Architectural Support for Implementing Service Function Chains in the Internet

Dissertation

zur Erlangung des Doktorgrades
Ph.D.
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

im PhD Programme in Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Alessio Silvestro

Göttingen
im Juni 2018

Betreuungsausschuss: Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Prof. Dr. Jussi Kangasharju,
University of Helsinki, Finland

Prüfungskommission:

Referent: Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Korreferenten: Prof. Dr. Jussi Kangasharju,
University of Helsinki, Finland
Prof. Dr. Simon Pietro Romano,
University of Naples "Federico II", Italy

Weitere Mitglieder der Prüfungskommission: Prof. Dr. Dieter Hogrefe,
Georg-August-Universität Göttingen
Dr. Fabian Schneider,
NEC Laboratories Europe, Heidelberg
Dr. Mayutan Arumaithurai,
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 25 Juni 2018

Acknowledgments

I am deeply honored to have been awarded with the prestigious *Marie Skłodowska-Curie actions* scholarship, within the "EU Initial Training Network (ITN) CleanSky" project. It gave me the opportunity to pursue a doctoral program while traveling around the world and meeting the best researchers in academia and industry. It has been one of the most insightful and thrilling experiences of my life.

I am thankful to my advisor Prof. Dr. Xiaoming Fu for his valuable guidance, suggestions, and feedback throughout the past three years. His long experience and international network of connections, in academia and industry, made him an invaluable Ph.D. supervisor.

I am thankful to Dr. Fabian Schneider for his time, patient, the many meetings, and brainstorming that steered and sharpened my research ideas.

I am thankful to my co-advisor Prof. Dr. Jussi Kangasharju. His many feedbacks, insightful discussions, and his listening skills make working with him a real pleasure.

I would also like to thank Dr. Roberto Bifulco for his time, support, and many challenging questions that helped me strengthen my critical skills.

I am thankful to Dr. Dirk Kutscher. Our career path diverged eventually, however, I am grateful for our rich brainstorming sessions. His mixed experiences between academia and industry have had an important contribution to my research attitude.

To all the professors and researchers cited above goes my greatest and most sincere gratitude as all of you, in different time periods, with your own peculiar ways, and rich experience have helped me being the researcher I am today.

Many thanks go to all the fellows and senior members of the *CleanSky* project. I will always remember the many brainstorming sessions at the various project events, along with the great leisure time we spend together. In particular, my thanks go to Dr. David Koll, Nitinder Mohan, and Besmir Tola.

Further, I could not have worked so hard without having a great atmosphere in my personal life. To this regards, I would like to thank all the friends I have met during the past years in Heidelberg. You made Heidelberg feels like my home, and you became my second family: Roberto B., Patricia, Giuseppe, Martha, Flavio, Ioanna, Jorge, Apostolos, Roberto G., Francesco, Fabio, Chiara, and Giulia.

Further, I would like to thank my friends Pantaleone and Alfonso because our strong friendship makes me feel like I never left "home".

Valeria, deserves a special mention. Her unconditional support, presence, and encouragement over these years have been crucial. Part of this success exists just because of her.

Last but not least, I would like to thank my family: Antonio, Stefania, Dario, and Sara. Having a place you call *home*, in which you feel safe, makes you feel you can conquer the world!

Gottingen, June 2018 *Alessio Silvestro*

This research work has been funded by the joint EU FP7 Marie Curie Actions *CleanSky* Project, Contract No. 607584.

Abstract

Network Functions (NFs) have a crucial role in today's Internet infrastructure. They provide additional services on top of plain connectivity such as content caching, Network Address Translation (NAT), etc. However, standard IP networks staticity hinders the possibility of a flexible deployment of NFs in the Internet. In fact, initially, NFs were introduced in the network by hard-wiring the NF instances on the network traffic's path. This is unacceptable, especially when multiple NFs are chained together to form the so-called Service Function Chain (SFC).

On these grounds, the SFC problem space has attracted the attention of the research community. SFC solutions have proliferated to a great extent, especially with the advance in Software-defined Networking (SDN) and Network Function Virtualization (NFV). However, by comprehensively reviewing state of the art, this thesis shows that Network Providers are reluctant towards state of the art SFC solutions, as they target "clean-state" system architecture i.e., they do not consider prior system architecture and equipment, which translates to greater investment. Therefore, this thesis introduces a "ready-to-deploy" SFC solution that targets legacy Mobile Networks. It aims to fill the gap of the current state of the art helping Network Providers in the transition phase towards more complex and costly SFC solutions.

This thesis further investigates *Internet-wide SFCs*, in which multiple NF providers are involved in the SFC provisioning. State of the art solutions use plain IP routing to steer the traffic through the SFC i.e., each NF composing the chain is identified by its IP address. However, there is no prior work on how to effectively support the SFC Resolution in the Internet. In tackling these challenges, this thesis proposes the Domain Name System (DNS) as a key element, in today's Internet infrastructure, to implement the SFC Resolution. The inefficiencies of the current DNS when dealing with SFCs are highlighted. So, a collaborative SFC Resolution process is proposed. It efficiently supports the SFC Resolution process when multiple and independent NF providers are involved in the SFC provisioning.

Moreover, this thesis highlights the importance of NFs which are strategically placed in the network, when dealing new application scenarios such as Internet of Things (IoT), vehicular networks, etc. In fact, in such use cases, NFs are placed at the network "Edge", in order to reduce the network latency required to reach the NFs. In order to support more variegated use cases, this thesis proposes a *Multi-tier Edge Network* architecture. It considers the case in which there is a large number of heterogeneous edge resources, widely distributed in the network. Therefore, an efficient placement algorithm is proposed. It exploits the multi-tier nature of the proposed network providing an optimized NF placement.

Contents

List of Figures

Acronyms

ISP Internet Service Provider

IP Internet Protocol

SFC Service Function Chain

IoT Internet of Thing

SDN Software-defined Networking

NFV Network Function Virtualization

IDS Intrusion Detection System

NSH Network Service Header

MNO Mobile Network Operator

CDN Content Delivery Network

OTT Over The Top

NF Network Function

OS Operating System

DNS Domain Name System

LTE Long-Term Evolution

UE User Equipment

SGW Serving Gateway

PGW Packet Data Network Gateway

PCRF Policy and Charging Rules Function

IGW Internet Gateway

RSP Rendered Service Path

SF Service Function

SFP Service Function Path

SFF Service Function Forwarder

NAT Network Address Translation

QoS Quality of Service

MEC Mobile Edge Computing

ONF Open Networking Foundation

NDA Non-Disclosure Agreement

ETSI European Telecommunications Standards Institute

ISG Industry Specification Group

VNF Virtualized Network Function

FT Flow Table

FTEIR Flow Table Entry Installation Rate

ESP Edge Service Providers

CSP Cloud Service Providers

CS Cloud Server

CP Cloud Platform

PM Physical Machine

Chapter 1

Introduction

1.1 The Internet Eco-system

The initial idea of the Internet was to build a robust, fault-tolerant communication via computer networks [1]. In order to keep the network structure as simple as possible, Internet has been designed using the end-to-end paradigm [2]. Network's intermediary nodes such as routers, switches, etc. have been designed to perform pure network functions e.g., routing, etc. On the other hand, application logic resides at the communication end-points, generally provided with more computational capability. This design choice was motivated by two main reasons. First, communication end-points, usually x86-based machines, show a high level of programmability provided by their Operating System (OS), especially when compared with hw-based network elements such as routers and switches. Second, the fact that the application logic – which includes the communication state – resides at the communication end-points, make the communication network-failure agnostic i.e., a fault in the network does not affect the application logic. This approach, also defined as "*fate-sharing*", suggests that is acceptable to lose the state information associated with any entity if, at the same time, the entity itself is lost [1]. This aspect was set as top-priority for the initial design, given that Internet was used in the military context.

The Internet eco-system has been constantly evolving ever since, adapting itself and trying to satisfy a number of heterogeneous applications scenarios' requirements. In the very beginning, a limited number of devices (e.g., personal computers, servers)

were connected to the Internet. Their main goal was to communicate in an efficient, robust, and fault-tolerant manner. However, use-cases and requirements are totally changed. For instance, in the era of the Internet of Thing (IoT), besides standard personal computer and servers, also smartphones, smartwatches, cars, and sensors are constantly connected to the internet. A whole new set of application use cases such as Industrial Automation, Virtual Reality, Content Delivery, Vehicular Networks, etc. have proliferated in the Internet to a great extent. Each use case is peculiar in terms of the number of devices involved, their processing capability, and network requirements. For example, Content Delivery Networks (CDNs) require to provide a large amount of data (i.e., in the order of GB) with reduced latency to end-users (e.g., computers, smartphones, etc.). With IoT scenarios, a large number of devices need to send small and frequent data among them or with a central controller. In Industrial Automation scenarios, human-to-machine interaction use-cases require predictable and very low network latency (e.g., in the order of milliseconds) between the connected machines and the central controller. The "Cisco Visual Networking Index Forecast" gives an idea of such a great heterogeneity. By 2021 the annual global Internet Protocol (IP) traffic will reach 3.3 Zettabytes (1000 Exabytes [EB]) per year. The number of devices connected to IP networks will be more than 3 times the global population. IP video traffic will be 82 percent of all consumer Internet traffic [3]. Such variety of application use cases, their heterogeneity in terms of the number of devices and requirements, are steering the decisional process of Internet Service Providers (ISPs) and network operators, shaping the "Internet of tomorrow".

In particular, we can observe that there has been a twist in the economics of the Internet. In former times, ISPs and network operators dominated the revenue from Internet, having as a main income the market for telecommunication (e.g.,

phone, Internet, etc.). As of today instead, Over The Top (OTT) service providers have a bigger share of the market in terms of revenue. The paradox lies in the fact that OTT service providers (e.g., Netflix, Youtube, etc.) are generating an enormous amount of the data in the network. For example, streaming services such as Netflix, Youtube, and Facebook, account for over 70% of peak traffic in North America [4]. Such important amount of data are forcing network operators and ISPs to important investments in order to satisfy their customers' requirement, supporting those services. However, OTT service providers do not share any revenue with network operators. The effects of this phenomenon on the industry have been investigated from the IBM's Institute for Business Value [5].

In the attempt to satisfy customer requirements and win back their share of the market, network operators and ISPs are offering additional services – besides the plain connectivity – that satisfy the customer's quality requirements such as low latency, high throughput, and additional security features. In fact, these additional requirements have important impacts for OTT service providers. For instance, Amazon reported that a latency increase of 100 ms causes 1% loss in their sales [6]. Further, security services are considered of utmost importance in many application scenarios (e.g., industrial automation, IoT, etc.) and are facing significant growth [7].

However, in the attempt of deploying such services, network operators and ISPs face the problem of the IP networks staticity. In fact, traditional IP networks are characterized by a vertical integration between the control plane and the data plane, both implemented at the networking devices. The former decides how to handle network traffic, while the latter forwards traffic according to the decisions made by the control plane. Such integration, bundled inside the networking devices, was considered an important factor for the design of the Internet in the early days as it seemed the best way to guarantee network resilience, which was a crucial design

goal at first. On the other hand, the outcome of such design choice is a very complex and relatively static architecture, as often reported in the state of the art [8–12]. In fact, it is very difficult to add new functions, besides the standard ones. Even a small change in a pre-existing function requires to change the control plane, that is embedded in the data plane (i.e., in the network device). Therefore, usually, a change implies the installation of new firmware and, in some cases, hardware upgrades. Therefore, In order to introduce such functions within the Internet infrastructure, network operators and ISPs carefully place Network Functions (NFs) in the network, typically implemented via expensive, specialized and hard-to-configure equipment, usually referred to as *middleboxes* [1], such as load balancers, Intrusion Detection Systems (IDSs), and firewalls, among others. As of today, middleboxes still play a crucial role in today's networks and DC networks architectures as proved in [13]. Usually, multiple network functions are chained together to form the so-called Service Function Chain (SFC). Users' traffic is steered through the right SFC depending on several parameters. For instance, in the mobile networks, users' traffic is steered through different SFCs depending on their subscription plan.

1.2 High-level Research Challenges & Thesis Contributions

1.2.1 Single-domain SFC

Introduction The NFs composing a single-domain SFC are deployed within a single administrative domain. The (S)Gi-LAN within the Mobile Networks and the

[1] the term refers to the physical appliances, often referred to as boxes, that are placed in the network in order to augment end-to-end connections

North-south or East-west SFCs within DC networks are among the most important use cases. Such use cases are characterized by a full knowledge and control of the underlying network topology. At times, single-domain SFCs are spanned over multiple facilities, within the same administrative domain (e.g., multiple DCs), distributed over the network.

On a deeper analysis of the state of the art, with the regards to Service Function Chains (SFCs) techniques, we highlighted that most of such techniques target greenfield approaches i.e., they do not consider in their design choices, prior infrastructure and network equipment, which represents an important limitation. In fact, network operators and ISPs are reluctant towards such solutions as they are not cost-efficient and involves important investments in terms of design, implementation, and testing of new infrastructure.

Research Challenge The main challenge in the field of single-domain SFCs is to provide efficient SFC systems, that satisfy the requirements such as traffic classification, high throughput, etc. Incremental solutions would foster the widespread of such solutions and motivates infrastructure providers to gradually upgrade their infrastructure.

Contribution The main contribution of this thesis in the research area of Single-domain SFC, is *Catenae* [14], a ready-to-deploy SFC solution for Mobile Networks. It exploits the (S)Gi-lan network and users' traffic properties, to provide an effective SFC solution, without affecting the network infrastructure. In fact, it only requires to use software switches on each server where the NFs are deployed, and to insert a software traffic classifier at the beginning of the chain.

In [15], we present a hybrid hardware-software SDN switch, to further increase the system scalability, and meet the telecom operators requirements, in terms of traffic throughput the SFC architecture can handle. In fact, the designed hardware-software traffic classifier supports the traffic rate of 2015 (i.e., ~1 GB/s) with zero packet loss. Further, it is able to handle ~29 GB/s, which corresponds roughly to three times the expected traffic rate at the Packet Data Network Gateway (PGW) of 2019 (i.e., ~10 GB/s) – about 4 years after the paper submission.

1.2.2 Internet-wide SFC

Introduction Single-domain SFCs represent the predominant use case for SFCs in the Internet. They are effective when applied within a single administrative domain, such as Mobile networks or DC, and they assume that the stakeholder enforcing the SFC has full control and visibility of the underlying network infrastructure. We argue that such assumptions are hindering the wide adoption of SFC techniques in more diverse scenarios, and the possibility to have multiple providers involved in the SFC provisioning. In fact, the NFs composing the SFC might be provided by different Service Providers. As a consequence, in a multi-providers SFC kind of scenario, there is no single entity that has full control and visibility over the underlying network infrastructure. Therefore, the different NF service providers implement choices – which affect the whole SFC provisioning – with knowledge which is local within their administrative domain. Each NF provider decision is independent of the others service providers, and that might lead to several inefficiencies in the SFC provisioning.

Research Challenge In the area of Internet-wide SFC, there are three main research questions that need to be addressed.

I Identify the limitation of current single-domain SFC techniques, when applied to internet-wide SFCs;

II Identify the key characteristics of Internet-wide SFC systems;

III Design of internet-wide SFC systems, that can be incrementally deployed, and nicely fit the current Internet infrastructure.

Contribution In Chapter 4, we present the state of the art of *Internet-wide SFC*, in which the NFs composing a SFC are distributed in the Internet, and multiple stakeholders are involved in the SFC provisioning. We highlight a shared limitation among those solutions. They all use plain IP routing for the traffic steering through the SFC i.e., each network node composing the chain is identified by an IP address. However, they assume that the clients are aware of the IP addresses of the nodes composing the chain, prior to the connection establishment. No prior work is available in the state of the art regarding how client and server agree and share the IP addresses of the NFs composing the chain i.e., SFC Resolution process. In our opinion, this aspect is hindering the deployment of such SFC solutions in the wild.

As a first contribution of this thesis with regards to internet-wide SFCs, in Section 4.4 we identify the Domain Name System (DNS) as a possible candidate to enable internet-wide SFC solutions, given its wide deployment in the current Internet architecture. Therefore, we evaluate the possible strategies that can be implemented with the current DNS architecture and their inefficiencies. In fact, the SFC Resolution process implemented using the current DNS architecture shows bad

performance in terms of NF instance selection. The main reason is that the current DNS is optimized for standard end-to-end connections (i.e., client-server), whereas with a SFC multiple nodes are involved (i.e., client, NFs, server).

As a second contribution, in Section 4.5 we propose *MISE*, an extension to the current DNS that adapts the current behavior to support the resolution process for a set of nodes (e.g., SFC's NFs). We implement such adaption increasing, as little as possible, the information shared among the multiple and independent domains, which enables to achieve near-optimum NF instance selection.

The main system architecture and design have been also submitted as a patent application [16].

1.2.3 Multi-tier Edge Networks

Introduction Network Functions have played a crucial role to enhance end-to-end connections providing additional services on top of the plain connectivity. They have been first introduced within specific domains, in order to provide additional services to ISPs and internet providers. Thus, the same concept has been extended to support further use-cases. For instance, internet-wide SFCs has been proposed in order to enable the case in which multiple NF providers were involved in the SFC provisioning. New application use-cases such as Internet-of-Things (IoT), vehicular networks, etc. are pushing the boundaries even further. In fact, they are requiring to off-load computational tasks with very stringent completion time. However, cloud deployed NFs failed to support such stringent requirement given the possible high network delay required to reach the central cloud location. Therefore, the Edge Computing paradigm has been proposed promising to deploy NFs at the very "Edge"

of the network, in order to reduce the network latency from the users.

Research Challenge In the scope of this thesis with regards to edge networks, we propose three main research questions, which we consider important in this problems space.

I Identify how the SFC techniques can enable Edge Resources deployments;

II Identify possible Edge deployment scenarios and their key requirements.

III Design of Edge-enabled solutions, open to third-party service providers, and that can be incrementally deployed in the current Internet infrastructure.

