the evaluation results and further describes differences with related work.

Finally, we conclude our paper in Section 8.

## 2 | BACKGROUND AND RELATED WORK

This section presents relevant background information about the mobile networks in which CATENAE can be deployed, introduces the current work on SFC performed by standard organizations like Internet engineering task force (IETF), and provides an overview of the SFC solutions proposed by the research community.

### 2.1 | Mobile networks

Our work is focused on implementing SFC in long-term evolution (LTE) cellular networks (cf Figure 1). A LTE network gives connectivity to a user equipment (UE) using a radio network provided by a set of eNode-Bs (eNBs), which are deployed by the operator over a geographic area. The eNBs encapsulate UE's network flows in a tunnel that, traversing the serving gateway (SGW), brings user's IP packets to the packet data network gateway (PGW). The PGW is the UE's gateway towards IP networks, i.e., all the IP traffic coming and going to the operator's IP network (and to the Internet) goes through the PGW. Also, the PGW is the point where the UE's IP address actually exists in the network. The policy

[†]Catenae is a Latin word that means "chains."

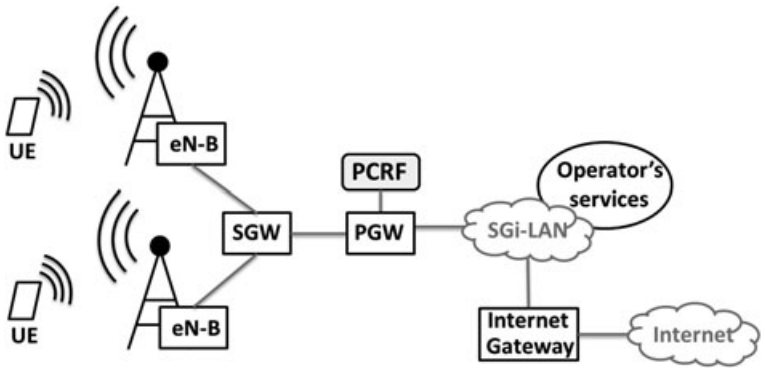


**FIGURE 1** Long-term evolution network architecture

and charging rules function provides the PGW with the policies to handle users' traffic, e.g., it provides the quality of service configuration. After the PGW, user's packets are sent to the SGi-LAN, which is the place where the operator provides additional services.[16] The SGi-LAN is usually an Ethernet network, where network functions are deployed and wired together either physically or logically (e.g., defining VLANs). Network functions can be either transparent, i.e., they do not modify packets' header, or opaque, i.e., they modify packets' header. After the packets have been processed by the various functions, they are finally delivered to an Internet gateway (IGW) that forwards them to the Internet.

We highlight a few points about LTE networks, which will help in understanding the design decisions presented in Section 3. First, operators plan to replace legacy network functions with virtualized ones, by deploying, in the SGi-LAN, a relatively small number of servers (e.g., less than a hundred) that will host VNFs. Thus, we expect an SFC system will deal with VNFs in the number of thousands and that these VNFs are connected to each other by an L2 network, since the SGi-LAN is usually a traditional Ethernet network. Second, the network traffic exposes properties that are typical of LTE deployments. That is, the upstream flows (ie, those generated at the users) are usually much smaller in size than the downstream flows.[17] Also, the connections are (almost) always initiated in the upstream direction.

### 2.2 | Service function chaining in standards

The IETF is the main standard organization that is dealing with SFC, stating the SFC problem in RFC[18] and defining the architecture of an SFC system in RFC.[19] In the IETF architecture, a network function is relabeled service function (SF). Thus, an SFC is an abstract definition of an ordered set of SFs.

The incoming traffic, e.g., in the upstream direction, at the edge of an SFC-enabled domain, is classified by a service classification function, to perform traffic steering through the correspondent chain. The service classification function adds an SFC-encapsulation to the classified packets. Notice that the architecture defines the encapsulation format as independent from the network encapsulation protocol used to interconnect the elements. This way, the SFC system does not necessarily need a homogeneous network between the chain's functions and can instead support more complex scenarios that enable service providers to use different technologies. The SFC-encapsulation is used by another component of the architecture: the service function forwarder (SFF). SFFs read the SFC-encapsulation to send network packets to directly attached SFs or to forward them to the SFF to which the next function in the chain is attached. For instance, a network switch may host an SFFs function if extended to read the SFC-encapsulation format. Because the RFC7665's architecture assumes that SFs can deal with the SFC-encapsulation format, SFC-unaware functions (e.g., legacy network functions) are supported by the usage of an SFC-proxy. A SFC-proxy removes the SFC-encapsulation at the ingress of a SFC-unaware area and adds it again on the egress of that area. An end-of-chain classifier has the responsibility to remove the SFC-encapsulation when packets exit the SFC-enabled domain and to classify the packets belonging to the downstream traffic.

**Network Service Header**. While there are no standards defined for the SFC-encapsulation format, a currently discussed proposal is the Network Service Header (NSH). The NSH is composed of a Base Header (32 bits), a Service Path Header (32 bits), and zero or more Context Headers. The Base Header provides information about the Service Path Header and the payload protocol. The Service Path Header is composed by a Service Path ID to identify the chains and a Service Index to provide location within the chain. Context Headers carry opaque metadata and variable length encoded information. The NSH header is located between the original packet/frame and the overlay network encapsulation protocol, if any. In fact, current NSH-based prototypes usually assume that an overlay network, e.g., based on VxLAN, connects SFFs. The original data unit, e.g., an L2 frame or an L3 packet, thus, is encapsulated within different transport protocols such as VLAN, VxLAN, Generic Routing Encapsulation Ethernet, etc. When an SF receives a packet coming from a service chain, it will decrement the service index header to update the location of the packet within the chain. At the end of the chain,

an end-of-chain classifier will remove the NSH header and forward the packet normally. NSH is transport independent because it can be used with different encapsulation protocols. It provides information about the chain each packet belongs to, through the Service Path ID header, and the location within the chain, through the Service Index Header. Context Headers make possible to share network and service metadata (L2-L7) that enable to reclassify the packets after an SF.

### 2.3 | Service Function Chaining in research

A number of proposals have been presented by the research community, to address the challenges of SFC.

SIMPLE[7] provides SFC using a SDN network. It implements inter-switch tunnels to aggregate the traffic with common destinations, to reduce the total number of forwarding rules in the SDN switches' forwarding tables. When such optimization is not required, hop-by-hop fine granular forwarding rules are used instead. Traffic reclassification, after an opaque network function, is performed using a dynamic module, which analyzes the similarities between packets entering and exiting the network function. However, such solution shows limited accuracy, and it introduces significant delays in the network flows.

To overcome such limitations, FlowTags[8] suggests the modification of the network functions to provide contextual information, in the form of a tag, which is added to the processed network packets, to perform traffic classification. The tags are defined by a centralized controller and cached at the network functions, using an approach similar to the handling of network packets at the controller in OpenFlow networks.[20] Like in SIMPLE, packets forwarding is performed writing appropriate forwarding rules in the SDN switches along the path.

Using a SDN network to perform traffic steering is the solution adopted also by StEERING.[10] In this case, the authors leverage a smart encoding of the forwarding rules in a multi-table switch's pipeline, to scale the total number of supported chains and network flows, still providing fine-grained traffic steering. However, StEERING is not able to reclassify the traffic in the presence of opaque network functions.

Finally, SoftCell[21] presents a solution that takes into account the deployment scenario's properties to simplify the implementation of SFC in mobile networks. To the best of our knowledge and putting aside CATENAE, it is the only proposal that explores such an approach. To be deployed, SoftCell requires a network of SDN switches and a modification of the mobile network's architecture. For instance, SoftCell removes serving gateway and PGW functions and therefore removes LTE's mobility management introducing a custom solution instead. Traffic classification is performed at switches colocated with the eNBs for the upstream direction, while classification for downstream traffic is performed leveraging information encoded in the source IP address/transport port of outgoing packets. In fact, traffic is assumed to be always initiated in the upstream direction; thus, any downstream packet will carry in the destination IP address/transport port the original upstream flow's encoded value.

## 3 | DESIGN

This section presents our design choices, the CATENAE's architecture, and provides an overview on possible deployment options in LTE infrastructures.

The main objective of CATENAE's design is to provide SFC while minimizing the impact on current infrastructures. To this aim, our design decisions are taken in the light of the properties characterizing the deployment scenario, ie, the LTE network. We make a few observations that motivate our design decisions. First, the main and most important observation is that network functions are connected using an Ethernet network, while user traffic is composed of IP packets, because the tunnel that brings the traffic from eNBs to the PGW only transports IP packets. Thus, the user traffic is agnostic to the L2 packets header, and therefore, we can manipulate the L2 header to perform traffic forwarding according to our needs. Second, the upstream flow is always started before the downstream flow, and upstream traffic's throughput is usually orders of magnitude smaller than downstream one. Because of these 2 observations, we can perform traffic classification in the upstream direction using a software classifier. In fact, while the classifier is traversed by all the user traffic, it could be still able to scale to handle millions of flows, if these flows contribute a relatively small aggregated throughput.

The remainder of this section presents the way we capture these observations in the designed architecture and in the corresponding traffic steering method.

### 3.1 | Architecture

CATENAE's architecture (cf Figure 2) is composed of 4 elements: the classifiers, which perform traffic classification on the packets entering the SGi-LAN; the VNFs' switches, which are deployed at the servers and connect VNFs with the SGi-LAN; the SGi-LAN itself, i.e., an Ethernet network that connects classifiers and VNFs' switches with each other; and the SFC controller that configures classifier and VNFs' switches in a coordinated way to enforce function chains. Both the classifiers and the VNF's switches are SDN switches (e.g., they implement OpenFlow), while the SGi-LAN implements a typical MAC learning algorithm. Thus, the SFC controller does not change the SGi-LAN network's operations but uses it as a mere transport network between VNFs located on the servers.

The SFC controller offers a function chains configuration interface, which could be connected to e.g., the policy and charging rules function of the LTE architecture. Upon reception of a chain installation request, the SFC controller

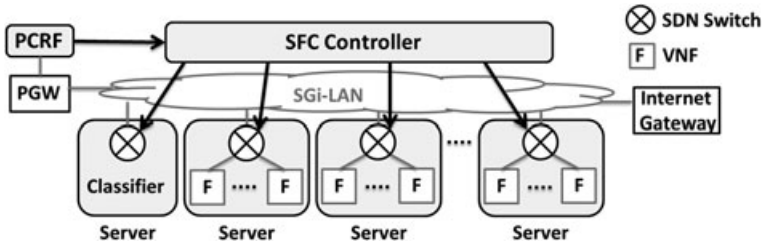


**FIGURE 2** CATENAE's architecture

implements the chain by installing forwarding entries at all the involved switches. Function chains are described by a list of flow identifiers (FIDs) and a list of functions. Each FID includes one or more of the following fields: IP addresses, transport ports, and the IP header's DSCP field. Also, while the FID always defines the upstream direction of a flow, it also identifies the downstream direction as well. In fact, the downstream direction of a flow can be identified by switching source IP addresses and transport ports values with destination ones. The functions' list contains the chain of network functions for the flow identified by the FIDs, specified in the order in which the upstream flow should traverse them. The last function in the list is the chain's exit point, e.g., a NAT. Each network function is further described by a network location. Network locations can be both provided with a static configuration, or the SFC system can perform a lookup for the location using a different interface, e.g., connected to a Virtual Machines (VMs) management system.

CATENAE supports 2 different deployment models, depending on where classifiers are deployed. In particular, the most general configuration assumes the presence of both an upstream classifier (u-classifier) and a downstream classifier (d-classifier). However, CATENAE can also use just the u-classifier, by making a stronger assumption on the deployment scenario, i.e., assuming the deployment of NATs as last chains' function. The next subsections describe the 2 deployment options.

### 3.2 | Two-classifier configuration

In a 2-classifier configuration, CATENAE uses a classifier per network traffic direction, i.e., upstream and downstream. Please notice that while conceptually separated, the u-classifier and d-classifier can be implemented by one single device. There are 2 important issues to be addressed with this configuration: first, the classifiers have to handle the entire system's network traffic, which impacts the overall system scalability; second, the downstream classifier needs to dynamically learn packet headers for flows that have been processed by opaque network functions. In Section 4, we further elaborate on these issues.

### 3.3 | Single classifier configuration

In the majority of mobile network deployments, there is always a NAT function used in a chain,[3] since private IP addresses are usually in use on the UE-side.[22] In such cases, CATENAE can perform traffic classification in the downstream direction without adding a dedicated d-classifier. This is accomplished mandating the deployment of NAT functions as the last chain's functions, which is anyway already a common practice. A NAT function performs a mapping between upstream traffic and corresponding downstream traffic, to apply address translation. When an upstream's packet traverses the NAT, its source IP address is rewritten with a NAT's routable IP address. Thus, any corresponding downstream traffic's packet will be delivered to the NAT, having the destination IP address set to the NAT's routable IP address. Because the NAT function is first hit by the already classified upstream traffic, the NAT will associate any downstream traffic to its upstream flow.[21] In effect, the NAT is providing both address translation and traffic classification, removing the need to deploy a dedicated classifier. Furthermore, as NATs are virtual network functions, they can be scaled to match the workload experienced by the system, following the same procedures used for any other network function.

### 3.4 | Deployment

CATENAE can be deployed in legacy SGi-LANs. The deployment process requires the configuration of SDN software switches in the servers connected to the SGi-LAN, the deployment of the SFC controller, and the redirection of the user traffic to the CATENAE's classifiers. While the former activities are a matter of software configuration on the servers, the redirection of the traffic to the classifier is the actual hook of the SFC system in the SGi-LAN. Such operation is as easy as changing the default IP gateway address in the PGW's configuration. In fact, the classifier is implemented as a software switch running on a general purpose server connected to the SGi-LAN. Of course, if a d-classifier is also used, the CATENAE deployment involves also the installation of the hardware SDN switch used to implement the classifier.

## 4 | CLASSIFIERS

In this section, we present the design of the classifiers employed in CATENAE.

In general, a classifier is configured with rules that define the chain a network packet belongs to. In the most common case, a rule specifies a set of packet headers, i.e., a network flow, and the corresponding classification action. In CATENAE, the classification action is as simple as forwarding the packet to the first function in the chain for the upstream flows, or to the last one for the downstream ones (for the downstream direction, the last chain's function is in fact the first function in such direction). CATENAE uses 2 types of classifiers, depending on the direction of the flows being classified.