Contribution In Chapter 5, we investigate the state of the art with a particular focus on the link between current Internet-wide SFC techniques and Edge deployments. We propose a multi-tier edge cloud architecture, in which several heterogeneous edge resources are widely distributed in the Internet. We show how current placement strategies fail to support such heterogeneous edge networks deployments. Therefore, we propose *Mute*, a placement strategy optimized for multi-tier edge cloud architecture. In fact, Mute achieves a significant reduction in edge network delay and completion time when compared to state of the art solutions, when applied to this kind of infrastructures.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: in Chapter 2, we introduce fundamental concepts of “Network Softwarization” such as Software-defined Net-

working (SDN), Network Function Virtualization (NFV), and Service Function Chain (SFC), which represent the technological background for a thorough comprehension of the following chapters. In Chapter 3 we introduce a ready-to-deploy solution for single domain SFCs. In Chapter 4 we explore the multi-domain problem space, find its limitation and propose an enhanced version of the DNS in order to provide close-to-optimum instance selection. In Chapter 5 we define a model of a multi-tier edge network. Thus, we design Mute, a placement algorithm which leverages multi-tier edge architecture to find an edge server which best supports the needs of a requested service. In Chapter 6 concludes and summarizes this thesis work.

Chapter 2

Background

2.1 Software Defined Networking (SDN)

Software-defined Networking (SDN) has been firstly introduced in 2009 by Nick McKeown [17]. Its main goal is to break the vertical integration of traditional IP networks by separating the network's logic (the control plane) from the underlying routers and switches that forward traffic (the data plane). With the separation of the control and data planes, network switches become simple forwarding devices, and the control logic is implemented in a logically centralized controller, simplifying policy enforcement, network (re-)configuration and evolution [11]. In fact, this allows network administrators to take and enforce network-wide decisions, by gaining control of the logic of simple NFs such as a switches and routers. Further, in order to guarantee adequate levels of performance, scalability, and reliability, production-level SDN network designs rely on physically distributed control planes [18,19]. SDN adoption has gained momentum since the introduction of the OpenFlow [20] protocol in 2008, with campus [21], wide area network [19, 22, 23], and datacenter [24] deployments gradually replacing traditional network designs [25].

With the logical separation between the control plane and the data plane, it is possible to abstract the network architecture as shown in Figure 2.1. It presents a three layers network abstraction. The *Network Infrastructure* is composed of network devices (e.g., SDN capable switches). The *Controller Platform* (e.g., the SDN

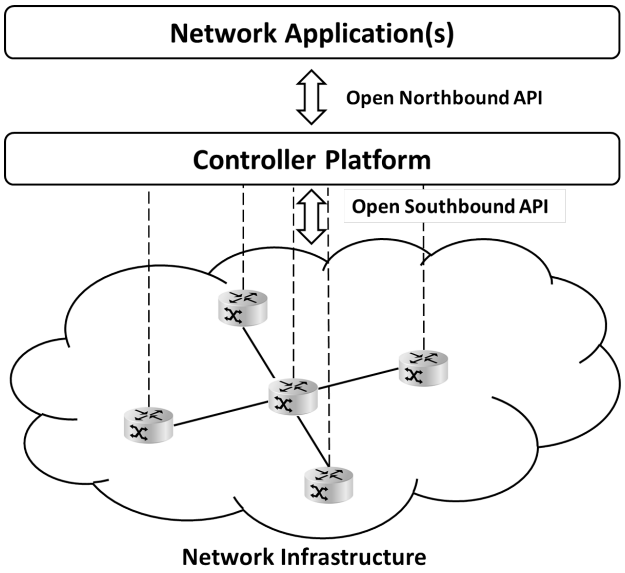

Figure 2.1: Simplified view of an SDN architecture

controller) is used to control the network devices. Thus, the *Network Application* layer is meant to abstract all the network applications such as traffic engineering, monitoring, etc. In fact, it enables a network application to express the desired network behavior without being responsible for implementing that behavior itself. The logical separation in layers introduces the need for well-defined Application Programming Interfaces (APIs) between the layers. In particular, such model introduces two APIs. *Northbound API* defines the interaction between the control plane (e.g., the SDN controller) and the network applications. Whereas, the *Southbound API* defines the interaction between the control plane (e.g., the SDN controller) and the data plane (e.g., SDN-enabled switches). The *Northbound APIs* are generally controller-specific. On the other hand, the *Southbound APIs* are switch-specific. In fact, such APIs need to be supported at the hardware-level. The de-facto standard as *Southbound API* is represented by OpenFlow [20, 26]. OpenFlow-enabled

switches have one or more tables of packet handling rules, that are defined as *flow tables*. Each rule matches a subset of the network flows and performs certain actions (e.g., dropping, forwarding, modifying, etc.) on the flows. Depending on the set of installed rules, an OpenFlow-enabled switch can behave like a router, switch, firewall, or perform other roles (e.g., load balancer, traffic shaper, etc.).

2.2 Network Function Virtualization (NFV)

In 2012 a call for industrial and research action on NFV has been issued by the world's leading network operators with a white paper [27]. In particular, European Telecommunications Standards Institute (ETSI) has been selected by network operators (i.e., AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica, and Verizon) to form the Industry Specification Group (ISG) for NFV, which is named ETSI ISG NFV [28]. ETSI is not the only standard entity dealing with NFV. In fact, also the Open Networking Foundation (ONF) has been active in the area of NFV. For instance, in [29] it is highlighted how an NFV deployment can benefit from the dynamic SDN-enabled service provisioning.

The main idea of NFV – similarly to the concept applied with SDN – is to decouple the logic of the network functions from the physical hw (e.g., server) that they run on. In order to achieve such separation, there are a number of different techniques which affect the way NFs are provisioned. We can summarize those differences as follows [30]:

I *Decoupling software from hardware:* the separation of the the hardware middleboxes from the software control logic running on top of it allows to separate the development timeline and the maintenance for software and hardware.

II *Flexible NF deployment:* NF service providers can flexibly assign the NF instances to the shared hardware infrastructure.

III *Dynamic scaling:* the decoupling of NF software control logic from the underlying hardware enables the NF software component to be instantiated on generic hardware. As follows, it is easier for NF service providers to scale the number of NFs deployed in a more dynamic fashion according the current traffic requirements.

It is worth to mention that the outcome from the NFV research trend does not necessarily require to implement the NFs on virtualized hardware. In fact, NF service providers, when high-performance are required, still deploy NFs on *bare-metal servers*. The main advantage is represented by the fact that now it is possible to deploy NFs on commodity off-the-shelf server, rather than on specialized hardware middleboxes. Nonetheless, running the NFs in virtualized environments is a very strong selling points of NFV, as it provides important improvements in terms of flexibility, dynamic resource scaling, energy efficiency.

Further, it is also important to mention that even if NFV is proposed as a standalone research trend, most of the flexibility claimed would not be possible without the flexibility brought in the whole picture from the SDN research. For example, while a NF can be easily migrated from a physical server to a second one, for scalability reason, if the network does not react fast, it would not be so effective. For this reason, we highlight that NFV and SDN present solutions which are complementary to each other.

2.3 Service Function Chaining (SFC)

The IETF is the main standard organization that is dealing with SFC, stating the SFC problem in [31] and defining the architecture of a SFC system in [32]. Figure 2.2 shows a graphical representation of the architecture proposed by the IETF.

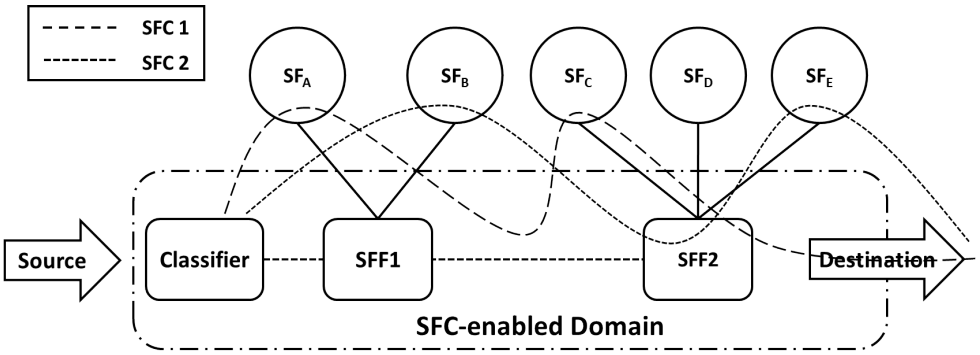

Figure 2.2: IETF SFC architecture

A network function is relabeled *Service Function (SF)*. Thus, a 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 (classifier), to perform traffic steering through the correspondent chain. The service classification function adds a 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 an 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)*. The 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 [32] architecture assumes that SFs can deal with the SFC-encapsulation format, SFC-unaware functions (eg, legacy network functions) are supported by the usage of a SFC-proxy as shown in Figure 2.3. An 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.

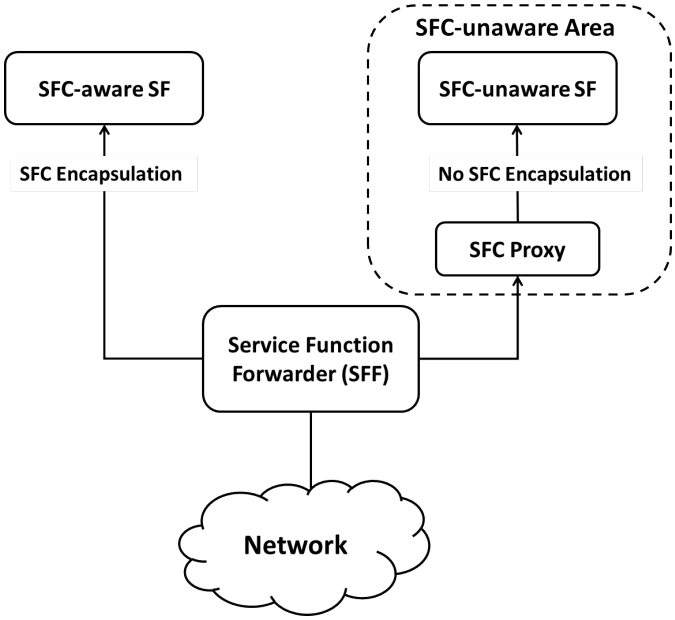

Figure 2.3: IETF SFC architecture with proxy

Network Service Header While there are no standards defined for the SFC-encapsulation format, an IETF draft 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, eg, based on VxLAN, connects SFFs. The original data unit, eg, an L2 frame or an L3 packet, thus, is encapsulated within different transport protocols such as VLAN, VxLAN, GRE, Ethernet, etc. When an SF receives a Q5 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.

2.3.1 Use Cases

Service Function Chains can be applied in different scenarios. In particular it is possible to apply the following categorization.

2.3.1.1 Single-domain SFC

A *Single-domain SFC* is spanned within a single administrative domain boundaries. There are several examples of such domains. (S)Gi-LAN is the place where Mobile Network Operators (MNOs) place a set of services in order to enhance their users' connection [33] as show in Figure 2.4. Within a single Data Center architecture there are usually several SFCs [34]. For instance, there are *North-South SFCs* for traffic destined or originated from a VM within a DC. Thus, there are *East-West SFCs* for traffic originated and destined from within the same DC.

Definition 2.1 (Single-domain SFC) *A single-domain SFC is defined as a SFC instance in which there is a single entity (e.g., DC administrator) has full control and visibility of the underlying network infrastructure including network elements and servers.*

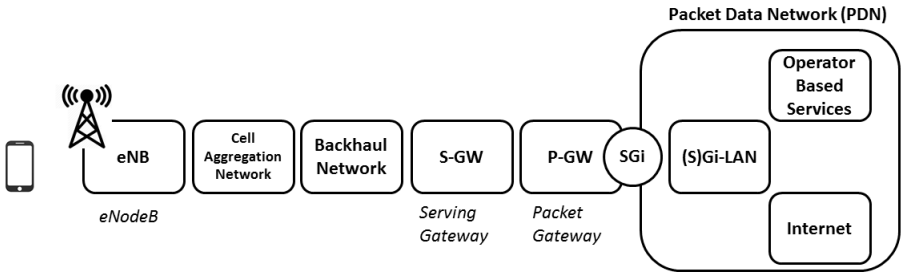

Figure 2.4: (S)Gi-LAN in the Mobile Network

2.3.1.2 Inter-domain SFC

In several use cases, there is the need to go beyond a single domain boundaries. In fact, several real use-cases combine intra-domain and inter-domain SFCs. It is a common case, for instance the *East-West SFCs* within multiple DCs. Migrate a VM

from a DC to another DC – managed by a single administrator e.g., Amazon's DC in Germany to Amazon's DC in the US. However, in this case, intra-domain SFC techniques still work. Tunnells are then created to connect the various domains.

2.3.1.3 Internet-wide SFC

Steering techniques used with intra/inter-domain SFCs cannot be applied if there is not a single entity which has full visibility and control on the underlying network infrastructure. In fact, this is a limiting assumption when there are multiple NF providers involved in the SFC provisioning as shown in Figure 2.5.

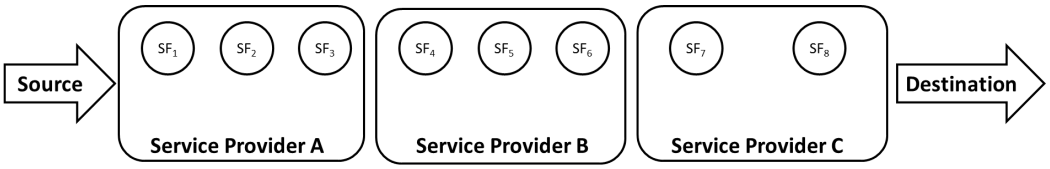

Figure 2.5: Multi-domain SFC

Definition 2.2 (Internet-wide SFC) *An internet-wide SFC is defined as a SFC instance in which the NFs composing the SFC are distributed int the Internet. No further assumptions are defined regarding the ownership of the underlying infrastructure.*

Such solutions cannot be applied in scenarios where there is no full control and/or partial visibility of the underlying network infrastructure, for example when the in-network services are outsourced to a third-party provider, e.g., deployed in the cloud [13]. In order to overcome such limitations, other SFC techniques use plain IP routing and steer the traffic through the SFC only based on the middleboxes' IP addresses [35–37]. We refer to such SFCs as *Internet-wide SFC*. However, such

solutions share a common aspect: they assume the clients know middleboxes' IP addresses before the connection establishment.

2.4 SDN & NFV & SFC

SDN has also contributed to the virtualization of the network infrastructure, providing the foundation to isolate, abstract, and share the network resources. On the other hand, NFV has been proposed to innovate in the service delivery by using standard computing virtualization technology to consolidate in commodity hardware (i.e., standard high volume servers, storage, and switches) the functions previously performed by specific hardware appliances. Virtualized Network Functions (VNFs) composing a service chain represent the basic elements to achieve the complete virtualization of service delivery and are commonly based on computing resources. The interconnection of VNFs, or traffic steering, is a challenging goal for the underlying network infrastructure. The migration of VNFs and dynamic composition of services make this task even harder than in legacy networks. Adapting the use of resources to the actual demand is one of the main outcomes from a virtualized infrastructure, providing elasticity of resources instead of over provisioning

SDN and NFV are complementary technologies, and each one can leverage off the other to improve the flexibility and simplicity of networks and service delivery over them. Initially, NFV has been introduced as a stand-alone project as well as an SDN-agnostic initiative. SDN was tought to modify the network behaviour flexibly, whereas the NFV initiative had the direction to enable to deploy network functions, from proprietary middleboxes to commodity hardware (i.e., standard high-volume servers, storage, and switches). However, SDN and NFV are complementary tech-

nologies, and each one can leverage off the other to improve the flexibility and simplicity of networks and service delivery over them [38]. Therefore, the research community is moving towards SDN-enabled NFV solutions. SDN-enabled NFV solutions are particularly relevant when speaking about Service Function Chaining. In such cases, NFV solutions are used to deploy the network functions. However, SDN techniques are required, in order to efficiently steer the traffic towards the right service chain.

Chapter 3

Single-domain Service Function Chains

3.1 Introduction

As introduced in Section 1, network operators deploy network functions to enforce their policies and to provide additional services on top of plain connectivity [39]. Content caching, Network Address Translation (NAT), TCP optimization, video transcoding, HTTP header enrichment, are examples of such services. Despite their ubiquitous usage [40], 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 are not acceptable when network functions are implemented by means of software running in virtual machines, as envisioned in the case of Network Function Virtualization (NFV) [41]. 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 in Service Function Chain (SFC) systems [42], 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 the flow through the functions of such chain, according to the chain's functions ordering [43]. 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 bi-directional, 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 two 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 [44].

To address these challenges, a number of SFC systems have been already proposed [44–48]. However, they usually target green field or long term deployments. In fact, they require a number of changes either in the network hardware [44] or in the network functions [48], or in both [45]. In other cases, they require modifications to the network architecture [46]. Ready to deploy solutions, which don't 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 [44,47]. 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 minimum impact on the legacy infrastructures [49, 50].

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 SDN software switches, which are anyway deployed at the servers hosting VNFs [51]. Flow re-classification after a VNFs is done by creating per-VNF VLAN topologies, using an approach conceptually similar to [52]. By implementing a proof of concept prototype, we demonstrate that CATENAE does not add per-packet 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.

3.1.1 Contributions

This chapter is structured as follows:

- Section 3.2 introduces background information on the mobile networks where CATENAE is deployed, and introduces related work;

- Section 3.3 presents the CATENAE's design, describing its architecture and deployment options;
- Section 3.4 describes the design of the classifiers employed to assign network traffic to service function chains;
- Section 3.5 presents the CATENAE's traffic steering method and provides a concrete example of the method applied to a network scenario;
- Section 3.6 reports the results of our prototype evaluation;
- Section 3.7 discusses our design in the light of the evaluation results and further describes differences with related work.

We conclude this chapter in Section 3.8.

3.2 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 *Single-domain SFC* performed by standard organizations like IETF, and provides an overview of the solutions proposed by the research community.