The u-classifier only performs classification for packets entering the SGi-LAN in the upstream direction; in fact, CATENAE enforces symmetric paths for upstream and downstream flows, with respect to the network functions, but only upstream flows are processed by the u-classifier. In our architecture, this classifier is implemented using a SDN software switch. Having a software switch handling all the traffic coming from the PGW may raise scalability concerns; however, upstream flows are contributing just a fraction of the overall load (cf Section 6). On the other side, a software switch guarantees very large forwarding tables, i.e., one could use a very large number of rules to classify the network traffic. In particular, a software switch typically uses various solutions that involve hash tables that leverage general purpose servers' memory hierarchy to achieve high throughput. This finally guarantees the possibility of using cheap DRAM to host most of the classification rules, while a much faster cache, hosted in the CPU's SRAM, provides high throughput for the subset of highly used rules.[23]

The second type of classifier is the d-classifier. In this case, the classifier has to face a much more pressing scalability problem, since downstream traffic is usually 10 times bigger in volume than upstream one (cf Section 6). A software switch may not scale to meet the system throughput requirements; thus, a hardware classifier may be needed instead.

While one could rely on ad-hoc hardware for such purpose, we preferred maintaining a consistent architecture and implement the classifier with a commodity hardware SDN switch. The main advantage of such a decision is that the classifier interface is always the same for both the software and hardware components, i.e., OpenFlow. However, current hardware SDN technology can only offer limited space to host the classification rules.[24] Also, a strategy that capitalizes on the table space, by installing entries only when a chain's flow is actually active,[8,25] is viable only in few cases. In fact, hardware switches could not support scenarios that require high entries installation rates, being typically too slow at installing new entries.[26,27]

To tackle this issue, we developed HSwitch, taking inspiration from previous work that combines a hardware switch with a software switch to extend the switch's forwarding table size.[28] The remainder of this section describes our implementation of the d-classifier.

### 4.1 | d-classifier design

Our d-classifier design[27] connects a hardware SDN switch with a server that runs a software SDN switch.[‡] A packet that does not match a hardware switch's forwarding entry is sent to the software switch, where the SFC controller installs all the forwarding entries. The software switch is extended to implement a logic that offloads entries to the hardware switch, depending on the network load. In effect, the hardware switch is used as a microflow cache,[23] i.e., the entries moved to hardware match on all the header fields the switch can match on. Notice that a microflow cache helps in avoiding the entries dependency issues, which are typical in SDN switches.[28] When the system is operative, the majority of the entries are installed in the software switch, while the hardware hosts entries up to its maximum capacity. The offloading algorithm should guarantee that the entries installed in hardware handle the majority of the network packets, to avoid overloading the software switch.

### 4.2 | Offloading algorithms

Given the hardware switch's flow table size and its entry installation rate (FTEIR) as constraints, an optimal caching algorithm for our d-classifier design should maximize the traffic amount offloaded to the hardware switch. While the optimal algorithm for populating the microflow caches is, generally, a NP-hard problem,[28] we investigate several heuristics, which exploit the OpenFlow primitives (e.g., time-outs and counters) in a lightweight manner. We consider the 3 following approaches:

1. FIFO1: the algorithm implements a generic FIFO flow caching strategy. When a packet is handled by the software switch, a pointer to the matching entry is added to a FIFO queue. Once there is a free space in the hardware switch's flow table, a pointer is taken from the offloading queue and the corresponding entry is installed in the hardware. The algorithm implies a minimum computational overhead, since it does not involve any additional logic or flow statistics evaluation. If the flow size distribution is uniform over the time and the Flow Arrival Rate (FAR): FAR $\sim$ FTEIR, the hardware switch can accommodate a constant share of incoming traffic limited by its table size and the FTEIR, while the rest is handled by the software switch. If FAR $\gg$ FTEIR, the FIFO offloading queue grows indefinitely. Taking into the account that the majority of flows in a typical LTE network are short-lived flows,[17] these flows expire in the overloaded FIFO queue before being offloaded to the hardware switch. In this case, the expired flows do not carry traffic anymore; the load of

[‡]An alternative implementation we are currently exploring leverages the powerful CPU of modern white box switches, such as the ones specified by the OpenCompute project http://www.opencompute.org/projects/networking.

the hardware switch vanishes, and the effectiveness of the algorithm degrades rapidly;
2. FIFO2: to overcome the offloading queue thrashing problem of the previous algorithm, we improve it in the following way. The flow entry is added into the offloading FIFO queue only if a packet for such entry was received in the last second. One of the ways to implement the described logic is to leverage individual flow statistics[29] by polling the perflow counters in the FIFO queue with a 1 second interval. This additional logic allows cutting off the expired flows; however, it still does not perform any optimization of the offloaded traffic share;
3. Heavy Hitters: the algorithm identifies heavy hitters in the FIFO queue allowing for better offloading of network flows to the hardware switch. This is done by marking the entries in the queue that matched a number of packets above a given threshold in the last second. As mentioned before, the logic can poll individual statistics of the flows in the offloading queue. The threshold value is derived from the analysis of the traffic traces and should be changed depending on the specific deployment. Marked entries are selected first for offloading, when there is free space in the hardware switch.

In any case, entries that are cached in hardware are removed when they do not match packets for a period longer than 10 seconds. This guarantees protection from cache trashing effects[30] and simplifies the implementation of the offloading algorithm, at the cost of a less efficient offloading. Recalling the offloading algorithm is implemented by the software switch, which would require to poll hardware switch's counters to implement a more complex cached entries deletion. Moreover, hardware SDN switches' counters may not be completely reliable.[31] Therefore, we use only the software switch's counters and leverage the hardware SDN switches' idle time-out feature.[29] Decreasing the idle time-out on the one hand enables evicting of the expired flows from the hardware cache faster and, on the other hand, increases the churn of cache misses. In turn, cache misses increase perpacket delay and jitter, since their packets need to be processed in the software switch again. The optimal idle time-out value depends on the incoming traffic characteristics and can be derived from flows duration distribution. The evaluation of the algorithms is presented in Section 6.

### 4.3 | Learning packet headers

When a network function modifies network packet headers, the downstream classifier cannot be configured with the correct classification rules. In fact, an opaque network function is applying unknown modification to a packet; thus, the classifier has to first learn the new packet's headers to specify the classification rule for downstream flows. Luckily, the presence of a software switch helps our system in dynamically learning the new headers for a given packet. In particular, the software switch is configured to create a new forwarding entry for a downstream flow, whenever a new upstream flow is detected.[§] The newly generated entry is built to match on the upstream's source and destination IP addresses and transport ports, but switching their positions.

To correctly steer the downstream packet toward the corresponding chain, the classifier looks at the upstream flow's packets source MAC address. In fact, such address corresponds to the last function in the chain, and in our traffic steering method, it also encodes the chain information. This point will be further clarified in Section 5.

Notice that we can apply this packet headers learning technique because we deal with flows that are always initiated in the upstream direction.

## 5 | TRAFFIC STEERING

Traffic steering is the process of defining the network paths for network flows, according to an explicit policy. CATENAE performs traffic steering configuring each of the managed switches (including the classifiers) to classify an incoming packet, retrieve the chain it belongs to, and forward it to the chain's next function. Since Ethernet networks perform packet switching based on Ethernet destination addresses, CATENAE performs packets delivery to a given function, over the SGi-LAN, configuring the switches to rewrite Ethernet addresses. In the remainder of this paragraph, we describe the operations for upstream and downstream cases. Notice that for the remainder of this section, we assume a single-classifier configuration. However, it should be clear that the traffic steering method does not change (and is not affected) when using a 2-classifier configuration instead.

**Upstream**. Upstream flows are first handled at the u-classifier, which uses the FIDs to classify packets and send them to the respective first chain's function. If the function is transparent, the function's switch delivers the packet directly to the function and reclassifies it using the FIDs, after the function's processing. When a function is opaque, packets' header values change, making the system unable to reclassify flows using the FIDs. Also, all the functions coming after an opaque one are handled as opaque functions by the system. In fact, once a packet's header has been changed, the original FIDs do not match the flows anymore. In these cases, classification is achieved creating local virtual L2 networks between a function and its switch. Because network functions typically separate flows received from different L2 networks, a packet will not change its network after the function. Hence, a different (virtual) L2 network per each chain traversing the function helps in associating a packet with its chain. That is, packets belonging to a given chain are tagged with a VLAN tag, which is maintained unchanged when the packet traverses

[§]For example, when using OpenvSwitch,[23] the special learn() forwarding action can be used to achieve this behavior.

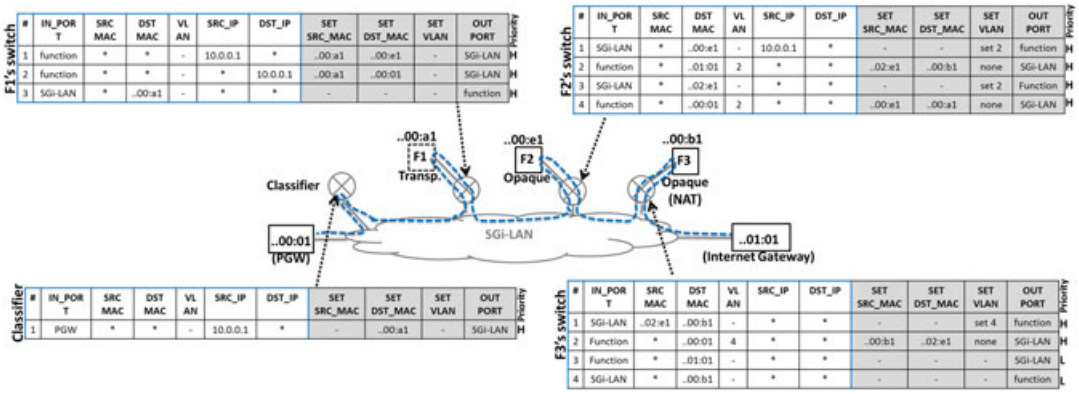


**FIGURE 3** Forwarding tables configuration example, for the steering of a flow with FID "src_IP=10.0.0.1", which traverses the function chain F1, F2, F3

the function[¶]. The VLAN tag is removed before sending a packet back to the SGi-LAN, since it is meaningful only on the switch-function link. However, the classification information is required also at the next function in the chain; thus, this information is encoded in the packets' source Ethernet address. Such an address is generated to be unique for each couple chain/function, and it is generated when the next chain's function is attached to a different software switch. In fact, for functions attached to the same switch, it is enough to read the VLAN tag value. When packets are received at the next function's switches, instead, classification is performed looking at the source Ethernet address.

**Downstream**. Downstream flows are classified either by the d-classifier or by the NATs deployed as chains' last functions. The function chain is then traversed in reverse order. CATENAE operations are again dependent on the type of function the packets traversed. Until there are opaque functions traversed by the downstream flow, the function's switch performs flows classification using VLANs. As in the upstream case, when required, the classification information is encoded in a MAC address value, which this time is written in the packet's Ethernet destination. Recall that this MAC address was generated already for each chain and functions during the handling of the upstream flow. Hence, the location of the generated address was already learned by the SGi-LAN. After the last opaque function (ie, the first one in the perspective of the upstream flow) has processed the downstream flow, the original FID is used to perform packets classification[‖]. Here, we assume an opaque function restores the original packet header for the downstream flow. For example, for downstream flows, a NAT restores the original upstream flow headers, with switched source/destination addresses and transport ports. Thus, the downstream flow coming from an opaque function can be classified at a transparent function's switch that receives it, using the FID.

Figure 3 shows a chain example and the switches's forwarding entries generated to implement such chain for a network flow, in the case of a single classifier deployment. The entries are expressed in an OpenFlow-like format, with a match part, which identifies the flow, and an action part, which specifies the actions that should be applied to the matched packets. A few details can be captured looking at these entries. First, notice that after a function, the packet's Ethernet source is rewritten to the function's MAC address. This rewriting is required to guarantee the correct SGi-LAN's MAC learning. Second, when flows are received from an opaque function, the flow's direction is detected looking at the destination MAC address. We assume that any opaque function is configured to always use IGW and PGW as forwarding gateways for the upstream and downstream directions, respectively.[9] Thus, if the value is the IGW's MAC address, then the direction is upstream; if the value is the PGW's MAC address, the direction is downstream.

## 6 | EVALUATION

This section describes a CATENAE's proof of concept implementation and its evaluation.

**Prototype**. We implemented the SFC controller on top of *Ryu*.[**] The core traffic steering algorithm is implemented in less than 100 lines of python code. We use OpenvSwitch (OVS) as VNFs' switches, and OpenFlow as protocol for the switches configuration. We emulate VNFs running either *click*[32] or *node.js* in Linux containers. Finally, we implemented HSwitch by combining a NEC PF5240 OpenFlow switch with a server running OVS. The server runs also a user space program, which implements the microflow caching logic. In all the tests, the SFC controller runs on a single core of an Intel i5-2540M CPU @ 2.60GHz, using the Python 2.7.3 interpreter shipped with the Ubuntu 12.04.5 LTS distribution. OpenvSwitch (v. 2.3) and VNFs instances run on servers equipped with an Intel CPU E31220 (4 cores @ 3.10GHz).

[¶] In today's network functions, this feature is usually called VLAN separation. The tag is maintained also for the new flows generated as a consequence of the reception of tagged packets. Further information can be found in network functions' manuals, e.g., https://techlib.barracuda.com/bwf/deplyvlan.

[‖] Actually, the FID is modified to switch source address and transport ports with the destination ones, to match the downstream flow.