3.2.1 Mobile networks

Catenae focused on implementing SFC in Long-Term Evolution (LTE) cellular networks (cf. Fig. 3.1), one of the main use-case for Single-domain SFCs. An 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 eNB encapsulate UE's network flows in a tunnel that, traversing

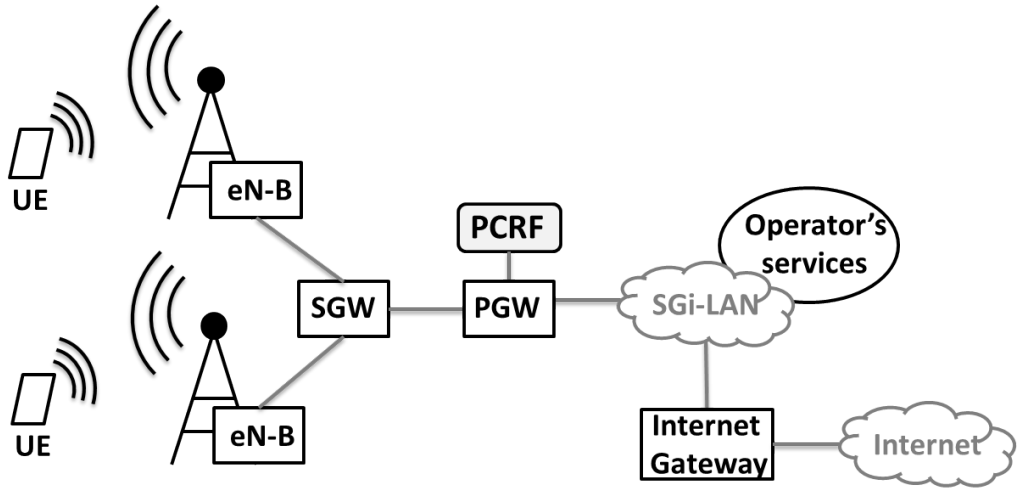

Figure 3.1: LTE network architecture.

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 and Charging Rules Function (PCRF) provides the PGW with the policies to handle users' traffic, e.g., it provides the QoS configuration. After the PGW, user's packets are sent to the SGi-LAN, which is the place where the operator provides additional services [53]. 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 don't 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 Sec. 3.3. 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 a 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 which are typical of LTE deployments. That is, the upstream flows (i.e., those generated at the users) are usually much smaller in size than the downstream flows [54]. Also, the connections are (almost) always initiated in the upstream direction.

3.2.2 Single-domain SFC in standards

The IETF is the main standard organization that is dealing with SFC, stating the SFC problem in *RFC7498* [31], and defining the architecture of a SFC system in *RFC7665* [32]. In the IETF architecture, a network function is relabeled Service Function (SF). Thus, a *Service Function Chain* is an abstract definition of an ordered set of SFs. To specify additional constraints about the chain, *RFC7665* introduces the concepts of Service Function Path (SFP) and Rendered Service Path (RSP). The former is a constrained specification of the previously defined Service Function Chain. For example, it may specify requirements such as Quality of Service (QoS), or it may limit the implementation of a chain to use only a subset of the overall network infrastructure. An RSP, instead, describes how a chain is actually realized in the network. For instance, while a *Service Function Chain* defines SFs in terms of function types, an RSP defines the specific SF instances (including their exact network location) traversed by the network packets assigned to such RSP.

The incoming traffic, e.g., in the upstream direction, at the edge of a SFC-enabled Domain, is classified by a Service Classification Function, in order to perform traffic steering through the correspondent chain. The Service Classification Function adds a 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. Since 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 a SFC-Proxy. An 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 in *RFC7665* the definition for the SFC-Encapsulation format is left open, *RFC8300* [48] defines the Network Service Header (NSH). The NSH is composed by 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 (24 bits, i.e., up to 16M RSP) to identify the

chains and a Service Index (8 bits, i.e., max 256 hops per chain) to provide the 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, GRE, Ethernet, etc. When a SF receives a packet coming from a Service Chain, it will decrement the Service Index header in order 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 re-classify the packets after a SF.

3.2.3 Single-domain SFC in research

A number of proposals have been presented by the research community, in order to address the challenges of Single-domain SFC.

SIMPLE [44] provides SFC using an SDN network. It implements inter-switch tunnels to aggregate the traffic with common destinations, in order 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 [45] suggests the modification of the network functions in order to provide contextual information, in the form of a tag, which is added to the processed network packets, in order 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 an SDN network to perform traffic steering is the solution adopted also by StEERING [47]. In this case, the authors leverage a smart encoding of the forwarding rules in a multi-table switch's pipeline, in order 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 presence of opaque network functions.

Finally, SoftCell [55] 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 SGW and PGW functions, and therefore removes LTE's mobility management introducing a custom solution instead. Traffic classification is performed at switches co-located 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.

3.3 Design

This section presents our design choices, the CATENAE's architecture, and provides an overview of 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, i.e., 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, since 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 two 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.

3.3.1 Architecture

CATENAE's architecture (cf. Fig. 3.2) 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; 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.

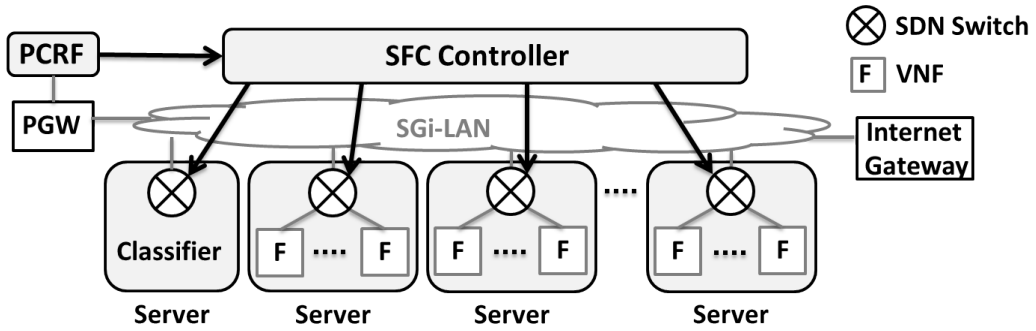

Figure 3.2: CATENAE's architecture.

The SFC Controller offers a function chains configuration interface, which could be connected to e.g., the PCRF of the LTE architecture. Upon reception of a chain installation request, the SFC controller 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 VMs management system.

CATENAE supports two 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 employ 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 two deployment options.

3.3.2 Two-classifiers deployment

In a two classifiers 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 two 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 Sec. 3.4 we further elaborate on these issues.

3.3.3 Single classifier deployment

In the majority of mobile network deployments, there is always a NAT function employed in a chain [40], since private IP addresses are usually in use on the UE-side [56]. 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, in order 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. Since the NAT function is first hit by the already classified upstream traffic, the NAT will associate any downstream traffic to its upstream flow [55]. 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.

3.3.4 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 employed, the CATENAE deployments involves also the installation of the hardware SDN switch used to implement the classifier.

3.4 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 service 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 employs two 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, in respect to the network functions, but only upstream flows are processed by the u-classifier. In our architecture, this classifier is implemented using an SDN software switch. Having a software switch handling all the traffic coming from the PGW may rise scalability concerns, however, upstream flows are contributing just a fraction of the overall load (cf. Sec. 3.6). 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 employs 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 [57].

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. Sec.3.6). 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 [58]. Also, a strategy that capitalizes on the table space, by installing entries only when a chain's flow is actually active [45, 59], 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 [60, 61].

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 [62]. The remainder of this section describes our implementation of the d-classifier.

3.4.1 d-classifier design

Our d-classifier design [61] connects a hardware SDN switch with a server that runs a software SDN switch[2]. 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 micro-flow cache [57], i.e., the entries moved to hardware match on all the header fields the switch can match on. Notice that a micro-flow cache helps in avoiding the entries dependency issues which are typical in SDN switches [62]. 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.

[2] 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.

3.4.2 Offloading algorithms

Given the hardware switch's Flow Table (FT) size and its Flow Table 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 micro-flow caches is, generally, an NP-hard problem (it can be demonstrated it reduces to a knapsack problem), we investigate several heuristics which exploit the OpenFlow primitives (e.g. timeouts and counters) in a lightweight manner. We consider the three following approaches:

- **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 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 $>>$ 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 [54], 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 the hardware switch vanishes and the effectiveness of the algorithm degrades rapidly;
- **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 [63] by polling the per-flow counters in the FIFO queue with a one second interval. This additional logic allows cutting off the expired flows, however, it still does not perform any optimization of the offloaded traffic share;

- **HH** (Heavy Hitters): the algorithm identifies "heavy hitters" in the FIFO queue allowing to increase the load of 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 [64] and simplifies the implementation of the offloading algorithm, at the cost of a less efficient offloading. Recall 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. Instead, we leverage the SDN switches idle timeout feature [63]. Decreasing the idle timeout 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 per-packet delay and jitter, since their packets need to be processed in the software switch again. The optimal idle timeout value depends on the incoming traffic characteristics and

can be derived from flows duration distribution. The evaluation of the algorithms is presented in Sec. 3.6.

3.4.3 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 an unknown modification to a packet, thus the classifier has to first learn the new packet's headers in order 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[3]. The newly generated entry is built in order to match on the upstream's source and destination IP addresses and transport ports, but switching their positions.

To correctly steer the downstream packet towards 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 Sec. 3.5.

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

[3]For example, when using OpenvSwitch [57], the special *learn()* forwarding action can be used to achieve this behavior.

3.5 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 two-classifiers configuration instead.

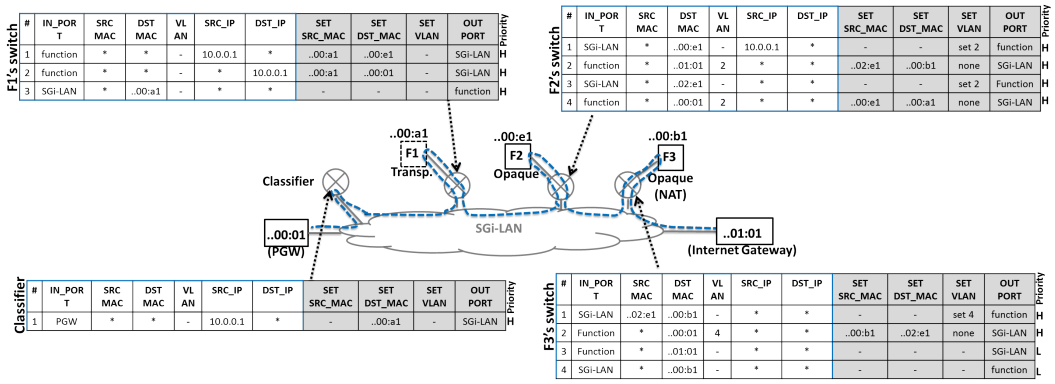

Figure 3.3: Forwarding tables configuration example, for the steering of a flow with FID "src_IP=10.0.0.1", which traverses the function chain F1, F2, F3.

3.5.1 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 re-classifies 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 don't match the flows anymore. In these cases, classification is achieved by creating local virtual L2 networks between a function and its switch. Since 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 the function[4]. 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

[4]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.

Ethernet address.

3.5.2 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 traverse. 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 function 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 (i.e., the first one in the perspective of the upstream flow) has processed the downstream flow, the original FID is used to perform packets classification[5]. Here, we assume an opaque function restores the original packet header for the downstream flow. E.g., 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 3.3 shows a chain example and the switches' 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

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

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 by 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 [46]. 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.

3.6 Evaluation

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

3.6.1 Prototype

We implemented the SFC Controller on top of *Ryu*[6]. 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 the protocol for the switches configuration. We emulate VNFs running either *click* [65] 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 micro-flow caching logic. In all the tests, the SFC Controller runs on

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

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. OVS (v. 2.3) and VNFs instances run on servers equipped with an Intel CPU E31220 (4 cores @ 3.10GHz).

3.6.2 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 $\leftrightarrow$ switch's port) generated during the MAC learning process [66]. 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 [66]). 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 4095 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 a few thousand 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 1024 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 Fig. 3.4 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 co-location of functions reduces such requirement.

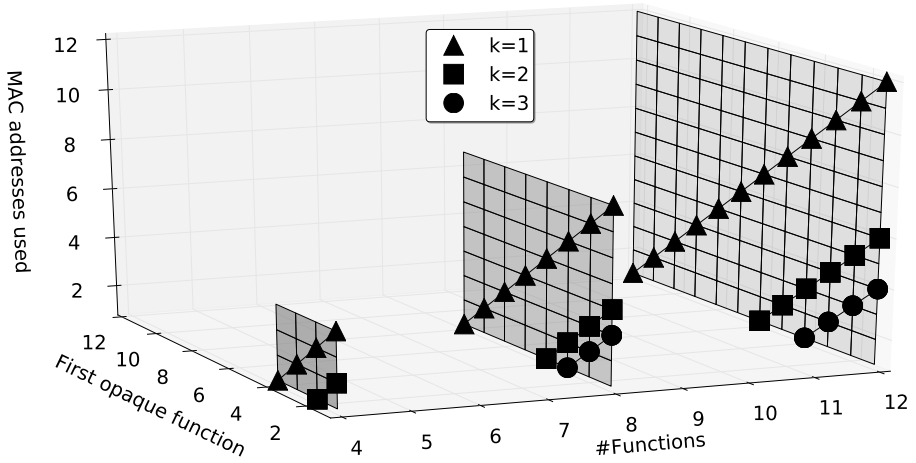

Figure 3.4: Number of required MAC address for the implementation of a chain, when varying the number of chain's functions and the position of first opaque function in the chain, for different values of K. Where K is the average number of chain's functions attached to the same switch.

3.6.3 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. Since 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 50B of memory (including all the header values [67] and rewriting actions), storing 10 million entries requires 500MB 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).

3.6.4 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 2200 entry configuration messages per second, limiting the flow configuration performance. Figure 3.5 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 all opaque. In order to confirm that this poor performance is a limitation of the Ryu-based implementation, we re-implemented the core algorithm of the SFC controller using the faster Beacon controller [68]. 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.

3.6.5 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., [45, 55]) that may instead introduce delays on (few) flows' packets. Also, notice that packets processed by HSwitch may have a slightly higher delay 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.

3.6.6 Overheads

It is well known that tunneling protocols increase the cost of processing packets at VNFs' switches [69] and VNFs themselves [45]. Furthermore, it is expected that the average packet size in mobile networks will decrease to 384B [70] in future. For some tunneling technologies, such as VXLAN, this would mean the introduction of more than 14% overhead in terms of on wire transferred bytes (54B 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 [48]).

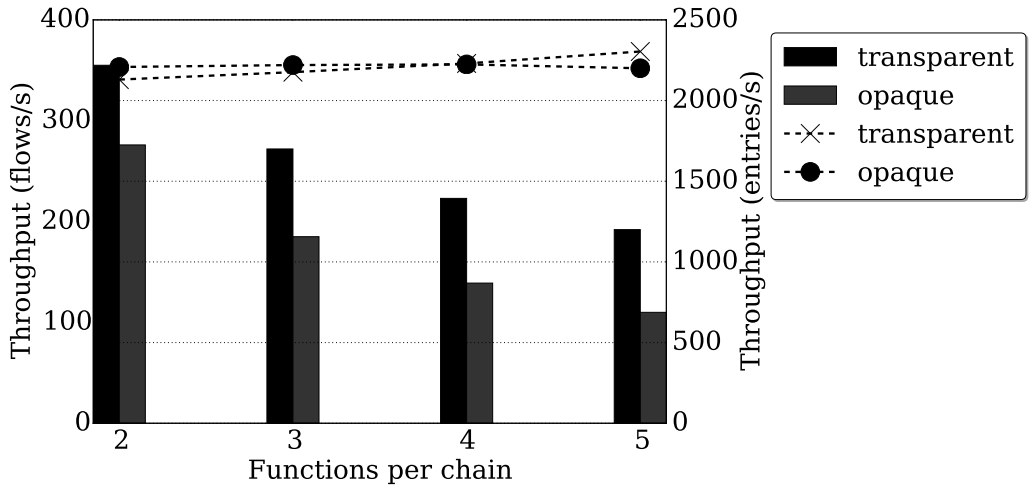

Figure 3.5: SFC Controller throughput in configured flows/s (bars) and generated switch's entries/s (lines).

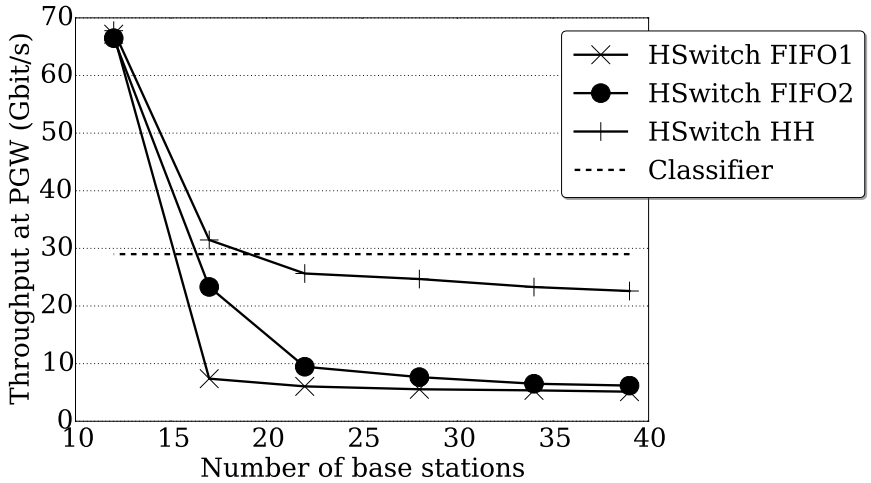

Figure 3.6: U-classifier and HSwitch scalability when increasing the number of base stations and the aggregated throughput handled by the PGW.

3.6.7 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, in order 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 [54,55] 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 [54]. 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 [54]. The dynamics of the network flows, e.g., flows arrival rate and a number of concurrent flows per second, are

extracted from [55], which provides base station's statistics in terms of average active users and data connections created per second. As a last check, we compared the numbers of concurrent flows reported in [54] with the numbers counted in our trace. Here, notice that the numbers of concurrent flows in our trace depend both on the generated flow durations, computed earlier, and the flows dynamics reported in [55].