[**] http://osrg.github.io/ryu

**Number of chains**. CATENAE generates new MAC addresses to support opaque functions. It is unlikely to define a number of chains that could consume the entire MAC address space; however, there is an actual limitation on the number of distinct MAC addresses one can use in an Ethernet network. In fact, Ethernet switches have limited memory to store the associations (address ↔ switch's port) generated during the MAC learning process.[33] For instance, consider chains that include 4 opaque functions on average (excluding the NAT function at the end, for which no MAC address is generated), and assume that a switch can learn 100k associations (e.g., this is the case of the Broadcom Trident switching chip[33]). In this case, the system could support 25k chains (actually slightly less, considering that some MAC addresses are required for, e.g., physical servers and VNFs). Also, each opaque function can be traversed by 4,095 chains at most, since VLAN tags are used to correlate function's entering and exiting flows. While this is a strict limitation, one should consider our initial assumption of supporting VNFs in the number of thousands and notice that the same chain may be applied to several network flows. In fact, operators typically define a single chain for a group of users (e.g., premium users) or services (e.g., web traffic). Furthermore, the actual total number of possible distinct chains is perhaps limited to only few thousands in practice. In fact, consider the case in which a user can pick her services out of a bucket of 10 possible services. If the operator will define a predefined order for the application of such services, such as, anomaly detection is applied before the web proxy, only 1,024 distinct chains could be defined (i.e., $2^{10}$ chains, since each function can be either included or not). Finally, notice that there is no such limitation when dealing only with transparent functions. In such cases, CATENAE does not need to generate any additional MAC address. Moreover, if multiple opaque functions are connected to the same switch, no additional MAC addresses are generated. With K representing the average number of chain's functions attached to the same switch and recalling that after the first opaque function all the remaining chain's functions are handled as opaque ones, in Figure 4, we show the number of required MAC address for a chain's implementation. Notice that an early positioning of an opaque function requires more MAC addresses, while the colocation of functions reduces such requirement.

**Number of flows**. The total number of flows supported by the system defines the number of supported users and how granular their policies can be. CATENAE assigns flows to chains performing classification at the SDN switches. The switch's entries are installed in advance, when a chain is first configured; thus, a switch has to host the entries for all the flows that may traverse it. The number of forwarding entries required to configure a flow in CATENAE scales linearly with the number of functions contained in the chain assigned to the flow. In particular, transparent and opaque functions require 2 and 4 entries each, respectively, per flow. Because we rely on software switches (HSwitch also includes a software switch) we can easily scale to millions of entries per switch. Assuming that an entry requires 50 B of memory (including all the header values[34] and rewriting actions), storing 10˜million entries requires 500 MB of RAM. Such numbers should be sufficient to support millions of users, even considering several policies per user, e.g., distinct chains per users and per user's flows carrying web, voice, video, etc. Also, notice that per flow entries are required only at the classifier and when dealing with transparent functions. In fact, flows that traverse the same chain are identified in an aggregated manner after an opaque function (i.e., they share the same generated MAC address value).

**Configuration time**. The system configuration time depends on the number of entries the SFC controller has to install. The number of entries scales with the product of the number of flows and number of functions per flow's chain. Our SFC controller prototype is developed in python and can send only about 2,200 entry configuration messages per second, limiting the flow configuration performance. Figure 5 shows the rate of flow configurations per second, for chains of lengths between 2 and 5 functions, when functions are either all transparent (but the last one, which is anyway a NAT) or

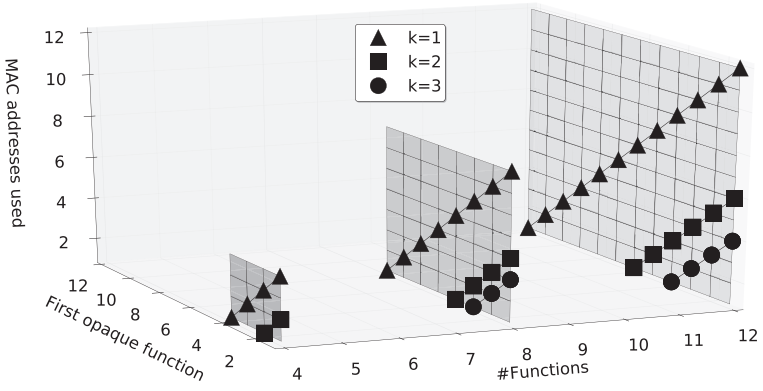


**FIGURE 4** Number of required MAC address for the implementation of a chain, when varying the number of chain's functions and the position of first opaque function in the chain, for different values of K. Where K is the average number of chain's functions attached to the same switch

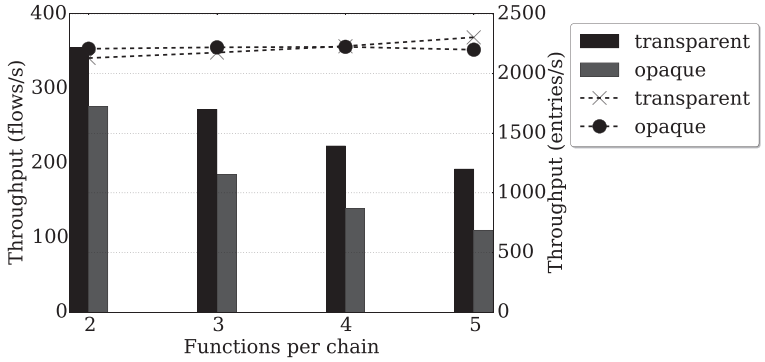


**FIGURE 5** Service function chaining controller throughput in configured flows/s (bars) and generated switch's entries/s (lines)

all opaque. To confirm that this poor performance is a limitation of the Ryu-based implementation, we reimplemented the core algorithm of the SFC controller using the faster Beacon controller.[35] This second implementation achieved, on the same hardware, a flow configuration rate of more than 16k flows per second, in case of chains with 5 opaque functions.

**Flow forwarding delays**. Forwarding entries in CATENAE are installed beforehand; thus, no delay is introduced by the traffic steering method, even when new flows are initiated. This is an advantage when compared to alternative solutions (e.g., these studies[8,21]) that may instead introduce delays on (few) flows' packets. Also, notice that packets processed by HSwitch may have slightly higher delays when the corresponding forwarding entry is hosted in the HSwitch's software layer. However, the additional delay is typically in the microsecond time scale, being comparable to that of any other software switch. Thus, even in this last case, the introduced forwarding delay is usually negligible.

**Overheads**. It is well known that tunneling protocols increase the cost of processing packets at VNFs' switches[36] and VNFs themselves.[8] Furthermore, it is expected that the average packet size in mobile networks will decrease[37] to 384 B in future. For some tunneling technologies, such as VXLAN, this would mean the introduction of more than 14% overhead of on wire transferred bytes (54 B are required for VXLAN encapsulation over IPv4). CATENAE does not use any extra header in the packets, avoiding these overheads, which are common to other solutions (e.g., NSH[11]).

**Data plane scalability**. The main CATENAE's bottlenecks for the system's data plane scalability are the classifiers. In fact, the servers running SDN switches and VNFs, which also handle data plane traffic, could be increased in number to scale with the offered load. Scaling the u-classifier, instead, would require the introduction of additional components, such as load balancers, between the PGW and the u-classifier. Such components would increase the deployment complexity of CATENAE and work against our aim of minimizing the impact on the legacy infrastructure. Likewise, the d-classifier, when present and implemented using HSwitch, may provide limited forwarding throughput if the implemented offloading algorithm is not effective.

Therefore, we built a trace-driven simulator for the classifiers, to analyze their performance under different traffic loads. We validated our simulator by comparing the reported performance with the one measured with our prototype, when running a small scale experiment with synthetic traffic. The validation test shows that for relevant performance metrics, such as the system's throughput, the simulator reports values with a general difference below 1% from those measured on the real system.