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 [54], 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 512B [70], the system has to handle ∼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 [57, 71]. For HSwitch simulation, we assumed the hardware switch could host 100k micro-flow entries in its forwarding table. Also, we assumed that it could sustain a rate of 700 entry installation/s. Both values are slightly below the actual performance of the NEC PF5240 [72]. 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 [73], which would correspond to an aggregated throughput of ∼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 two directions: we increased the number of base stations connected to the PGW and the per-flow load. Fig. 3.6 shows the results, plotting the points after

which an increase in any of the two 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. This impacts HSwitch since only 700 flows per second can be offloaded to hardware. Here, as expected, a smarter caching algorithm, like HH (with a 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 29Gbit/s of aggregated PGW's throughput: a value three-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. I.e., 4.35 Gbit/s, which is about 1 Mpps if the packet size is 512B.

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 a 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 employing 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.

3.7 Discussion

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

3.7.1 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. Sec. 3.3), and without requiring any architectural change. This is a unique feature when compared to the related work presented earlier. When employing 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 [74]. 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, in most of the cases, is 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 [45, 48], 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 [75]. Finally, CATENAE nicely integrates with systems that provide the VNFs deployment, such as OpenStack, which in turn can perform, e.g., optimal VNFs placement.

3.7.2 Hardware network functions

While CATENAE seamlessly supports software legacy functions, hardware network functions can be only supported if directly attached to an SDN switch. Hence, two 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 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 Sec. 3.2, furthermore, unlike other solutions that modify L3 headers [76], CATENAE only rewrites MAC addresses, which is an operation commonly supported in hardware switches.

3.7.3 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 (DPI). 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 Sec. 3.6, 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 [51], while a software switch is usually capable of forwarding packets in the order of 10 Mpps [71]. 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 DPI engine instead of a simple software switch.

3.7.4 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 [45, 48].

3.8 Conclusion

We presented CATENAE, a 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 two 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 mid-term 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.

Chapter 4

Internet-wide Service Function Chains

4.1 Introduction

Since the early stage, the research community has been mainly focused on Single-domain SFCs. Mobile networks and DC networks are the most notable use-cases in such problem space. Their constraints, technical specifications, and requirements, have steered the Single-domain SFC solutions of the research community. At high-level, we can characterize single-domain SFC systems as follows:

I Full visibility of the network topology where the NFs are deployed;

II Full control of the underlying network topology i.e., they assume to have the ability to make changes to the network infrastructure, for instance with SDN;

III SFC enforced transparently to users' traffic i.e., users are SFC-unaware.

The full visibility and control of the underlying network topology properties originate from the fact that NFs provider and consumer are represented by the same stakeholder in the main use cases (e.g., mobile network, DC network). In fact, Mobile Network Operators (MNOs) and DC operators represent the entity defining the SFC, and at the same time, enforcing it. Further, in single-domain SFCs users are generally SFC-unaware because the NFs are introduced by the content server and/or the infrastructure operator (e.g., MNO, DC), and do not provide any explicit service to the clients.

We argue that the above-mentioned assumptions are hindering the wide adoption of SFC techniques in more diverse scenarios. Therefore, in this chapter, we explore the more general case of *Internet-wide SFCs*, in which multiple stakeholders are involved in the SFC provisioning, and the NFs are not constrained within a single domain and are rather distributed in the network. Several SFC systems, that respect the proposed definition of *Internet-wide SFC*, have been proposed by the research community [35–37, 77]. Such solutions enable to distribute the NFs in the network, without any particular constraint. Further, they do not require any change to the network control plane, as they use plain IP routing to steer the users' traffic through the SFC. As follows, each NF is identified in the network by its IP address i.e., a SFC instance is identified by a set of IP addresses.

On the other hand, the mentioned internet-wide solutions share a similar aspect, that in our opinion, limits their the wide deployment in the network. In fact, [35–37, 77] assume that the client – which is the entity initializing the connection establishment – is aware of the NFs' IP address prior the connection establishment. Likewise, they consider the NFs' IP address discovery as an orthogonal problem to the SFC provisioning. Currently, and to the best of our knowledge, there is no prior work available on how to approach the SFC resolution problem, and which are the challenges involved when dealing with real-world deployments of a SFC Resolution process.

In this chapter, we first envision, in the current Internet landscape, the Domain Name System (DNS) system to be a candidate for the SFC Resolution process. In fact, for each client's request, it selects a serving node, among a pool of service instances, based on multiple properties e.g., network location, availability, etc. Further, it returns to the client the IP address of the selected instance, which is then used by the client to initiate the end-to-end connection – as required by [35–37, 77].

Nonetheless, the current DNS resolution process is optimized for standard end-to-end connections. In fact, it assumes that the domain name to resolve represents the communication's end-host e.g., content server. Whereas when dealing with SFCs, the resolution of the server's domain name represents only one case. Thus, it includes the resolution of the domain names associated with the NFs composing the SFC. And in such cases, the serving instance has to be selected based on the client's location – as with the standard DNS – as well as on the next hop within the SFC. However, no prior work is available how to approach the SFC Resolution process in the current Internet architecture, likewise, the efficiency of the current DNS system when dealing with SFCs.

4.1.1 Contributions

In Section 4.2 we introduce DNS background information, important for a thorough comprehension of the design choice implemented in the following sections. In Section 4.3 we formulate the *SFC Resolution* in detail, highlighting the properties and the challenges that a SFC Resolution process show in practice. In Section 4.4, we present the possible SFC Resolution strategies and their inefficiencies, when implemented with the current DNS architecture. So, in Section 4.5 we present a *Collaborative SFC Resolution* process. Two DNS extensions are proposed enabling to share minimal information among multiple and independent providers. They enable to enhance the instance selection for each NF provider, allowing close-to-optimum results as proved in Section 4.6. We present related work in Section 4.7 and conclude the chapter in Section 4.8.

4.2 Background

When a client connects to a web server (e.g., www.example.edu), she needs to resolve the related domain name before establishing the connection as shown in Figure 4.1. Therefore, the client sends a DNS query to its local DNS. The local DNS acts as a recursive DNS – it resolves the domain name hierarchy (i.e., Root, .edu) and then returns the result to the client. In our example, Root, .edu and the content server DNS act as iterative DNS. They return either the best answer based on their responsibility zone (e.g., content server DNS) or refer to a DNS server of a lower level of the domain name space (e.g., Root and .edu DNS). The server's authoritative DNS selects the serving node(s) – among the others parameters – based on the client (or its local DNS) network location.

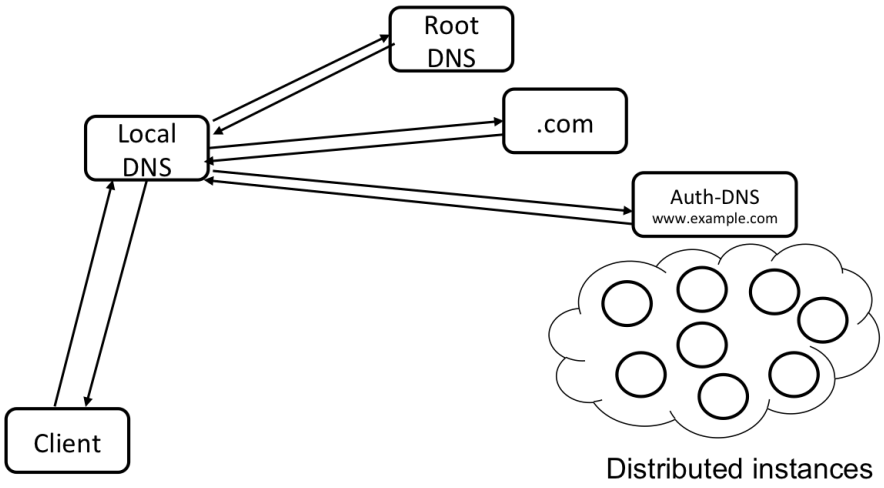

Figure 4.1: Example of a DNS architecture

4.3 Problem Statement

State of the art internet-wide SFC techniques [35–37, 77] target standard client-server connections. The client is the communication end-host starting the connection establishment. So, a SFC is established from the client, to the server, through the introduced NFs. It is worth mentioning that, in order to establish the connection, the client needs to be aware of the NFs' IP address prior to the connection establishment.

As it has happened for single-domain SFCs, we abstract a general use-case – which we consider as target use-case in this chapter – starting from technical constraints and requirements imposed by the state of the art internet-wide SFC techniques [35–37, 77]. In the most general case, a client (C) connects to her bank's website (B). B enriches its client's connection with some cloud-provided application-level NFs such as a Content Delivery Network (Service A) and a DDoS protection service (Service B). Commonly, we can assume that each NF provider deploys multiple NF instances, distributed in the network. Each NF is identified by its IP address. Given that the NF provider wants to use the best instance for the specific SFC, we can further assume that each NF service is identified by a domain name (e.g., serviceA.edu, serviceB.edu). Therefore, as shown in Figure 4.2, each NF provider implements an authoritative DNS that maps the NF domain name to the right instance's IP address according to the specific request – which is then returned to the client that can start the connections establishment.

We highlight that the SFC Resolution process shares many aspects with today's DNS, whereas it shows one important difference as shown in the following property.

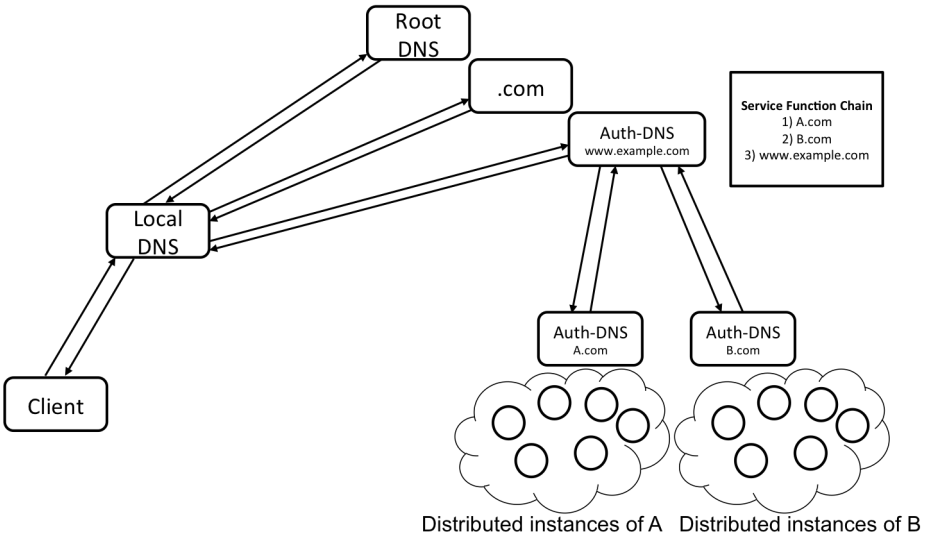

Figure 4.2: Example of a DNS architecture with SFC

SFC Resolution Property: end-to-end-through-many paradigm

The SFC Resolution process takes into account the *end-to-end-through-many* nature of the network flows. In fact, the NF instances are selected in order to optimize the entire SFC path, which originates from the client, passes through the NFs, and terminates to the server.

The previous SFC resolution property highlights the fact that today's DNS deals with standard end-to-end connections. Therefore, the domain name to resolve is usually represented by one end-host (i.e., server), and the resolution process is optimized, among other parameters, based on the other end-host (i.e., client) location in the network. However, when dealing with the NFs within a SFC, each NF's domain name has to be resolved by taking into account the location of the NF within the chain. So, each NF instance is selected based on the previous hop in the chain, as well as the next hop in the chain.

4.4 Today's DNS Resolution Strategies

When considering today's DNS, it is possible to implement two *SFC Resolution* strategies, as we examine in [78].

I – Client Selection (CS): the client sends a DNS query for each node composing the SFC (e.g., NFs and server). The serving instance, at each hop, is selected to be the closest to the client. The selection process is optimized for the client-server paradigm – the node to be resolved is the end host of the communication.

II – Hop-by-hop Selection (HS): the client uses the *EDNS0* client-subnet extension of the DNS protocol [79], which enables to specify an IP prefix in a query in order to provide the DNS server with a hint about the client's location. Such extension is used to optimize the path selection based on the previous hop. Thus, at each hop, the serving node is selected to be the closest to the node selected at the previous hop.

The previous approaches are then compared to the Global Selection (GS), in which we assume to have full visibility of all middlebox instances and their locations. Therefore, the problem is modeled as a shortest path problem. GS represents the best case. A preliminary evaluation has been performed on simulated network graphs for SFCs composed of 5 in-network services. CS and HS show end-to-end network delay +49% and +28% greater than GS, respectively. These poor results are rooted in the uncoordinated decision performed by multiple and independent parts (e.g., in-network services authoritative DNS). In fact, a serving instance, identified by its IP address, is usually selected among the available instances by the in-network service provider, according to a number of optimization criteria[7]. Since each in-network

[7] For the sake of our discussion and without loss of generality, we will assume that a serving instance

service provider selects its instance independently from the instances selected by the other providers, there is no chance to provide an instances selection that is optimized for an end-to-end global optimum [80].

4.5 Collaborative SFC Resolution

In Section 4.4 we highlighted the inefficiencies when using the current DNS architecture for the SFC Resolution process. We showed that such inefficiencies are rooted in an uncoordinated decision performed by multiple and independent parts.

In this section, we present a **Collaborative SFC Resolution** which enables to achieve close-to-optimum NF instance selection, by sharing a minimum amount of information among the different parties. We implemented a system prototype, performing minimal changes to the current DNS architecture, in order to evaluate its impact on the NF instance selection. In our evaluation in Section 4.6, we show that the proposed Collaborative SFC Resolution process enables to find solutions that show up to 30% improvement, in terms of end-to-end network delay, when compared to solutions that do not share information and perform the NF instance selection independently. Likewise, they also represent *near-optimal* solutions, as they are only 4.8% worse compared to the global optimum.

4.5.1 Enabling collaborative SFC resolution

We increase the information shared among the parties defining 2 additional sections: the *Next Service* and *Source* sections.

is selected to be the closest to the client's location. For example, this is a common approach for the selection of a service instance for a CDN-like service.

The ***Next Service*** section includes the domain name of the next service of the chain. It enables the in-network services authoritative DNS to optimize the middlebox selection considering also the next hop of the chain, whereas before it was optimized based only on the source (e.g., CS, HS).

During the design phase, also the case of including further hops ahead (e.g., 2 hops, 3 hops, etc.) has been considered. However, in order to narrow down the number of information shared among authoritative DNSs, in Section 4.6.3 we prove a single hop ahead to be the right trade-off. It improves notably the *SFC Resolution* while sharing as little information as possible.

In the most general case, a DNS answer includes one or multiple IP addresses in order to be resilient to a node failure. In fact, if the node corresponding to the first IP address returned fails, the client can use one of the following IP addresses included within the answer. In order to enhance the DNS functionalities while reducing the number of changes to the current architecture, we enrich the semantic of DNS answers while keeping the structure unchanged. In particular, we embed the selection preference in the IP addresses order. Therefore, the whole set of IP addresses received at one hop, is used to define the ***Source*** section of the next query. Thus, the selection preference embedded in the *Source* section is considered for the next hop resolution.

4.5.2 Extended Selection (ES)

Fig 4.3 shows the network abstraction. *Extended Selection (ES)* is the process of selecting the shortest path on the *SFC Network*, which is composed of the client, the in-network services, and the content server. However, the in-network services

```
;; QUESTION SECTION:
www.example.edu.                                IN     A

;;;; SOURCE SECTION:
www.previous_hop.edu.        300        IN     A   199.99.22.X
www.previous_hop.edu.        300        IN     A   199.99.23.X
www.previous_hop.edu.        300        IN     A   199.99.24.X
www.previous_hop.edu.        300        IN     A   199.99.25.X
www.previous_hop.edu.        300        IN     A   199.99.26.X

;;;; NEXT SERVICE SECTION:
www.next_hop.edu.
```

It represents the extended DNS query, sent from the content server to the in-network services, which includes the Source and Next Service section.

Listing 1: Extended DNS Query Example

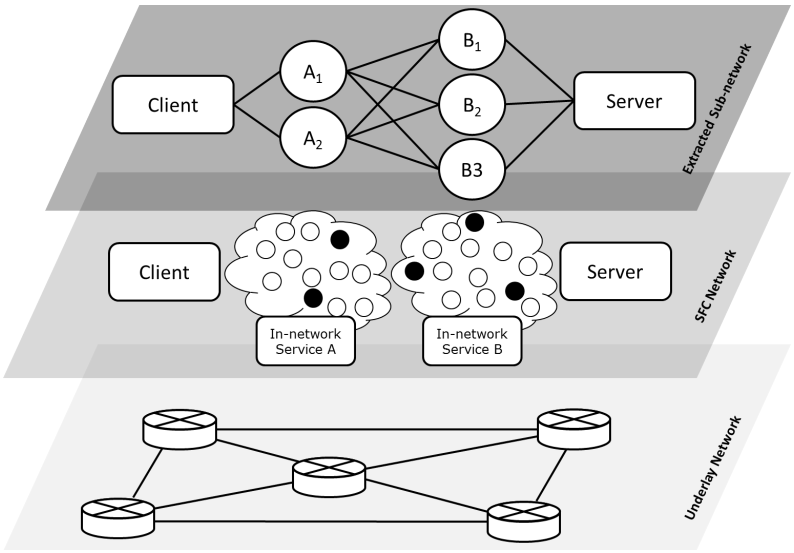

Figure 4.3: Network abstraction

are provisioned by multiple and independent domains and their distribution in the network is known only locally at each service provider. Therefore, ES aims first to extract a *Sub-network*, relevant for the specific SFC. Then, it models the *SFC Resolution* as a shortest path problem on the extracted sub-network, on which it has full visibility.

Implementation Details ES can be implemented, seamlessly, on the local DNS or on the content server authoritative DNS. However, the former requires changes to the current DNS architecture, whereas the latter requires changes only to the content server – which is the main stakeholder that benefits from an optimized middleboxes selection. Therefore, in order to foster the implementation of our solution within the today's DNS, we implement ES on the content server.