Lacking access to real traffic traces, we extracted relevant traffic properties from these studies[17,21] and designed a flow-level trace generator to feed our simulator. The generated traffic trace reproduces the distributions of flow sizes and rates, for the network traffic seen at the PGW, as extracted from this 1 study.[17] Fixing these parameters, we derive corresponding flow durations. As a correctness check, we verify that the CDFs of the generated flow durations as well as the flow's correlation coefficients between size, rate, and duration match the ones reported in this 1 study.[17] The dynamics of the network flows, e.g., flows arrival rate and number of concurrent flows per second, are extracted from this 1 study,[21] which provides base station's statistics of average active users and data connections created per second. As a last check, we compared the numbers of concurrent flows reported in this 1 study[17] with the numbers counted in our trace. Here, notice that the numbers of concurrent flows in our trace depend both from the generated flow durations, computed earlier, and the flows dynamics reported in this 1 study.[21]

We fed our simulator with the generated traffic trace, to verify if the classifier and HSwitch were able to handle the offered load with zero packet loss. Notice that in the scenario presented in this 1 study,[17] the PGW is connected to 22 base stations and handles an aggregated traffic of less than 1 Gbit/s. Considering an average packet size of 512 B,[37] the system has to handle $\sim$ 0.23 million packets per second (Mpps). We configured the simulator to cap the software switch forwarding performance at 1 Mpps. This is a very conservative assumption, since current software switches can forward several Mpps.[23,38] For HSwitch simulation, we assumed the hardware switch could host 100k microflow entries in its forwarding table. Also, we assumed that it could sustain a rate

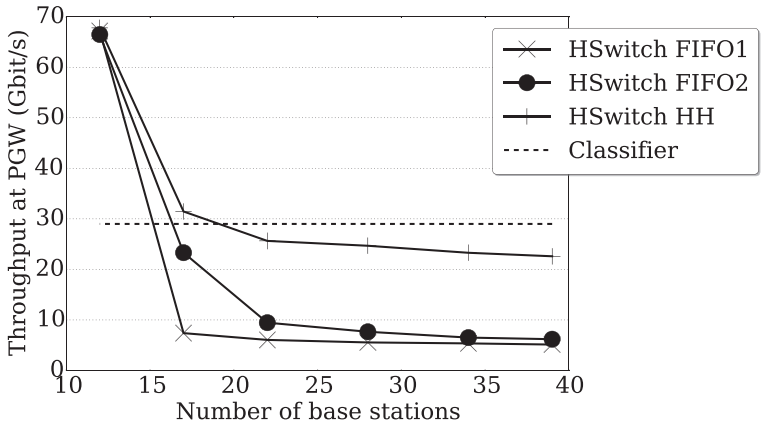


**FIGURE 6** U-classifier and HSwitch scalability when increasing the number of base stations and the aggregated throughput handled by the packet data network gateway

of 700 entry installation/s. Both values are slightly below the actual performance of the NEC PF5240.[39] With this configuration, we simulated 30 minutes of system operations, generating 4.6M flows, in which the u-classifier and HSwitch achieved zero packet loss, i.e., they did not become overloaded with the provided workload.

Considering that a 10x increase in load is expected in 2014–2019[40] which would correspond to an aggregated throughput of $\sim$ 10 Gbit/s in our simulation, we decided to scale the workload to match such numbers. Thus, we performed new simulations to push the system to a corner case and understand its limits. We scaled the offered load in 2 directions: we increased the number of base stations connected to the PGW and the per-flow load. Figure 6 shows the results, plotting the points after which an increase in any of the 2 directions would introduce packet drops. The number of base stations affects the rate of new flows created per second as well as their total number (with 40 base stations, we create up to 8.6M flows). This may impact the distribution of the system load peaks, which in turn impacts HSwitch since only 700 flows per second can be offloaded to hardware. Here, as expected, a smarter caching algorithm, like heavy hitters (with threshold value of 43 pps), can increase the system scalability. The u-classifier, instead, is not influenced by the rate of incoming flows, having a software cache that can be updated fast.

Our test results show that the u-classifier can handle up to 29 Gbit/s of aggregated PGW's throughput: a value 3 times bigger than the 2019's forecast. In fact, the classifier handles only the upstream flows, which in the worst case account for the 15% of the overall throughput, in our trace. That is, 4.35 Gbit/s, which is about 1 Mpps if the packet size is 512 B.

The performance of HSwitch is instead influenced by the adopted offloading algorithm. When scaling to 40 base stations and more than 8M flows, HSwitch can handle only about 8 Gbps of traffic if the FIFO offloading strategies are implemented. The reason is that the system is subject to a significant cache trashing effect, i.e., the flows that are cached from in the HSwitch's hardware layer do not persist for long time in the cache. This is a combined effect of the big number of flows and the way the algorithms select them. In fact, in the FIFO algorithms case, a flow is moved to the hardware layer just in dependence of the time in which it appears in the network. When using a smarter algorithm that tries to select the flows to cache, in dependence of their contributed throughput, HSwitch performance improves, enabling the system to handle up to about 22 Gbps with 40 base stations.

## 7 | DISCUSSION

This section discusses the implication of our design choices and provides a few consideration stemming out from our evaluation results.

**Legacy infrastructures**. CATENAE matches our original aim of minimizing the impact on legacy infrastructures in several ways. First, it can be seamlessly deployed in the LTE architecture. When using the single classifier configuration, CATENAE requires only the installation of software components in the general purpose server attached to the SGi-LAN (cf Section 3), and without requiring any architectural change. This is a unique feature when compared to the related work presented earlier. When using also a d-classifier, there is still only one single hardware switch to deploy in the infrastructure, while all the previous considerations remain valid. Second, it does not use any tunneling protocol, be it an L2 tunneling protocol, such as VLAN, or a higher level ones, such as VxLAN. This provides a number of advantages, and it is another clear distinction point in comparison with the previously mentioned related work. When considering tunneling protocols at the higher network layers, the introduced processing overheads in the servers may be high, unless hardware offloading mechanisms are implemented in the network interface cards (NICs). While it is fair to expect that the most successful protocols, e.g., VxLAN, will be soon offloaded by the majority of the NICs, this is still not the case.[41] Thus, we expect CATENAE will be more efficient in using the server's processing power in the next few years, when servers will be

facing a limited tunneling offloading support. In the case of protocols such as VLAN, for which the offloading is already well established in the NICs, CATENAE provides perhaps an even bigger advantage. In fact, VLAN-like protocols are extensively used to perform logical network separation by a number of systems. In effect, as it became clear in several discussions with network operators, using e.g., VLANs, is in most of the cases not an option, since it would require very complex, time-consuming, and error-prone integrations with the systems that deal with the VLANs management. With CATENAE, the coordination with such systems is not required; in fact, CATENAE operations deal with VLANs only on the link between software switches and VNFs. Third, while other solutions require modifications to the network functions,[8,11] CATENAE supports current network functions with no modifications, leveraging features that are already extensively used, such as VLAN separation. That is, VNFs are considered as black boxes, helping in decoupling the deployment and configuration of network functions from their composition in a chain.[42] Finally, CATENAE nicely integrates with systems that provide the VNFs deployment, such as OpenStack, which in turn can perform, e.g., optimal VNFs placement.