Extended Selection Phases *(I) Middlebox Discovery:* the content server uses an approach similar to HS. In particular, it resolves the whole SFC using the extended DNS query defined in Section 4.5.1. For the first service, the client is defined as the source and the second service of the chain as the *Next Service*. Therefore, the answer is used to define the next query extended *Source* section and the third service is defined as *Next Service*. And so on. In the end, the content server extracts a *Sub-network* from the *SFC Network* (Fig. 4.3).

(II) Cost Discovery: it represents the process of finding the weights of the *Extracted Sub-network*. The weights represent the network delay between any couple of nodes (i.e., middleboxes). The weights can be evaluated either with monitoring actions or through state of the art techniques such as [81].

(III) Path Selection: the content server authoritative DNS has now full visibility on the *Extracted Sub-network*. Therefore, it models the *SFC Resolution* as a shortest path problem, and uses state of the art algorithms (e.g., Dijkstra).

4.5.3 Client's DNS answer

We include the SFC related information within the answer returned to the client from the content server authoritative DNS. In particular, we define the *SFC Answer*

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25778
;; QUERY: 1, ANSWER: 1, AUTHORITY: 1, SFC ANSWER: 2, SFC AUTHORITY: 2
;; QUESTION SECTION:
www.example.edu.                                    IN    A

;; ANSWER SECTION:
www.example.edu.                    00300 IN    A     199.99.22.39

;; SFC ANSWER SECTION:
www.A.edu.                          00300 IN    A     199.99.22.37
www.B.edu.                          00300 IN    A     199.99.22.38

;; AUTHORITY SECTION:
www.example.edu.                    93375 IN    NS    ns1.www.example.edu.

;; SFC AUTHORITY SECTION:
www.A.edu.                          93375 IN    NS    ns1.a.edu.
www.B.edu.                          93375 IN    NS    ns1.b.edu.
```

Listing 2: Extended Client DNS Answer Example

and *Authority* sections. As for a standard DNS answer, the former includes the IP addresses of the in-network services composing the SFC. While the latter includes the corresponding authoritative DNSs. Moreover, we keep the original DNS answer structure unchanged. A client, that does not support the extension, is still able to connect to the content server using the IP address in the answer section.

4.6 Evaluation

In our evaluation, we consider the scenario of a SFC composed of the client, five in-network services ($N_{services} = 5$) and the content server – which we assume to be deployed in a single network location. The selected number of in-network services has been selected to reproduce a real use case. When the number of in-network services increases, some performance metrics might be severely impacted. For instance, in

[35] the Time To First Byte (TTFB) for a SFC composed of 5 services is 1500 ms, which in some cases might already be considered unacceptable. In Section 4.6.1, we evaluate the overall number of DNS Query, and in Section 4.6.2 the Resolution Time (RT) required to resolve the SFC. Thus, in Section 4.6.3, we evaluate the impact of the *Collaborative SFC Resolution* on the connection establishment (i.e., Time To First Byte). In Section 4.6.4, we consider the overhead introduced by the proposed solution in the Domain Name System (DNS) architecture. We conclude the evaluation in Section 4.6.5, comparing ES against state of the art solutions considering all the evaluated aspects.

4.6.1 Number of DNS queries

We define $N_{tot} = N_{services} + 1 = 6$ the number of nodes to be resolved. CS, HS, and ES are compared against the case of a standard *Single Domain Resolution* – which we assume as a *baseline* – evaluating the number of DNS queries sent (M).

In our evaluation, a simple caching strategy is considered. The local DNS caches Top Level Domains (TLD) such as .com, .net, .edu, etc. Whereas, the content server DNS caches the in-network services authoritative DNSs. We claim this to be a realistic assumption because the content server resolves a single service chain i.e., the one it defines.

Single Domain Resolution (SD): $M_s = 3$. The client sends a DNS query to its local DNS. The local DNS resolves the content server authoritative DNS sending a query to the TLD authoritative DNS (i.e., .edu) – using its cached value of it. Thus, it forwards the client's query to the content server authoritative DNS, which finally resolves the domain name.

Client Selection (CS): $M_{cs} = M_s * N_{Tot} = 18$. The client sends N_{Tot} queries to resolve each in-network service and the content server with a total of 18 queries.

Hop-by-hop Selection (HS): $M_{hs} = M_{cs} = 18$. HS, similarly to CS, needs to resolve N_{Tot} domain names.

Extended Selection (ES): $M_{es} = M_s + (1 * N_{services}) = 8$. M_s queries are required by the client to resolve the content server authoritative DNS. Therefore, the content server resolves $N_{services}$ in-network services. However, only a single query is needed to resolve each in-network service, as we assume the content server uses the cached values of the in-network services authoritative DNSs.

Please note that in order to enable each in-network service authoritative DNS to optimize the middlebox selection also for the next service, an additional resolution is required. However, each service of a defined SFC steers the traffic to the same service i.e., its next hop. Therefore, we assume that an important optimization can be done here and, for this reason, the mentioned additional resolution is not included in the evaluation of the overall number of DNS queries.

4.6.2 Resolution time

The *Resolution Time (RT)* is the time interval, observed by a client, to receive a DNS answer after a DNS query has been issued. Several factors condition the RT e.g., Round-Trip Time (RTT) among authoritative DNSs, the time required by each DNS to generate the answer, etc. However, most of these factors depend on the specific implementation and/or current network conditions. We perform a quantitative analysis on the RT considering the number of queries evaluated in Section 4.6.1. More in detail, we consider the number of DNS queries as a comparison

metric, as we assume that a higher number of queries corresponds to a higher RT. We can observe similar results for the *Single Domain Resolution (SD):* $RT_s = M_s = 3$. For the Client Selection (CS) instead, we can make the following consideration; each query is independent from the others enabling the client to send them in a parallel fashion. Therefore, we assume that CS, even if generates more queries than SD, shows RT similar to it: $RT_{cs} = RT_s = 3$. The same consideration cannot be applied for HS, because each query is dependent on the previous one – the result of a query is used to define the next query. Therefore, the resulting resolution time is: $RT_{hs} = M_{hs} = 18$. No further optimization can be applied also for ES: $RT_{es} = M_{es} = 8$.

4.6.3 TTFB evaluation

The Time To First Byte (TTFB) is the time required for a client to receive the first byte of a content server's response, consequent to its issued request. We evaluate the impact of different *SFC Resolution* strategies on the TTFB. We assume that TCP is used to establish the connection throughout the SFC. Assuming the service chain end-to-end delay (D), TTFB is proportional to it by a factor four (TTFB = 4D) [82].

The *SFC Network* (Fig. 4.3) is composed of the client, the in-network services' middleboxes, and the content server. It represents an *Overlay Network* over the *Underlay Network*. The *SFC Network* is not a full mesh. The client can select one of the first service's deployed middleboxes. Thus, the first service can select one of the second service's deployed middleboxes. And so on. As follows, assuming the links in the *Underlay Network* are symmetric, we can model the *SFC Network* as a non-complete Directed Acyclic Graph (DAG).

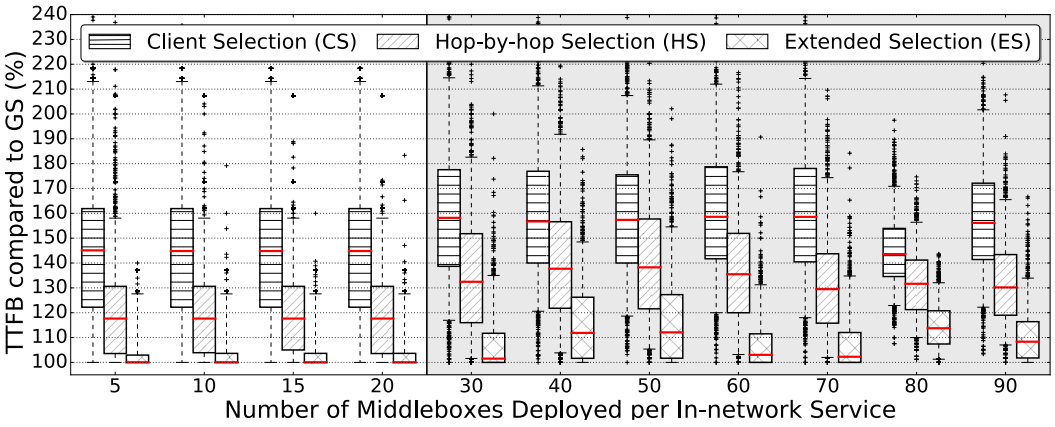

Figure 4.4: **Time To First Byte (TTFB) Graph. SFC Composed of 5 In-network Services** The left and white part represents the experiments performed on Rocket Fuel topologies. The right and light grey part represents the experiments performed on simulated network graphs.

Experiment Global Selection (GS) is considered as a comparison metric for our experiment. It represents the ideal case because it assumes to have full visibility on the *SFC Network*, which enables to apply a shortest path algorithm (e.g., Dijkstra). We generated underlying network graphs using Rocket Fuel [81] topologies, which also provide per-link delays. We assume each node in the underlying network to be a potential location for placing an instance of the in-network services (e.g., middlebox). In total, we generate 61 underlying networks with a number of nodes varying between 27 and 115. Then, we generated different *SFC Networks* (overlay network) with a number of nodes varying from 27 (i.e., 5 middleboxes per service) to 82 nodes (i.e., 20 middleboxes per service). An increased number of middleboxes deployed represents a wider distribution of the in-network services in the network. The mapping of the SFC network to the underlying network is performed placing each middlebox on a random assigned underlying network's node. For each underlying network, we performed 100 different placements, resulting in 100 different SFC networks. In total, we generated 24400 SFC networks. The distance between two middlebox on

the overlay network, which are not adjacent in the underlying network, is evaluated as the sum of link's delays on the shortest path between such nodes. We implemented *CS*, *HS* and *ES*, while we use the *NetworkX Dijkstra* implementation for *GS*.

Scalability In order to evaluate the impact of a wider distribution of the services (more than 20 middleboxes per service), we performed further experiments on simulated network graphs generated using Python. We generated SFC Networks with a number of nodes varying from 152 to 452 (from 30 to 90 middleboxes per service). Each node is placed randomly on a 100x100 coordinate grid, and the Euclidean distance between any two nodes is considered as network delay. For each experiment, we generated 1000 different SFC Networks and all the algorithms are evaluated on the same graph.

Results The experiments results are shown in the box plot of Fig. 4.4. The left and white part represents the experiments performed on Rocket Fuel topologies. Whereas, the right and light grey part represents the ones performed on simulated network graphs. The x-axis represents the number of middleboxes deployed by each in-network service in the network. The y-axis represents the TTFB (in %) normalized to GS.

Observing Fig. 4.4 it is possible to make the following observations. *(I):* the experiments performed on Rocket Fuel topologies and simulated network graphs show results that are coherent with each other. *(II):* For a number of middleboxes deployed per service between 5 and 20, ES shows a median equal to 0 i.e., 50% of the samples show end-to-end network delays which are comparable with GS, even with a partial visibility on the SFC network graph. *(III):* CS, HS, and ES – considering the average of the median – show end-to-end network costs increase of +51.7%, +27.7%

	CS	HS	ES
SFC IPs	✗	✗	✓
RT	3	18	8
# DNS Query	18	18	8
TTFB	+51.7%	+27.7%	+4.8%
Local DNS	$\Theta(N_{ser}+1)$	$\Theta(N_{ser}+1)$	$\Theta(1)$
Server DNS	$\Theta(1)$	$\Theta(1)$	$\Theta(N_{ser}+1)$

Table 4.1: Results at a glance
The table shows an overview of the evaluation section which compares all the approaches considering all the results

and +4.8%, respectively, when compared to GS.

4.6.4 Complexity analysis

Each in-network service authoritative DNS selects the serving node based on a set of source locations and the next hop of the service chain, whereas before the selection was based only on the query's source location. We claim this increase in complexity to be negligible. The content server instead, has the overhead to run ES – which is composed of 3 phases. The *Middlebox Discovery* overhead is evaluated in Section 4.6.1. The *Cost Discovery* overhead instead, can be performed off-line and therefore it does not have a direct impact on the resolution process. The *Path Selection* is modeled as a single-source single-destination shortest path problem (i.e., from the client to the server) on a non-complete, *Directed Acyclic Graph (DAG)* i.e., *Extracted Sub-Network* (Fig. 4.3). Therefore, it can be efficiently solved using topological sort with a resulting time complexity of $O(|V|+|E|)$ [83].

4.6.5 Results at a glance

Table 4.1 shows a comparison of all the proposed approaches. Please note that CS and HS are used as a comparison metric to evaluate the impact of SFC Resolution on the system. However, they don't represent solutions which are ready-to-deploy, because they assume the client knowledge of the SFC – which would require an additional *handshake phase* – prior to the resolution process. Likewise, GS is only used as a comparison metric of the global optimum.

End-to-end Network Delay CS, HS and ES show end-to-end network delay +51,7%, +27.8% and +4.8%, respectively. However, consider that those are relative values, compared to GS, evaluated assuming TCP is used to establish the connection. Therefore, end-to-end delay increases have a different impact on different protocols. For instance, when establishing an encrypted connection using TLS, additional RTTs, on top of the TCP handshake, are required to establish the secure connection. Moreover, we would like to emphasize that increased network delay might severely impact user experience and service providers' revenue. For instance, injecting just 400 ms of artificial delay into Google search results caused the delayed users to perform 0,74% fewer searches after 4-6 weeks [84].

Impact on the System Local DNSs are affected by the proposed solution as they are required to forward the *extended client answer* received from the content servers. Further changes are required on the content server and the in-network services, which need to support ES and the extended DNS queries, respectively. However, we claim this to be acceptable trade-off since only the stakeholders that benefit from an optimized middleboxes selection are affected.

Impact on the Scalability While CS and HS do not require changes to the DNS architecture, they generate an increased workload on the local DNSs ((i.e., $\Theta(N_{ser}+1)$)) which harms the current DNS scalability. On the other hand, ES requires almost no changes to the DNS infrastructure, and it does not introduce overhead on any level of the DNS infrastructure (i.e., $\Theta(1)$), because it distributes the increased overhead on the content server authoritative DNS. We claim that this aspect to be an important factor while incrementally introducing our approach in today's Internet.

4.7 Related Work & Use Cases

In the context of the Domain Name System, DNS redirection [85] is a state of the art technique adopted by service providers to redirect the clients' requests to the closest CDN node. However, a CDN node acts as a surrogate server and it does not define a service chain. IP anycast [86] instead, is used to route the traffic to the nearest node in a group of identical resources. For instance, it is used to route the Google DNS (e.g., 8.8.8.8) traffic to the nearest instance from the client. However, also in this case, IP anycast is optimized for the client-server paradigm and cannot be adapted for the context of a service chain composed of multiple nodes.

Dysco is a *session-level protocol* that steers the traffic of a TCP session through a SFC [37]. It can dynamically re-configure the SFC without requiring any changes to TCP stacks, IP routing, and middleboxes. However, it requires the middleboxes' IP address to establish the TCP connections.

Multi-context TLS (mcTLS) is a pluggable TLS extension [35]. It enables a client and a server to introduce a set of in-network services within TLS connections.

A handshake phase is used to agree on the SFC – defined by the client – and distribute different encryption keys to the communication nodes. However, the server cannot introduce in-network services. It can either accept the SFC defined by the client or refuse it. Moreover, it assumes that the client has the list of middleboxes – defined by their IP addresses – prior to initiating a handshake (see § 6.1 in [35]).

Segment Routing (SR) is a source routing architecture developed within the IETF [36]. It allows packets to follow non-shortest paths towards the destination by specifying a list of waypoints defined as *segments*. The list of segments is transported in a new type of Routing Extension Header called *SR Header (SRH)* [87]. In [88] the authors proposed to leverage SFC using the IPv6 version of Segment Routing (SR-IPv6). Only the nodes that actually process SR-enabled packets need to support SR-IPv6. Whereas intermediate nodes, that are on the path of a segment, forward the packets as regular IPv6 packets. However, their solution is limited to network functions which are SR-aware. In [89] the authors propose to leverage SFC using SR-IPv6 also for network functions which are unaware of SR. However, SR-IPv6 assumes that the client, prior to the connection establishment, is aware of the list of segments (i.e., IPv6 addresses).

Middleboxes selection itself is an orthogonal problem compared to SFC traffic steering. However, it represents the groundwork that can foster the wide-spread implementation of such techniques in today's Internet. The proposed solution provides the means to resolve the middleboxes' IP addresses, related to a specific SFC, that can be used for example to define the *mcTLS handshake* or the *SR Header (SRH)*.

4.8 Conclusion & Future Work

In this chapter, we addressed the *SFC Resolution* problem in the context of *Internet-wide SFCs*. We sketched a *Collaborative SFC Resolution* that enables to share a minimum amount of information among multiple and independent in-network service providers. We implemented a prototype using the current Domain Name System (DNS) infrastructure. Through an extensive evaluation on real network topologies and simulated network graphs, our solution, on average, shows 30 % improvement, in terms of end-to-end network delay, when compared to solutions that do not share any information and perform the NF instance selection independently. At the same time, they represent *near-optimal* solutions, as they show only 4.8 % higher end-to-end network delay, on average, when compared to the global optimum.

The proposed *SFC Resolution*, applied using the DNS, aims to provide the basis and to foster the wide deployment of Internet-wide SFC steering techniques that use plain IP routing for steering the traffic through the SFC in today's Internet.

As future work, we plan to take into account a case in which client and server can actively negotiate the list of services to introduce within the end-to-end path. Therefore, an efficient *negotiation phase* needs to be designed.

Chapter 5

Mute: MUlti-Tier Edge networks

5.1 Introduction

In recent years, several application use-cases, requiring high data availability and quick computation, such as Internet-of-Things (IoT), vehicular networks, etc. have proliferated to a great extent. Such applications require computational resources that can handle highly variable data with stringent completion time requirements. The traditional centralized cloud model is unable to support these use cases due to possibly high network delays encountered while offloading data to the location of cloud data centers. Researchers have proposed decoupling the traditional cloud model to several smaller computation resources installed closer to data generators [90]. Due to their proximity to the network "edge", these collections of resources are termed as Edge cloud [91].

In the past, several edge cloud models have been proposed [92–95] to decouple network delay from computation time in concrete deployments. Telecom operators have also adopted such models, for instance, models such as Mobile Edge Computing (MEC) enable edge servers and cellular base stations to be operated simultaneously [96]. Recently, significant improvements in MEC have enabled Mobile Network Operator (MNO) to integrate 5G telecommunication in the cloud platform itself [91]. However, at this stage, only proprietary edge services can be deployed by MNOs MEC instances, and the platforms are not open to third-party providers.

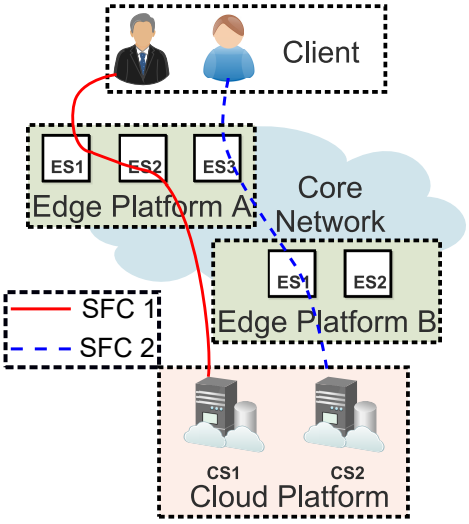

Figure 5.1: SFCs over Edge Cloud

Other research proposals have considered open-to-all edge cloud instances to be composed of possible community driven compute resources which drastically increase the density and variability of the edge. For instance, Mohan et al. [97] present a model where the edge cloud is composed of a combination of voluntary compute resources such as mobile phones, workstations, etc. and managed micro-cloud instances such as mini-datacenters.

Existing service/task placement and resource selection algorithms attempt to map multiple services on a set of homogeneous cloud resources with a consistent network delay from the clients [98]. Figure. 5.1 shows an end-to-end SFC deployment on the edge. However, the significant heterogeneity of the edge resources, regarding processing capability and network distribution, necessitates re-designing such placement algorithms, to make them match better with edge computing environments.

In this chapter, we provide the following contributions.

1) We define a use case scenario for the Edge where Edge Platform Providers open their infrastructure to third-party Service Providers as shown in Figure 5.2. Building on the success of other open systems, with the Internet and Web being prime examples, we conjecture that a similarly open approach will enable edge computing to flourish. Existing solutions, such as auctioning strategies [99] can be used to allow platform providers to run their systems for profit.

2) We define a model of a *multi-tier edge network* in which edge resources are logically clustered into distinct tiers, based on their characteristics such as processing capabilities, network delay from the client, etc. as shown in Figure 5.2. This characterization enables Edge Service Providers, to efficiently manage their governed edge resources, and to find more granular solutions optimized for multi-tier scenarios.

3) We design *Mute*, a placement algorithm which leverages multi-tier edge architecture to find an edge server which best supports the needs of a requested service.

4) We perform an extensive set of simulations using real network topologies [100]. We show that the *Mute* algorithm achieves 66% reduction in network cost, when compared to state of the art non-edge aware placement algorithms. Additionally, *Mute* leverages the multi-tier structure to achieve the service placement up to 50% faster, when compared to non-tier aware placement algorithms.

5.1.1 Contributions

The rest of the chapter is organized as follows. We discuss the architecture and stakeholders in an edge network in Section 5.2, and present Multi-Tier Edge architecture and *Mute* algorithm in Section 5.3. In Section 5.4, we evaluate *Mute* with state of the art algorithms on realistic topologies. In Section 5.5, we present related

work and use-cases of proposed edge architecture. We conclude in Section 5.6.

5.2 Architecture & Stakeholders

In Figure 5.1, we show the architecture and the stakeholders of an edge cloud. We envision a model where several edge platforms co-exist on the network. An end-to-end connection is established between the client and the cloud platform. Multiple services can be deployed on the edge servers which can enrich client's connection with the cloud (e.g., video transcoder, web proxy, etc.). The resulting *Service Function Chain (SFC)* is routed through deployed services which can either be placed on servers co-located in the same facility or in different platforms and location. Based on the ownership of resources, the edge cloud model has three primary stakeholders: Clients, Edge Service Providerss (ESPs) and Cloud Service Providerss (CSPs) (shown in Figure 5.2).

Clients establish the connection with a Cloud Server (CS) in a Cloud Platform (CP). The clients and/or cloud can request to include virtualized services deployed at the edge via SFC. The client can have either an *active* or a *passive* role in the SFC traffic steering depending if the SFC is transparently enforced [14], for instance by the network operator, or is explicitly defined by the client or the server [35, 37].

Cloud Service Providers (CSPs) consolidates several CSs deployed on a CP. Individual CSs are grouped together in Data Centers (DCs) facilities, which are distributed at various locations in the network [101]. Upon arrival of a connection request from a client, the CSP re-routes the request to one of the available CSs in the CP.

Edge Service Providers (ESPs) act as an intermediate entity between clients and CSPs. An ESP hosts several computation and storage capable *Edge Servers (ES)* federated into *Edge Data Centers (Edge DCs)*. Unlike the cloud DCs, the Edge DCs are spread over a broader geographic region and have a significantly lower latency to connect to a client in proximate location.

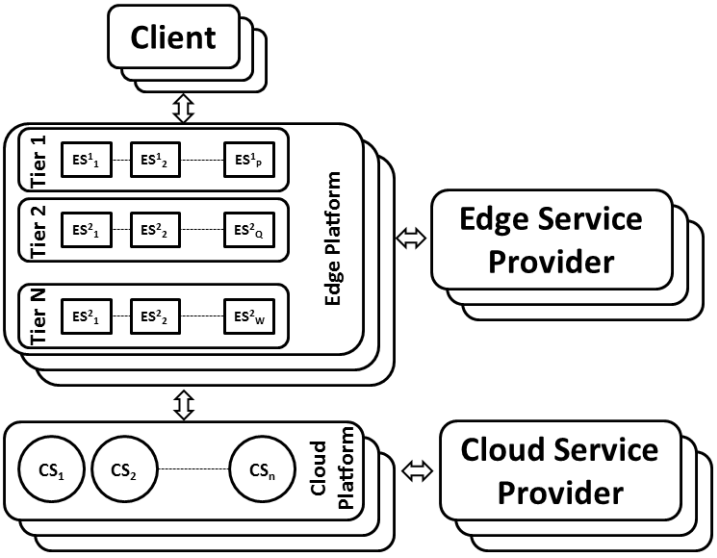

Figure 5.2: Multi-tier Edge architecture & stakeholders

5.3 Edge Platform Modelling and Deployment

The edge cloud model discussed in Section 5.2 considers the interaction between the involved stakeholders. In this section, we discuss the techniques by which the ESPs can model its governed Edge Platform and select the best ES to deploy requested service functions.

5.3.1 Multi-Tier Edge

Past state of the art research has proposed grouping ESs in an EP wherein all servers have similar characteristics [94]. The ESP can then employ a selection algorithm which "picks" the best ES capable of supporting the processing and number of users[8] required by the requested service.

We propose *multi-tier edge* platform, which further categorizes resources within a platform in *tiers* depending on their network delay from the edge. Fig. 5.2 illustrates platform's architecture. The properties of the ESs in resulting architecture varies from tier-to-tier.

I) Network Delay: the network delay to the edge is the primary attribute for classifying ES in tiers. The tiers which are closer to the network edge are composed of ESs with lower network delay to clients than higher tiers. However, the network delay of ES to end-client is location-dependent; e.g. Tier 1 ESs in New York and California will have the least network delay to clients located in their proximate locations but will have a very high delay for the other's location.

II) Bandwidth and Processing Power: edge servers in lower tiers have limited processing capability and can only support a restricted number of users simultaneously. However, their proximity to the network edge makes them desirable to deploy virtualized services. Tiers closer to the cloud are composed of ESs with higher processing capability.

III) Number of Servers: lower tiers are characterized by a greater number of small edge servers, that show limited resource capability. On the other hand, tiers

[8] We denote the number of users that can be supported by a server as its *bandwidth* throughout the paper.

which are closer to the cloud, have a small number of more powerful servers.

5.3.2 Network Structure & Model Definition

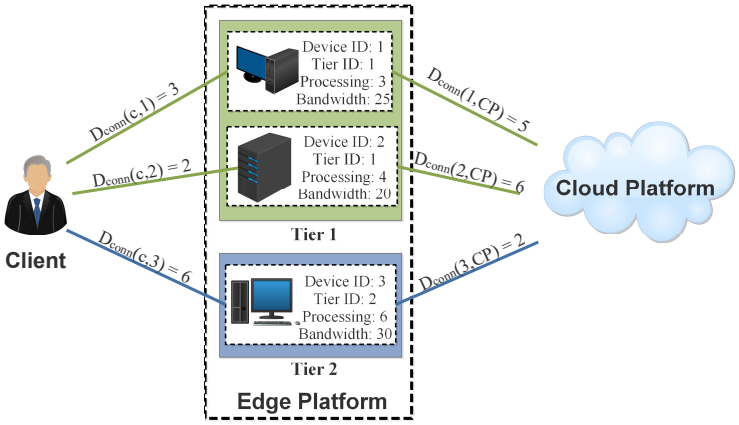

Figure 5.3: Multi-Tier Edge network architecture example

Here we formulate the problem of placing services on an Edge Platform (EP). Figure 5.3 shows the network architecture of a multi-tier Edge Platform composed of three ES with different bandwidth and processing capability. Each server has a network connection to the client and the cloud platform of a certain weight. The EP is composed of multiple Edge Servers (ES) of different attributes, i.e. $EST = ES_1, \ldots, ES_r$ where EST denotes set of all ES physical machines. As discussed in Figure 5.3.1, the Edge Service Provider (ESP) categorize ES into tiers $t = t_1, \ldots, t_n$ wherein each ES in $EST \in t$. The resulting ES are grouped as:

$$ES^n = \{ES^n_q, ES^n_{q+1}, ES^n_{q+2} \ldots, ES^n_r\} \quad \text{Edge Servers Tier } n$$

where

$$EST = ES^1 \cup ES^2 \cup \ldots \cup ES^n$$

The Edge tier ES^1 lies closest to clients $C = C_1, C_2, ..., C_m$ and the last tier ES^n is closer to the end server/cloud. Each ES physical machine has a maximum resource utilization denoted by $ES^{proc} = ES^p_1, ES^p_2, \ldots, ES^p_r$ and maximum bandwidth as $ES^{bw} = ES^{bw}_1, ES^{bw}_2, \ldots, ES^{bw}_r$. As shown in Figure 5.1, ES in lower tiers have smaller processing and bandwidth capability but also have a significantly lower network cost to clients.

5.3.3 Placing Services on the Edge

Let $S = S_1 \ldots, S_m$ denote the set of $'m'$ services to be placed on EST. The service providers want to enrich the experience of their clients and overall end-to-end connections by deploying services on ES closest to the client. We define services to have different processing requirements (in terms of the number of CPU-cycles) required for their execution denoted as $S^{proc} = S^p_1, S^p_2, \ldots, S^p_m$. Similarly, according to the Service Level Agreement (SLA), each service must support a bandwidth quota denoted as $S^{bw} = S^{bw}_1, S^{bw}_2, \ldots, S^{bw}_m$.

Furthermore, every Edge server ES has an associated cost for deploying service S_j on a Physical Machine (PM) ES_i per unit time denoted as c_{ij}. The cost is dependent on ES's capability (regarding processing and networking) for running the service and the tier-level it belongs.

$$EST^c = \{ES^c_1, ES^c_2, \ldots, ES^c_r\} \quad ES \text{ deployment cost}$$

The resulting pricing model assigns more cost to low tier ESs due to their limited processing capabilities and lower network delays.

We denote variable x to indicate whether a service S_i is deployed on the edge

server EST_j.

$$x_{ij} = \begin{cases} 1, & \text{if } S_i \text{ deployed on } ES_j. \\ 0, & \text{otherwise.} \end{cases}$$

Considering the placement requirements, the deployment algorithm can optimize the following attributes, i) the networking delay between client and the service, and ii) operational cost of deploying a service on ES with certain processing capability.

5.3.3.1 Minimizing Operational Cost

As discussed in Section 5.3.1, a low tier ES is likely to have a higher cost of deployment compared to a higher tier server. The operational cost of deploying service S on the Edge Platform EST can be formulated as

$$C(\lambda) = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{S_i^{proc}}{ES_j^{proc}} ES_j^c x_{ij} \tag{5.3.1}$$

subject to

$$0 < \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{S_i^{proc}}{ES_j^{proc}} < 1 \tag{5.3.2}$$

$$\sum_{j}^{n} x_{ij} = 1 \qquad \forall ii \in S = S_1, \ldots, S_m \tag{5.3.3}$$

The deployment algorithm minimizes Equation 5.3.1 to optimize total cost of processing a virtualized service on an ES. The constraint in Equation 5.3.2 ensures

that the ES has enough processing capability to host the requested. Equation 5.3.3 guarantees deployment of all requested services.

5.3.3.2 Minimizing Network Delay

Considering that the requested service is to be deployed on an ES member of an EP (shown in Figure 5.3), the resulting end-to-end connection between client and cloud will be composed of the nodes client, ES and cloud. We denote network link between client to Edge Server (ES_i) hosting the service and ES to the cloud as denoted as $d_{conn}(c, ES_i)$ and $d_{conn}(ES_i, cloud)$ respectively. The end-to-end network cost can be denoted as

$$N(S_j) = \sum_{i}^{m} [d_{conn}(c, ES_i) + d_{conn}(ES_{i-1}, ES_i) + \tag{5.3.4}$$

$$d_{conn}(ES_i, cloud)]\, x_{ij} \tag{5.3.5}$$

subject to

$$\sum_{i}^{m} S_i^{bw} x_{ij} \leq ES_j^{bw} \qquad \forall j \in ES = ES_1, \ldots, ES_n \tag{5.3.6}$$

$$\sum_{j}^{n} x_{ij} = 1 \qquad \forall i \in S = S_1, \ldots, S_m \tag{5.3.7}$$

The network optimizing deployment algorithm minimizes network cost presented in Equation 5.3.5. The constraint in Equation 5.3.6 ensures that the selected ES is

able to support the bandwidth by the service.

The algorithm can be further modified to minimize only the network cost between client and the Edge Server. i.e.

$$N(S_j^{edge}) = \sum_{i}^{m} (d_{conn}(c, ES_i))\, x_{ij} \tag{5.3.8}$$

5.3.4 Tier-based Optimization

The processing and network optimizing algorithms iterate over the entire search space (read Edge Platform) to find the ES satisfying the requirements. As discussed in Section 5.1, an EP can be composed of hundreds of ESs. The algorithms discussed above imposes significantly large compute time to find the optimal solution.

In Algorithm 1 we present the pseudo-code of *Mute*. It exploits the multi-tier edge to find a server with optimal network cost to edge and near-optimal processing cost.

The algorithm exploits the network cost trend of the tiers, i.e., the Edge servers in lower tiers have a lower network cost to clients than higher tiers. The ESP associates EST_{tier}^{bw} and EST_{tier}^{proc} with each tier, which denotes the maximum supported bandwidth and maximum processing capability of the tier respectively. The algorithm approximates the location of the ideal ES by utilizing tier parameters and prioritizes lower tiers for placement. It further iteratively searches for ES only in the tier whose processing and bandwidth best satisfy the requirements imposed by the service.

Algorithm 1: Mute

Input :
`Total number of tiers` `tiers`$\in \{\texttt{tier}_1, \ldots, \texttt{tier}_n\}$
Available Edge servers in tiers
$\texttt{EST} \in \{\texttt{EST}_{tier1} \cup \ldots \cup \texttt{EST}_{tiern}\}$
$EST^{bw} \in \{\texttt{EST}^{bw}_{tier1}, \ldots, \texttt{EST}^{bw}_{tiern}\}$
$EST^{proc} \in \{\texttt{EST}^{proc}_{tier1}, \ldots, \texttt{EST}^{proc}_{tiern}\}$

Initialize:
`selectedServer` $\leftarrow$ `NONE`