**Hardware network functions**. While CATENAE seamlessly supports software legacy functions, hardware network functions can be only supported if directly attached to a SDN switch. Hence, 2 options are actually viable. In a first case, the hardware function may be attached back-to-back to a server running a software switch. However, the network function may overload the software switch, which, unlike the case of the u-classifier, should handle both upstream and downstream flows. An alternative solution is to deploy a hardware SDN switch. In this second option, a limitation could be the size of the hardware switch forwarding table. In fact, the issue in this case is the same; we faced with the design of the d-classifier; hence, a switch technology like the one implemented for HSwitch could be used to address it. Anyway, please notice that this is a common issue for all the solutions presented in Section 2; furthermore, unlike other solutions that modify L3 headers,[43] CATENAE only rewrites MAC addresses, which is an operation commonly supported in hardware switches.

**Classification**. In our proof of concept prototype, we implemented the u-classifier using a software OpenFlow switch. Such a decision may limit the ability of CATENAE to perform complex classification functions that may require Deep Packet Inspection. However, please notice that CATENAE design does not limit the options for the implementation of a more complex classifier, provided that it exposes an SDN-like interface for configuring the MAC address rewriting operations. In effect, in the evaluation of Section 6, we performed our data plane scalability simulations using a particularly low forwarding capacity for the classifier, with the purpose of evaluating the system in the case in which the classifier is performing complex operations. In fact, the 1-Mpps throughput cap is better suited for a complex network function,[14] while a software switch is usually capable of forwarding packets in the order of 10 Mpps.[38] Similar considerations can be applied for the d-classifier and our HSwitch implementation. In fact, the HSwitch's software layer could be improved to implement a Deep Packet Inspection engine instead of a simple software switch.

**Traffic characteristics**. The HSwitch implements an offloading algorithm to move flow entries to hardware and offload the software switch. Of course, the characteristics of the network traffic have an important influence on the efficiency of the offloading algorithm. In particular, a very skewed traffic distribution, with a few heavy hitters contributing most of the traffic and a vast majority of very small flows, is the best case for our classifier. In fact, only the few entries that handle heavy hitters have to be moved to hardware. Moreover, such heavy hitters are likely to stay active for longer time (and therefore, they would stay longer in hardware), which reduces the cache's churn rate. On the other hand, when the traffic distribution does not have flows that are particularly heavier than others, our classifier would hardly scale its throughput. In such a case, the hardware forwarding table would quickly become full with entries. Still, the throughput contributed by the flows handled by such entries would be modest and will not offload enough traffic from the software switch. Furthermore, smaller flows may increase the churn rate, requiring more often the movement of entries from software to hardware. However, the rate to move entries to hardware is usually rather slow in current hardware SDN switches, which would further prolong the time in which a flow selected for offloading stays in the software switch.

**Metadata**. CATENAE does not support the delivery of metadata to the network functions. For instance, a user's wireless link quality information has to be delivered to the network functions that may need it, e.g., transcoders, using out-of-bound channels. Other solutions support metadata delivery requiring modifications to the VNFs.[8,11]

**Other use cases**. While we presented CATENAE for the application to the mobile networks case, the system can be applied also to other networks that rely on L2 (Ethernet) forwarding. However, it should be clear that a few limitations apply, since CATENAE's assumptions have to hold. In particular, it is required that network flows are initiated in only one direction. Furthermore, CATENAE can deal with deployments of few thousands network functions. Scaling to bigger numbers would require a redesign of the system's traffic steering technique. Finally, the aforementioned impact of traffic characteristics, when using a d-classifier, should also be taken into account to verify that the classifiers can scale with the offered workload.

## 8 | CONCLUSION

We presented CATENAE, an SFC system for the SGi-LAN. CATENAE can be deployed on legacy infrastructures, introducing effective SFC without paying the overheads of

additional packet header fields but still scaling to provide fine grained policies for millions of network flows. Given that service function chain configurations are performed proactively and do not require any dynamic action on the data plane, the only CATENAE's throughput bottlenecks are in the classifiers used to classify upstream and downstream traffic. We presented both an architectural solution and a technological solution to address the 2 cases, respectively. For the upstream direction, we ensure only upstream traffic is handled by the upstream classifier. Given that upstream traffic typically contributes just the 10% of the overall system workload, this allows us to support the traffic growth expected in the midterm using a state of the art software switch as classifier. For the downstream direction, instead, we presented HSwitch, a hybrid software/hardware classifier based on commodity SDN switches. HSwitch caches the heavier flows in its hardware layer, while the software layer guarantees the possibility to install the big number of flows required to support the CATENAE's traffic steering method.

CATENAE lacks some advanced features provided by related work, e.g., support for network functions' metadata. However, the lack of such features is traded with the possibility of deploying the system today, on legacy infrastructures. In effect, our design experience shows that those desired features may be still lacking in other system's components as well. For example, network functions do not support metadata exchange yet. Thus, CATENAE provides support for service function chaining with today's technologies, while the mentioned advanced features could be eventually introduced as the legacy infrastructure gradually evolves, when they are actually needed.

As a final remark, our experience suggests that a design tailored to the problem can help in solving issues that otherwise would require much more expensive solutions. Notice that, in the past, highly-specialized solutions in networking were not considered economically convenient. Traditionally, a network operator would buy an expensive hardware box, which was required to support a number of different deployment scenarios, since its monolithic design would not allow for modifications. Today, we have to observe that the landscape is considerably changed with the introduction of software-based networks. In fact, the "swiss-knife" solution is not required anymore since the solution now runs on a programmable infrastructure. That is, the solution is not a monolithic design that cannot be changed anymore; instead, it can evolve overtime to meet the changing requirements and constraints.

## ACKNOWLEDGMENT

This work has been partly funded by the EU in the context of the "FP7 ITN CleanSky" project (Grant Agreement: 607584).