```
1  //This function returns Edge Server ES for deploying Service S for each
     t ∈ tiers do
2    if EST_t^bw ≥ S^bw and EST_t^proc ≥ S^proc then
3      lowestNetworkCost ← ∞
4      for each ES ∈ EST_t do
5        if ES^proc ≥ S^bw and EST_t^proc ≥ S^proc then
6          Optimize network cost as equation 5.3.8
7          if networkCost ≤ lowestNetworkCost then
8            lowestNetworkCost ← networkCost
9            selectedServer ← ES
10         end
11       end
12     end
13     if selectedServer ≠ NONE then
14       break
15     end
16   end
17 end
```

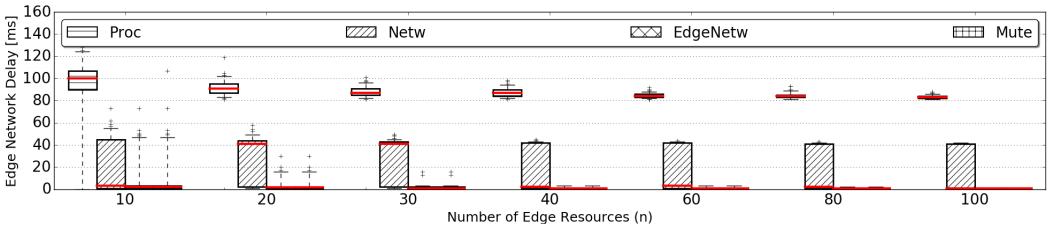

Figure 5.4: Network cost comparison

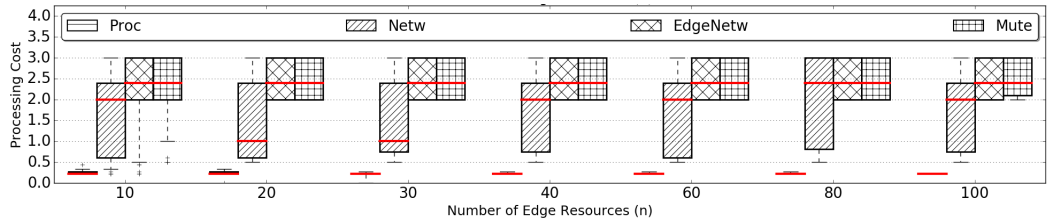

Figure 5.5: Processing cost comparison

Figure 5.6: Service deployment cost comparison between *Proc*, *Netw*, *EdgeNetw* and *Mute* algorithms

5.4 Evaluation

In this section, we analyze the performance achieved by *Mute* and compare it with the *Netw*, *Proc*, and *EdgeNetw*. *Netw* and *Proc* are state of the art placement algorithms, applied on edge networks whereas *EdgeNetw* is an iterative-variant of *Mute*.

I) Network Optimizing Server Selection (*Netw*) iteratively searches for ES with least associated network cost of deployment (as modeled in Equation 5.3.5) in an Edge Platform.

II) Processing Optimizing Server Section (*Proc*) selects ES with least processing cost of service deployment (as modeled in Equation 5.3.1).

III) Edge-Network Optimizing Server Selection (*EdgeNetw*), similarly to *Mute*, selects the server with least network cost to the client (as modeled in Equation 5.3.8). However, unlike *Mute*, *EdgeNetw* is not aware of the multi-tier structure of the edge network and iteratively searches for the optimal server in the search space.

5.4.1 Experiment Setup

We implemented *Mute* and the selection algorithms discussed above in a custom Python-based simulator. The simulator considers Edge network graphs based on Rocket Fuel topologies [100] which also provides per-link delays between graph nodes. Overall, we generated 61 network graphs with $\approx 25-115$ nodes. Edge networks are generated by assigning Edge Servers (e.g., from 10 to 100) on the generated network graphs, which we assume as underlying network topology. The network cost between any two ESs is defined as the sum of the link's delay on the shortest path between such nodes in the underlying network. We provide processing and bandwidth capabilities to each ES in the edge network and cluster them in tiers, as discussed in Section 5.3.1. For each network graph, we perform 100 placements, resulting in 100 different edge networks. Therefore, we generate ≈ 6000 edge networks throughout the experiments. For the sake of simplicity, we only consider placing a single service on the network in the current evaluation and leave the analysis of multiple services as future work.

5.4.2 Results

Figure 5.4 and Figure 5.5 present the box plot results of network and processing cost respectively and capture data distribution throughout all our experiments. The top and the bottom of the boxes represent the first and third quartile respectively, and the red waist represents the median.

Proc, as expected, performs the best in terms of processing cost but has the worst performance for network cost. It does not represent a practical solution as it always selects an ES on tiers that show the least processing cost. ESs within such tiers have significantly higher network delay from the edge and thus show the worst networking cost.

Netw performs significantly better than *Proc* and selects an ES with much lower networking cost. However, it performs worse than *EdgeNetw* and *Mute*. *Netw* selects an ES with the least end-to-end delay which encompasses the delay from the client to the ES and the delay from the ES to the cloud. The algorithm does not optimize edge placement as it is unable to make a distinction between servers with similar end-to-end network cost but significantly different path delays between client to the ES. Considering the median of the experiment results, *EdgeNetw* and *Mute* (which show similar results) achieve 66% reduction in network cost on average when compared to *Netw*. However, as discussed in Figure 5.3.1, ESs in lower tiers are not processing capable which reflects in the processing cost achieved by both the algorithms. Both *EdgeNetw* and *Mute* show an increase of 20% in associated processing cost, on average, when compared to *Netw*.

Additionally, we analyze the average time required by each algorithm to complete the service placement, the results of which are shown in Figure 5.7. As evident

from the figure, *Mute* completes its placement in ≈ 50% lesser time (37.5% average reduction) when compared to the other algorithms. *Mute*, unlike the other algorithms, searches over a significantly reduced problem space as it utilizes EST^{bw} and EST^{proc} to estimate the tier which hosts the optimal ES. Therefore, the proposed *Mute* algorithm can discover the ES which can support the requirements imposed by Service Provider while ensuring least network delay to the client. Furthermore, it efficiently utilizes the multi-tier architecture of an Edge Platform to achieve a deployment time significantly lower than the state of the art.

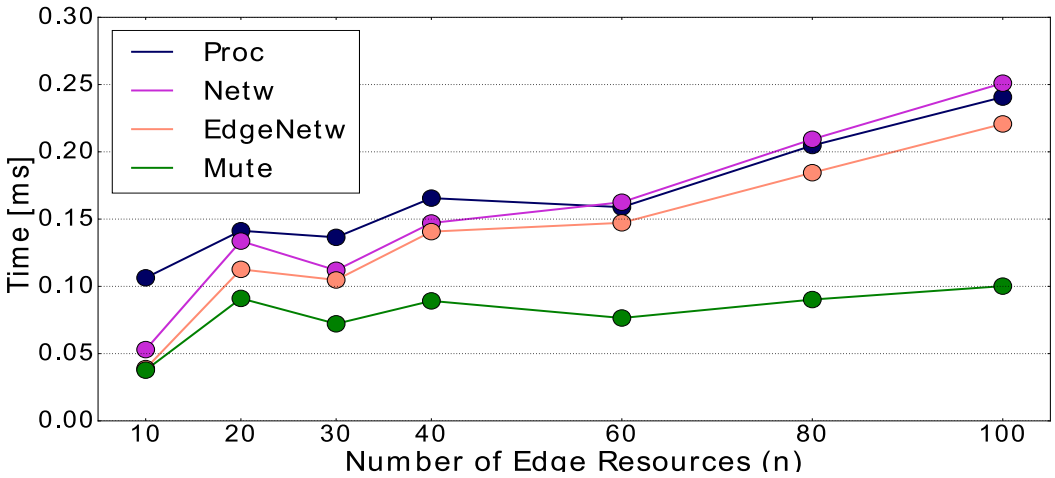

Figure 5.7: Time complexity graph

5.5 Related Work & Use-Cases

An increasing number of application use-cases such as Internet-of-Things (IoT), industrial automation, Augmented Reality (AR), smart cities, autonomous transportation systems, etc., are warranting the need for edge compute clouds [102–104]. Several other research areas, such as Industry 4.0 utilize edge clouds to provide

efficient solutions to several open questions. For example, researchers have proposed automated collaborative robots which require time-critical processing with extremely low latency (in order of milliseconds) to create a safety zone for their operators [92, 105]. Augmented Reality glasses can assist operators in a continuously varying production environment by performing markerless object recognition and accurate tracking in a factory. However, such use-cases can only be fulfilled if the required data is cached and computed at closely located servers.

State of the art solutions in the fields of virtualization, Software Defined Networking (SDN) and Network Function Virtualization (NFV) represents key technologies to deploy virtualized services at the very edge of the network in a flexible way and on cheap commodity hardware [91]. In this paper, we envisioned the case in which clients have control over which services will be included in their network path using state of the art Service Function Chaining (SFC) techniques such as [35, 37]. In an open market of choosing services, the client can discover specific edge servers which are hosting the required service by utilizing Domain Name System (DNS) based techniques [78, 106]. The clients can significantly enhance their connectivity with the end-server by using services with very low network delay.

5.6 Conclusion

In this chapter, we proposed *Mute*, a multi-tier edge cloud architecture which enables edge cloud providers to efficiently deploy services at the edge. *Mute* categorizes edge servers into groups based on their network delay from the client. Due to its unique architecture abstraction, *Mute* can efficiently deploy service function chains on edge servers across multiple edge platforms. Through our extensive simulation-

based evaluation on RocketFuel topologies, we show that *Mute* achieves a significant reduction in edge network delay and completion time when compared to state of the art.

In our future work, we plan to investigate the impact of different edge resources clustering strategies on the service placement.

Chapter 6

Conclusion & Future Work

In this thesis, we explored the solutions available in the state of the art with regards to SFC systems, analyzing their characteristics and limitation. Along these lines, we introduced a high-level categorization of SFC systems based on their requirements and target use-case: *single-domain* and *internet-wide* SFCs. The former targets SFC systems typically deployed within a single administrative domain, such as Mobile Networks and Data Centers, which are characterized by the fact that the SFC is defined and enforced by the same stakeholder. The latter instead, are characterized by the fact that multiple stakeholders are included in the SFC provisioning. Such logical separation helps in finding solutions which are optimized for the specific use-case requirements and specifications.

We observed that single-domain SFC solutions were characterized by a "clean-state" approach, when considering the SFC system design and implementation. In fact, they do not consider prior system architecture and network equipment. In Chapter 3, we propose *Catenae*, a ready-to-deploy SFC solution that can be deployed on legacy Mobile Networks infrastructure. It provides an effective SFC system without requiring important changes in the network infrastructure. Further, Catenae scales to provide fine grained policies for millions of network flows. We proved it can handle three times the expected traffic rate at the PGW of 2019 (5 years later the publication).

In Chapter 4, we explored the state of the art for internet-wide SFC solutions,

highlighting their main characteristics and requirements. We identified an important limitation, shared among the state of the art solutions. They assume that the clients have the knowledge of the IP addresses of NFs composing the SFC prior to the connection establishment. However, no prior work is available in the state of the art regarding how it is possible to retrieve those information, and which are the challenges in such process in the current Internet scenario. Therefore, in Section 4.4, we proposed to use the current Domain Name System (DNS) to retrieve the IP addresses related to a SFC, highlighting its properties and inefficiencies. In Section 4.4, we proposed a collaborative SFC Resolution approach. It requires minimal changes to the current DNS architecture, enabling to achieve close-to-optimum NF instance selection. The presented solution, implemented using a wide deployed system such as the DNS, represents the basis to foster the wide deployment of internet-wide SFC techniques. The presented system has been submitted as a Patent application [16] and is currently under review.

The DNS is a well established Internet service, however, at the same time, it is always evolving and optimizing itself as its role is crucial within the Internet architecture. In fact, the DNS resolution time affects the connection establishment time for all the web traffic. Further, it has been proved that the DNS traffic accounts for more than 50% of the overall web traffic [107]. Therefore, as future work, we envision to deploy an efficient implementation of the collaborative SFC Resolution presented in Section 4.4.

New application scenarios such as Internet-of-Things (IoT), vehicular networks, etc. have proliferated to a great extent in the Internet scenario. Such applications are requiring to offload computational resources with stringent completion time requirements. This is the case of devices not equipped with enough computation power on-board (e.g., sensors, electrical appliance, etc.) or when the task to per-

form is too computational intensive (e.g., autonomous driving). In either cases, the traditional cloud model might fail to support those use cases due to possibly high network delays encountered while offloading data to the location of cloud data centers. Therefore, traditional cloud resources are being decoupled in smaller resources, deployed closer to the users, Due to their proximity to the network "edge", these collections of resources are termed as Edge cloud.

We investigated the state of the art with regards to edge computing solutions. SFC and Edge Computing are related by the fact that a SFC is usually implemented from the client (e.g., IoT device, car, etc.), through an edge deployed service, to the cloud deployed server. However, the state of the art with regards to the service placement on the edge is very limited, as it is only consider the case in which there are homogeneous edge resources distributed in few location in the network. In Chapter 5, we proposed *Mute*, a multi-tier edge cloud architecture in which several heterogeneous edge resources are widely distributed in the network. Mute enables edge cloud providers to efficiently deploy services on such network architecture, categorizing edge servers into groups, based on their characteristics (e.g., network delay from the client, resource availability, etc.). Due to its unique architecture abstraction, Mute achieves a significant reduction in edge network delay and completion time when compared to state-of- the-art solutions, when applied to this infrastructure. In the context of multi-tier edge networks, we plan to investigate the impact of different edge resources clustering strategies on the service placement.

6.1 Thesis impact

Scientific Publications The work on designing, implementing and evaluating *Catenae* has been published in the following peer-reviewed international conference proceeding.

- Roberto Bifulco, Anton Matsiuk, and ***Alessio Silvestro***. "Ready-to-deploy service function chaining for mobile networks." NetSoft Conference and Workshops (NetSoft). IEEE, 2016 [14].

An extension of Catenae that includes the hw-sw switch design and implementation has been published in the following peer-reviewed international journal.

- Roberto Bifulco, Anton Matsiuk, and ***Alessio Silvestro***. CATENAE: A scalable Service Function Chaining system for legacy mobile networks. International Journal of Network Management, 27(2), 2017 [15].

The preliminary work and the full system architecture on enabling Internet-wide SFC has been published in the following peer-reviewed international conference proceedings.

- ***Alessio Silvestro***, Roberto Bifulco, Fabian Schneider, Xiaoming Fu, and Jussi Kangasharju. "Is today's DNS the right solution for middleboxes selection?". Proceedings of the 4th Workshop on CrossCloud Infrastructures & Platforms. ACM, 2017. [Poster] [78].
- ***Alessio Silvestro***, Roberto Bifulco, Fabian Schneider, Xiaoming Fu, and Jussi Kangasharju. "MISE: MIddleboxes SElection for multi-domain service function chains." Proceedings of the 2nd Workshop on Cloud-Assisted Net-

working. ACM, 2017 [106].

Thus, the work on enabling *Multi-tier Edge Networks* has been published in the following peer-reviewed international conference proceeding.

- ***Alessio Silvestro***, Nitinder Mohan, Jussi Kangasharju, Fabian Schneider, and Xiaoming Fu. MUTE: MUlti-Tier Edge networks. In Proceedings of the 5th Workshop on CrossCloud Infrastructures & Platforms (CrossCloud'18). ACM, 2018. [108]

Patent Applications Further, related to the broad topic of this dissertation, preliminary results and ideas have been published as patent applications.

- ***Alessio Silvestro***, Dirk Kutscher, and Fabian Schneider. Method and System for Introducing In-Network Services in an End-To-End Communication Path. Publication number: *WO 2017/194168 A1*. Application date: 2016-05-13. Publication date: 2017-11-16. [109]
- Fabian Schneider, ***Alessio Silvestro***, and Thomas Dietz. Software Defined Network and Method for Operating the same. Application date: 2016-12-22. A method to flow installation time sensitive network control [110].
- ***Alessio Silvestro***, Fabian Schneider, and Roberto Bifulco. Explicit Service Function Chaining (SFC) using DNS extensions. Application date: 2017-03-10. [16]

The text of [109] is publicly available. [16, 110] instead, are still under review and the Patent applications text cannot be made public. However, I would like to invite Ph.D. committee members, who are interested in accessing those documents, to sign an Non-Disclosure Agreement (NDA) with NEC Laboratories Europe in order to

obtain a copy of the Patents application text.

Bibliography

[1] David Clark. The design philosophy of the darpa internet protocols. *ACM SIGCOMM Computer Communication Review (CCR)*, 18(4):106–114, 1988.

[2] Jerome H Saltzer, David P Reed, and David D Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[3] Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. September 15, 2017. [Online]. Available: `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`.

[4] [Published: 07.12.2015]. [On-line accessed: 22.05.2018]. [Online]. Available: `https://venturebeat.com/2015/12/07/streaming-services-now-account-for-over-70-of-peak-traffic-in-north-america-netflix-dominates-with-37/`.

[5] IBM Institute for Business Value, "Five telling years, four future scenarios," 2010.

[6] Jim Liddle. Amazon found every 100ms of latency cost them 1% in sales. *The GigaSpaces*, 27, 2008.

[7] Ibrar Yaqoob, Ejaz Ahmed, Muhammad Habib ur Rehman, Abdelmuttlib Ibrahim Abdalla Ahmed, Mohammed Ali Al-garadi, Muhammad Imran, and Mohsen Guizani. The rise of ransomware and emerging security challenges in the Internet of Things. *Computer Networks*, 129:444–458, 2017.

[8] Theophilus Benson, Aditya Akella, and David A Maltz. Unraveling the Complexity of Network Management. In *NSDI*, pages 335–348, 2009.

[9] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Intelligent design enables architectural evolution. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNETs'11)*, page 3. ACM, 2011.

[10] Raghavan, Barath and Casado, Martín and Koponen, Teemu and Ratnasamy, Sylvia and Ghodsi, Ali and Shenker, Scott. Software-defined internet architecture: decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNETs'12)*, pages 43–48. ACM, 2012.

[11] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[12] Jianli Pan, Subharthi Paul, and Raj Jain. A survey of the research on future internet architectures. *IEEE Communications Magazine*, 49(7), 2011.

[13] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review (CCR)*, 42(4):13–24, 2012.

[14] Roberto Bifulco, Anton Matsiuk, and Alessio Silvestro. Ready-to-deploy service function chaining for mobile networks. In *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pages 175–183. IEEE, 2016.

[15] Roberto Bifulco, Anton Matsiuk, and Alessio Silvestro. Catenae: A scalable service function chaining system for legacy mobile networks. *International Journal of Network Management (IJNM'17)*, 27(2), 2017.

[16] Alessio Silvestro, Roberto Bifulco, and Fabian Schneider. Explicit Service Function Chaining (SFC) using DNS extensions. Patent. Application date: 2017-03-10, 2017.

[17] Nick McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.

[18] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[19] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined WAN. 43(4):3–14, 2013.

[20] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review (CCR)*, 38(2):69–74, 2008.

[21] Masayoshi Kobayashi, Srini Seetharaman, Guru Parulkar, Guido Appenzeller, Joseph Little, Johan Van Reijendam, Paul Weissmann, and Nick McKeown. Maturing of openflow and software-defined networking through deployments. *Computer Networks*, 61:151–175, 2014.

[22] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 43, pages 15–26. ACM, 2013.

[23] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. Sdx: A software defined internet exchange. *ACM SIGCOMM Computer Communication Review (CCR)*, 44(4):551–562, 2015.

[24] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 45, pages 183–197. ACM, 2015.

[25] Georgios P Katsikas. *Realizing high performance NFV service chains*. PhD thesis, KTH Royal Institute of Technology, 2016.

[26] Open networking foundation. `https://www.opennetworking.org/`.

[27] R Guerzoni et al. Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In *SDN and OpenFlow World Congress*, volume 1, pages 5–7, 2012.

[28] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016.

[29] OpenFlow-enabled SDN and Network Functions Virtualization. Open Networking Foundation (ONF) Solution Brief. February 17, 2014.

[30] ETSI GS NFV 002 V1.2.1: Network Functions Virtualisation (NFV); Architectural Framework. ETSI Ind. Spec. Group (ISG) Netw. Functions Virtualisation (NFV), Sophia-Antipolis Cedex, France, Dec. 2014. [Online]. Available: `http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf`.

[31] Paul Quinn and Tom Nadeau. Problem Statement for Service Function Chaining. RFC 7498, April 2015.

[32] Joel Halpern and Carlos Pignataro. Service Function Chaining (SFC) Architecture. RFC 7665, October 2015.

[33] W. Haeffner et al. SFC Use Cases in Mobile Networks IETF Draft.

[34] S. Kumar et al. SFC Use Cases In Data Centers. IETF Draft.

[35] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 45, pages 199–212. ACM, 2015.

[36] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir. Segment Routing Architecture. IETF Draft.

[37] Pamela Zave, Ronaldo A Ferreira, Xuan Kelvin Zou, Masaharu Morimoto, and Jennifer Rexford. Dynamic service chaining with dysco. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*,

pages 57–70. ACM, 2017.

[38] Jon Matias, Jokin Garay, Nerea Toledo, Juanjo Unzilla, and Eduardo Jacob. Toward an sdn-enabled nfv architecture. *IEEE Communications Magazine*, 53(4):187–193, 2015.

[39] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is It Still Possible to Extend TCP? In *ACM IMC '11*, 2011.

[40] Zhaoguang Wang, Zhiyun Qian, Qiang Xu, Zhuoqing Mao, and Ming Zhang. An untold story of middleboxes in cellular networks. In *ACM SIGCOMM '11*.

[41] ETSI. Network Functions Virtualisation - White Paper. [Online]. Available: `https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf`.

[42] IETF. Service Function Chaining working group. SFC.

[43] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[44] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying middlebox policy enforcement using SDN. ACM SIGCOMM computer communication review. 43(4):27–38, 2013.

[45] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *NSDI*, volume 14, pages 533–546, 2014.

[46] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer. Position paper: Software-defined network service chaining. In *IEEE EWSDN '14*.

[47] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Geoffrey Lefebvre, Ravi Manghirmalani, Ramesh Mishra, Ritun Patneyt, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, et al. Steering: A software-defined networking for inline service chaining. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.

[48] P.Quinn, U.Elzur, and C.Pignataro. Network Service Header. IETF RFC8300. January 2018.

[49] Mohamed Boucadair, Christian Jacquenet, Yuanlong Jiang, Ron Parker, and Kengo. Requirements for service function chaining (sfc). Internet-Draft draft-boucadair-sfc-requirements-06, IETF Secretariat, February 2015.

[50] Dan Levin, Marco Canini, Stefan Schmid, and Anja Feldmann. Incremental sdn deployment in enterprise networks. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 43, pages 473–474. ACM, 2013.

[51] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. ClickOS and the Art of Network Function Virtualization. In *USENIX NSDI '14*, 2014.

[52] Stefano Vissicchio, Laurent Vanbever, and Jennifer Rexford. Sweet little lies: Fake topologies for flexible routing. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 3. ACM, 2014.

[53] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. Investigating Transparent Web Proxies in Cellular

Networks. In *PAM*. Springer, 2015.

[54] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. An in-depth study of LTE: Effect of network protocol and application behavior on performance. In *ACM SIGCOMM '13*.

[55] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 163–174. ACM, 2013.

[56] Philipp Richter, Mark Allman, Randy Bush, and Vern Paxson. A primer on ipv4 scarcity. *ACM SIGCOMM Computer Communication Review (CCR)*, 45(2):21–31, 2015.

[57] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vswitch. In *USENIX NSDI '15*.

[58] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the "one big switch" abstraction in software-defined networks. In *ACM CoNEXT'13*.

[59] Maurizio Dusi, Roberto Bifulco, Francesco Gringoli, and Fabian Schneider. Reactive logic in software-defined networking: Measuring flow-table requirements. In *Proceedings of the 5th International Workshop on TRaffic Analysis and Characterization (TRAC)*, 2014.

[60] Aggelos Lazaris, Daniel Tahara, Xin Huang, Li Erran Li, Andreas Voellmy, Y. Richard Yang, and Minlan Yu. Tango: Simplifying SDN Programming with Automatic Switch Behavior Inference, Abstraction, and Optimization. In *ACM CoNEXT'14*.

[61] Roberto Bifulco and Anton Matsiuk. Towards scalable sdn switches: Enabling faster flow table entries installation. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 45, pages 343–344. ACM, 2015.

[62] Naga Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. Infinite cacheflow in software-defined networks. In *Proceedings of the third workshop on Hot topics in Software Defined Networking (HotSDN'14)*, pages 175–180. ACM, 2014.

[63] Open Networking Foundation. Openflow specification 1.4.0.

[64] Nadi Sarrar, Steve Uhlig, Anja Feldmann, Rob Sherwood, and Xin Huang. Leveraging zipf's law for traffic offloading. *ACM SIGCOMM Computer Communication Review (CCR)*, 42(1):16–22, 2012.

[65] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, August 2000.

[66] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. Past: Scalable ethernet for data centers. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 49–60. ACM, 2012.

[67] Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, An-

drew R. Curtis, and Sujata Banerjee. Devoflow: cost-effective flow management for high performance enterprise networks. In *ACM SIGCOMM HotNets '10*.

[68] David Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2013.

[69] Ryota Kawashima, Shin Muramatsu, Hiroki Nakayama, Tsunemasa Hayashi, and Hiroshi Matsuo. Sclp: Segment-oriented connection-less protocol for high-performance software tunneling in datacenter networks. In *IEEE NetSoft '15*.

[70] Stoke, Inc. LTE equipment evaluation: Considerations and selection criteria, 2012.

[71] Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. mSwitch: a highly-scalable, modular software switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 1. ACM, 2015.

[72] NEC. Programmableflow pf5240 switch.

[73] Cisco. Visual networking index: Global mobile data traffic forecast update 2014–2019 white paper.

[74] Sergey Guenender, Katherine Barabash, Yaniv Ben-Itzhak, Anna Levin, Eran Raichstein, and Liran Schour. Noencap: Overlay network virtualization with no encapsulation overheads. In *ACM SOSR*, 2015.

[75] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu. Research directions in

network service chaining. In *IEEE SDN4FNS '13*.

[76] Vittorio Manetti, Pasquale Di Gennaro, Roberto Bifulco, Roberto Canonico, and Giorgio Ventre. Dynamic virtual cluster reconfiguration for efficient iaas provisioning. In *Proceedings of the 2009 International Conference on Parallel Processing*, Euro-Par'09, pages 424–433, Berlin, Heidelberg, 2010. Springer-Verlag.

[77] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. And then there were more: Secure communication for more than two parties. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 88–100. ACM, 2017.

[78] Alessio Silvestro, Roberto Bifulco, Fabian Schneider, Xiaoming Fu, and Jussi Kangasharju. Is today's DNS the right solution for middleboxes selection? In *Proceedings of the 4th Workshop on CrossCloud Infrastructures & Platforms, in ACM EuroSys'17*, page 6. ACM, 2017.

[79] C. Contavalli et al. Client Subnet in DNS Queries. RFC 7871.

[80] A.Silvestro et al. Issues in Supporting Third-Partys In-Network Services in the Internet, NetSys'17.

[81] R.Mahajan et al. Inferring link weights using end-to-end measurements. In *ACM SIGCOMM IMW 2002*.

[82] Giuseppe Siracusano, Roberto Bifulco, Simon Kuenzer, Stefano Salsano, Nicola Blefari Melazzi, and Felipe Huici. On the Fly TCP acceleration with miniproxy. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes*

and Network Function Virtualization. In ACM SIGCOMM'16, pages 44–49. ACM, 2016.

[83] Single-Source Shortest Paths in Directed Acyclic Graphs. [Online]. Available: `http://www.utdallas.edu/~sizheng/CS4349.d/l-notes.d/L17.pdf`.

[84] J. Brutlag. Speed matters for Google web search, June 2009.

[85] Bruce M Maggs and Ramesh K Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review (CCR)*, 45(3):52–66, 2015.

[86] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of an anycast cdn. In *Proceedings of the 2015 Internet Measurement Conference (IMC)*, pages 531–537. ACM, 2015.

[87] Previdi et al. IPv6 Segment Routing Header (SRH). IETF Draft.

[88] David Lebrun. Leveraging IPv6 Segment Routing for Service Function Chaining. In *CoNEXT 2015 student workshop*, 2015.

[89] Ahmed AbdelSalam, Francois Clad, Clarence Filsfils, Stefano Salsano, Giuseppe Siracusano, and Luca Veltri. Implementation of virtual network function chaining through Segment Routing in a linux-based NFV infrastructure. In *Network Softwarization (NetSoft), 2017 IEEE Conference on*, pages 1–5. IEEE, 2017.

[90] Abhishek Chandra, Jon Weissman, and Benjamin Heintz. Decentralized Edge Clouds. *IEEE Internet Computing*, 17(5):70–73, 2013.

[91] ETSI MEC-IEG004. Mobile-Edge Computing (MEC): Service Scenarios.

[92] Nitinder Mohan, Pengyuan Zhou, Keerthana Govindaraj, and Jussi Kangasharju. Managing Data in Computational Edge Clouds. Proceedings of the Workshop on Mobile Edge Communications (MECOMM'17). pages 19–24. ACM, 2017.

[93] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile Cloud Computing (MCC'13)*, pages 15–20. ACM, 2013.

[94] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile Cloud Computing (MCC)*, pages 13–16. ACM, 2012.

[95] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[96] Hu et al. Mobile edge computing – A key technology towards 5G. *ETSI white paper*, 2015.

[97] Nitinder Mohan and Jussi Kangasharju. Edge-Fog cloud: A distributed cloud for Internet of Things computations. In *Cloudification of the Internet of Things (CIoT)*, pages 1–6. IEEE, 2016.

[98] Xu et al. Multi-objective virtual machine placement in virtualized data center environments. In *IEEE GreenCom*, 2010.

[99] Abhinandan S Prasad, Mayutan Arumaithurai, David Koll, and Xiaoming Fu.

Raera: A robust auctioning approach for edge resource allocation. In *Proceedings of the Workshop on Mobile Edge Communications (MECOMM'17)*, pages 49–54. ACM, 2017.

[100] Mahajan et al. Inferring link weights using end-to-end measurements. In *ACM SIGCOMM Internet Measurement Workshop (IMW) 2002*.

[101] Md Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys & Tutorials*, 15(2):909–928, 2013.

[102] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things (IoT) journal*, 1(1):22–32, 2014.

[103] CISCO. Fog Computing and IoT (Whitepaper). 2015.

[104] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile Cloud Computing (MCC'13)*, pages 15–20. ACM, 2013.

[105] Robert Bosch GmbH Bosch APAS description. [Online]. Available: `https://www.bosch-apas.com/produkte-und-services/apas-assistant-mobile/`.

[106] Alessio Silvestro, Roberto Bifulco, Fabian Schneider, Xiaoming Fu, and Jussi Kangasharju. MISE: MIddleboxes SElection for multi-domain service function chains. In *Proceedings of the 2nd Workshop on Cloud-Assisted Networking, in ACM CoNEXT'17*, pages 37–42. ACM, 2017.

[107] Mario Almeida, Alessandro Finamore, Diego Perino, Narseo Vallina-Rodriguez, and Matteo Varvello. Dissecting dns stakeholders in mobile networks. 2017.

[108] Alessio Silvestro, Roberto Bifulco, Jussi Kangasharju, Fabian Schneider, and Xiaoming Fu. MUTE: MUlti-Tier Edge networks. In *Proceedings of the 5th Workshop on CrossCloud Infrastructures & Platforms (CrossCloud), in ACM EuroSys'18*. ACM, 2018.

[109] Alessio Silvestro, Dirk Kutscher, and Fabian Schneider. Method and system for introducing in-network services in an end-to-end communication path. Patent, 2017. Publication number: WO 2017/194168 A1. Publication date: 2017-11-16. Application date: 2016-05-13.

[110] Fabian Schneider, Alessio Silvestro, and Thomas Dietz. Software Defined Network and Method for Operating the same. Patent. Application date: 2016-12-22, 2017.

Alessio Silvestro

Via Domenico Zampieri, 34
40121, Bologna, Italy
+39 (0) 338 8917706
Alessi.Silvestro@gmail.com
alessiosilvestro

Energetic, ambitious and strongly motivated Network Engineer with deep technical experience on Cloud & Edge Computing, Software Defined Networking (SDN), Network Function Virtualization (NFV) and High Performance Networking Stack. I co-authored several scientific publications at top-tier conferences and patent applications. Additionally, I have been involved in project management tasks for several European and NEC Japan driven projects. Great communication skills and strong orientation towards team building.

Experience

June'18–Present **Engineer**, *Philip Morris International*, Operations Excellence.

Feb–July'17 **Visiting Researcher**, *COllaborative NEtworking (CONE) Group at Univesity of Helsinki, Finland (headed by Prof. Jussi Kangasharju).*

Apr'15–Apr'18 **Early Stage Researcher**, *Software Defined Networking (SDN) Group, Network Research Division*, NEC Europe Ltd., Germany.

Keywords, *Software Defined Networking (SDN), Cloud & Edge Computing, Network Function Virtualization (NFV), High-performance Networking Stack.*

Mar–Dec'14 **Research Intern**, *IoT Group*, NEC Europe Ltd., Germany.

Keywords, *Context Management, Multi-device, Web Applications. EU FP7 MediaScape.*

Education

Sep'15–Present **Ph.D. Fellow**, *Marie SkÅĆodowska-Curie Actions Research Fellowship*, Computer Network Group (headed by Prof. Xiaoming Fu), Georg-August-Universität Gőettingen (Germany).

Mar'12– Apr'15 **Master Degree in Computer Engineering –110/110 cum Laude**, *University of Naples Federico II*, (Italy).

Sep'08-Jan'12 **Bachelor Degree in Computer Engineering – 109/110**, *University of Naples Federico II*, (Italy).

Languages

Italian	Native	English	Full professional working proficiency
German	Basic communication skills	French	Basic communication skills
Spanish	Basic communication skills		

Computer Skills

Program. Lang.	C/C++, Java, JavaScript, JQuery, Python, Bash Scripting
Cloud Platforms	Amazon EC2, Amazon S3, OpenStack, Microsoft Azure, VMware, KVM
SDN Controller	ONOS, OpenDaylight, Ryu, Floodlight
Fast Packet Proc.	Vector Packet Processing (VPP), Data Plane Development Kit (DPDK), PKTGEN-DPDK
Network	TCP/IP, Linux Networking Stack, DNS, SmartNIC (Netronome NFP-4/6XXX), FTP, HTTP, HTTPS, UDP, SSH, VLAN, VxLAN, OpenVSwitch, Libvirt, QEMU/KVM
DBMSs	MySQL,Oracle, SQLite
Office Automation	MS Office
Softwares	Matlab/Simulink, JMP Statistical Discovery, Sharpe, LT Spice, LabView, SIS, LaTex, Shell
Middleware	Corba

OS Unix, Linux, MS Windows, Mac OS, Linux RTAI

Relevant Projects

Mar-Jun'17 **Big Data Frameworks**

Keywords: Resilient Distributed Datasets (RDD), Map Reduce, Spark, Machine Learning

2015-Present **A Network for the Cloud Computing Eco-System**

EU Fundings: EU FP7 ITN CleanSky

Keywords: Cloud Computing, Cross-Datacenter Cloud, Virtualization, NFV, SDN, High-performance Networking Stack

2014 ***A Rule Based Adaptation Engine for Multi-Device Context-Aware Web Applications (Master Thesis)***

EU Fundings: EU FP7 MediaScape

Keywords: Multi-device application, Context-management, Data Analytics

2014 **Workload Characterization - Performance Analysis - Capacity test**

Keywords: Workload Characterization (Application & System Level), Analysis of Variance (ANOVA), Principal Component Analysis (PCA), Clustering, Linear Regression Model, Experimental Design and Analysis, Benchmarking of Virtualized System (SciMark V2.0, IOzone), Apache JMeter, Jmp Statistical Discovery

2013 **Live Migration of Entire Virtual Networks on OpenVSwitches**

Keywords: SDN, OpenFlow, Floodlight, OpenVSwitch, Libvirt, QEMU/KVM, iSCSI, Apache Tomcat

2013 **CoDesign Software and Hardware of an Electronic Embedded System**

Keywords: VHDL, Xilinx ISE, ARM Cortex based SoC, FPGA, Device Drivers for Leon3-SPARC processor, Design and Development of SPI (Serial Peripheral Interface) HW-Controller for Leon3-SPARC,

2011 ***Monitoring of Distributed Infrastructure Systems (Bachelor Thesis)***

Description: Monitoring of a distributed installation of OpenStack using Nagios.

Keywords: Distributed Systems, OpenStack, Nagios, Unix, Shell Scripting

Publications

2017 A. Silvestro, R. Bifulco, F. Schneider, Xiaoming Fu, Jussi Kangasharju: MISE: MIddleboxes SElection for Multi-Domain Service Function Chains. In Proceedings of Cloud Assisted Network (CAN'17)– ACM CoNEXT 2017.

2017 A. Silvestro, R. Bifulco, F. Schneider, X. Fu, J. Kangasharju: Is today's DNS the right solution for middleboxes selection?. ACM CrossCloud'17 – ACM EuroSys'17.

2017 A. Silvestro, R. Bifulco, S. Sharma, F. Schneider, J. Kangasharju, X. Fu: Issues in Supporting Third-Partys In-Network Services in the Internet. In Proceedings of NetSys'17.

2017 R. Bifulco, A. Matsiuk, A. Silvestro: CATENAE: A scalable service function chaining system for legacy mobile networks. International Journal of Network Management.

2016 R. Bifulco, A. Matsiuk, A. Silvestro: Ready-to-deploy service function chaining for mobile networks, In Proceedings of the 2nd IEEE Conference on Network Softwarization (NetSoft).

Interests

Photography Movies

Guitar Reading

Sports (e.g., basketball, beach volley, bowling, table tennis, etc.)