## REFERENCES

1. Bifulco R, Matsiuk A, Silvestro A. Ready-to-deploy service function chaining for mobile networks. *IEEE Netsoft*; Seoul, Korea, 2016.
2. Honda M, Nishida Y, Raiciu C, et al. Is it still possible to extend tcp? *ACM IMC '11*; Berlin, Germany, 2011.
3. Wang Z, Qian Z, Xu Q, Mao Z, Zhang M. An untold story of middleboxes in cellular networks. *ACM SIGCOMM '11*. Toronto, On, Canada; 2011.
4. ETSI. Network functions virtualisation - white paper.
5. IETF. Service Function Chaining working group. SFC.
6. Mehraghdam S, Keller M, Karl H. Specifying and placing chains of virtual network functions. *IEEE CloudNet*, Luxembourg; 2014.
7. Qazi ZA, Tu CC, Chiang L, et al. Simple-fying middlebox policy enforcement using sdn. *ACM SIGCOMM '13*. Hong Kong, China; 2013.
8. Fayazbakhsh SK, Chiang L, Sekar V, Yu M, Mogul JC. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. *USENIX NSDI '14*. Seattle, WA; 2014.
9. Blendin J, Ruckert J, Leymann N, Schyguda G, Hausheer D. Position paper: Software-defined network service chaining. *IEEE EWSDN '14*. Budapest, Hungary; 2014.
10. Zhang Y, Beheshti N, Beliveau L, et al. Steering: A software-defined networking for inline service chaining. *IEEE ICNP '13*. Göttingen, Germany; 2013.
11. Quinn. Network Service Header. draft-quinn-sfc-nsh-07; 2015.
12. Boucadair M, Jacquenet C, Jiang Y, Parker R, Kengo. Requirements for service function chaining (sfc). Internet-Draft draft-boucadair-sfc-requirements-06, IETF Secretariat; February 2015.
13. Levin D, Canini M, Schmid S, Feldmann A. Incremental sdn deployment in enterprise networks. *SIGCOMM*. Hong Kong, China: ACM; 2013.
14. Martins J, Ahmed M, Raiciu C, et al. Clickos and the art of network function virtualization. *USENIX NSDI '14*. Seattle, WA; 2014.
15. Vissicchio S, Vanbever L, Rexford J. Sweet little lies: Fake topologies for flexible routing. *ACM HOTNETS'14*. Los Angeles, California, USA; 2014.
16. Xu X, Jiang Y, Flach T, et al. Investigating transparent Web proxies in cellular networks. *PAM*. Springer; 2015.
17. Huang J, Qian F, Guo Y, et al. An in-depth study of LTE: Effect of network protocol and application behavior on performance. *ACM SIGCOMM '13*. Hong Kong, China; 2013.
18. Quinn P, Nadeau T. Problem statement for service function chaining. RFC 7498; April 2015.
19. Halpern J, Pignataro C. Service function chaining (sfc) architecture. RFC 7665; October 2015.
20. McKeown N, Anderson T, Balakrishnan H, et al. Openflow: enabling innovation in campus networks. *SIGCOMM Comput Commun Rev*. 2008.
21. Jin X, Li LiErran, Vanbever L, Rexford J. Softcell: Scalable and flexible cellular core network architecture. *ACM CoNEXT '13*. Santa Barbara, California; 2013.
22. Richter P, Allman M, Bush R, Paxson V. A primer on ipv4 scarcity. *ACM SIGCOMM Comput Commun Rev*. 2015;45(2):21–31.
23. Pfaff B, Pettit J, Koponen T, et al. The design and implementation of open vswitch. *USENIX NSDI '15*. Oakland, CA; 2015.
24. Kang N, Liu Z, Rexford J, Walker D. Optimizing the "one big switch" abstraction in software-defined networks. *ACM CONEXT'13*. Santa Barbara, California; 2013.
25. Dusi M, Bifulco R, Gringoli F, Schneider F. Reactive logic in software-defined networking: Measuring flow-table requirements. *Proceedings of the 5th International Workshop on Traffic Analysis and Characterization (TRAC)*. Cyprus; 2014.
26. Lazaris A, Tahara D, Huang X, et al. Tango: Simplifying SDN programming with automatic switch behavior Inference, Abstraction, and Optimization. *ACM CoNEXT'14*. Sydney, Australia; 2014.
27. Bifulco R, Matsiuk A. Towards scalable sdn switches: Enabling faster flow table entries installation. *ACM SIGCOMM*. London, United Kingdom; 2015.
28. Katta N, Rexford J, Walker D. Infinite cacheflow in software-defined networks. *ACM SIGCOMM HotSDN '14*. Chicago, IL; 2014.
29. Open Networking Foundation. Openflow specification 1.4.0.

30. Sarrar N, Uhlig S, Feldmann A, Sherwood R, Huang X. Leveraging zipf's law for traffic offloading. *ACM SIGCOMM Comput Commun Rev*. 2012;42(1):16–22.
31. Hendriks L, De R, Schmidt O, et al. Assessing the quality of flow measurements from openflow devices. *Proceedings of the 8th Traffic Monitoring and Analysis Workshop*, TMA'16. Louvain La Neuve, Belgium; 2016.
32. Kohler E, Morris R, Chen B, Jannotti J, Kaashoek MF. The click modular router. *ACM Trans Comput Syst*. 2000;18(3): 263–297.
33. Stephens B, Cox A, Felter W, Dixon C, Carter J. Past: Scalable ethernet for data centers. *ACM CoNEXT '12*. Nice, France; 2012.
34. Mogul JC, Tourrilhes J, Yalagandula P, et al. Devoflow: cost-effective flow management for high performance enterprise networks. *Hotnets-X 10th ACM Workshop on Hot Topics in Networks*. Monterey, CA, USA; 2010.
35. Erickson D. The beacon openflow controller. *ACM SIGCOMM HotSDN '13*. Hong Kong; 2013.
36. Kawashima R, Muramatsu S, Nakayama H, Hayashi T, Matsuo H. Sclp: Segment-oriented connection-less protocol for high-performance software tunneling in datacenter networks. *IEEE NetSoft '15*. University College London, London; 2015.
37. Stoke Inc. LTE equipment evaluation: Considerations and selection criteria; 2012.
38. Honda M, Huici F, Lettieri G, Rizzo L. mSwitch: A highly-scalable, modular software switch. *ACM SOSR '15*. Santa Clara, CA; 2015.
39. NEC. Programmableflow pf5240 switch.
40. Cisco. Visual networking index: Global mobile data traffic forecast update 20142019 white paper.
41. Guenender S, Barabash K, Ben-Itzhak Y, et al. Noencap: Overlay network virtualization with no encapsulation overheads. *ACM SOSR*. Santa Clara, CA; 2015.
42. John W, Pentikousis K, Agapiou G, et al. Research directions in network service chaining. *IEEE SDN4FNS '13*. Trento, Italy; 2013.
43. Manetti V, Di Gennaro P, Bifulco R, Canonico R, Ventre G. Dynamic virtual cluster reconfiguration for efficient iaas provisioning. *Proceedings of the 2009 International Conference on Parallel Processing*, Euro-Par'09. Springer-Verlag, Berlin, Heidelberg; 2010:424–433.

**How to cite this article:** Bifulco R, Matsiuk A, Silvestro A. CATENAE: A Scalable service function chaining system for legacy mobile networks. *Int J Network Mgmt*. 2017;e1965. https://doi.org/10.1002/nem.1965

## AUTHORS BIOGRAPHIES

**Roberto Bifulco** is a researcher in telecommunications with a focus on programmable networks. His main contributions are in the fields of Scalability and Security of Software-defined Networks, high-performance Network Function Virtualization, and programmable networks design. Since 2012, Roberto joined the NEC Laboratories in Heidelberg, Germany, where he is a Senior Researcher. Earlier, Roberto worked as consultant for small technological enterprises and start-ups. During his career, Roberto has taken part to several research projects funded by the European Commission, such as FP7 ONELAB2, FP7 MPLANE, and H2020 BEBA. He holds a PhD from University of Napoli "Federico II".

**Anton Matsiuk** is a Research Scientist in the SDN research group of the NEC Laboratories Europe. Before joining NEC, he worked in telecommunications and networking areas starting from operation and administration tasks to design and implementation of network management applications for large-scale ISP-operated networks. He holds a Dipl.-Ing. in Telecommunications (2006) and a M.Eng. in Information Technology (2014) as well as industry-specific certifications and awards. His current research interests include SDN-enabled infrastructures for industrial and IoT domains as well as SDN performance, scalability, and programmability challenges.

**Alessio Silvestro** received both his Bachelor and Master degrees in Computer Engineering from the University Federico II, Naples, Italy. Alessio joined the NEC Laboratories Europe in 2015, in the context of the EU FP7 ITN CleanSky project, as Early Stage Researcher. His research is mainly focused in the fields of Software-defined Networking (SDN), Network Function Virtualization (NFV) and Cloud Computing.