Service Lifecycle Management in the Cloud Continuum

Sami Yangui

Université de Toulouse

Institut National Polytechnique de Toulouse

Laboratoire d'Analyse et d'Architecture des Systèmes

Professorial Dissertation

for the degree of

HDR in Computer Science

prepared by

Dr. Sami Yangui

Service Lifecycle Management in the Cloud Continuum

Presented on 19-02-2024, at LAAS-CNRS, in front of the exam jury:

Reviewer:	Prof. Boualem Benatallah	Dublin City University, Ireland
Reviewer:	Prof. Schahram Dustdar	Technischen Universität Wien, Austria
Reviewer:	Prof. Massimo Tornatore	Politecnico di Milano, Italy
Examiner:	Prof. Alberto Leon-Garcia	University of Toronto, Canada
Examiner:	Prof. Brahim Medjahed	University of Michigan-Dearborn, USA
Examiner:	Prof. Lionel Seinturier	Université de Lille, France
Examiner:	Prof. Mickael Sheng	Macquarie University, Australia
Advisor:	Dr. Khalil Drira	LAAS-CNRS, Toulouse, France

"Citius, Altius, Fortius"
Olympics Motto...

Acknowledgement

La reconnaissance est la mémoire du coeur.
Hans Christian Andersen, écrivain danois (1805-1875)

To Everyone Who Made This Possible ...

Abstract

Service computing (alternatively termed service-oriented computing) refers to the cross-discipline that covers the science and technology binding IT services to business services. Service computing helps in the modeling and the support of complex processes while combining business goals and technical concerns. Specifically, it focuses on the co-creation of value between service consumers and service providers in the context of the several steps (e.g., design, implementation, execution) that make up the lifecycle of the service. The underlying key enabling technologies of service computing involve Service-Oriented Architecture (SOA), cloud computing, social computing, business process management and Web services. Emerging service-based applications in such environments are distributed, modular, reusable through composition techniques (e.g., service orchestration, service choreography) and might adopt novel design patterns that are even more distributed (e.g., micro services, Function-as-a-Service).

From one side, the emerging applications' services are getting more and more heterogeneous, distributed and mobile. They impose strong requirements and strict Quality of Service (QoS) management on the hosting service providers. Platoons of connected automated vehicles, adaptive streaming and augmented/virtual reality are among the examples of emerging applications. On the other side, with the rise of the cloud continuum, the target runtime environments for these applications are getting more and more distributed, heterogeneous, dynamic and mobile. The cloud continuum encompasses a range of resources and capabilities from public to edge and everything in between, all seamlessly integrated by the so-called next-generation networks (e.g., The fifth-generation/sixth-generation (5G/6G) wireless telco network, content delivery networks (CDN) and information-centric networks).

This research work focus on re-considering the service lifecycle management process to support the provisioning of novel and emerging applications in the cloud continuum and its inherent next-generation networks and infrastructures. It proposes several research contributions that cover and support every single step of the process. For practical reasons and for the sake of efficiency, the target contributions mostly focus on NFV domain and consider the Virtualized Network Functions (VNF) as the service building block of the cloud continuum. The contributions range from design (i.e., architectures, specification, prototypes) to models (i.e., semantic-based models, optimization models) and procedures (i.e., description, automation, (re)configuration). The research results were reported in several highly-ranked forums in the networking and service computing research fields.

Keywords: Cloud Continuum - Everything as-a-Service (XaaS) - Network Function Virtualization - Quality of Service (QoS) - Service Computing - Service Lifecycle Management - Service-oriented Architecture.

Table of contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

The term "service" has already existed for several centuries along human history. According to Cambridge dictionary, a service is *"a process that creates benefits by facilitating a change in customers, a change in their physical possessions, or a change in their intangible assets"* [1]. Services are intangible and non-physical, as opposed to goods. Basically, one can call service any performed work that is aimed to benefit another (e.g., transportation service, kitchen service, religious ceremony service). As for service computing, the concept refers to the cross-discipline that covers the science and technology that bind IT services to business services [2]. Service computing helps in the modeling and the support of complex processes while combining business goals and technical concerns [3]. Specifically, it focuses on the co-creation of value between service consumers and service providers in the context of the several steps (e.g., design, implementation, execution) that make up the lifecycle of the service.

The underlying key enabling technologies of service computing involve Service-Oriented Architecture (SOA), cloud computing, social computing, business process management and Web services. These technologies can be combined together to support and provision even more complex and "sophisticated" services [4]. For instance, the adoption of the SOA principles and cloud computing in the Web 2.0 led to the rise of the so-called *Everything-as-a-Service* (XaaS for short) concept. XaaS encompasses any kind of IT services (e.g., monitoring service, storage service) that could be remotely offered by cloud providers to prospective cloud end-users according to the so-called *pay-as-you-go* business model [5]. With the wider embracement of SOA and cloud computing principles in business, the XaaS concept evolved to refer, not only to IT services, but also to any kind of business services that fall under the defintion introduced in the previous paragraph. For instance, Airbnb[1] is considered as one of the biggest hotelier in the world. Basically, it is one of the very few, to not say the unique, hotelier that could propose accommodation in almost every single city on Earth. However, Airbnb owns neither hotels nor any accommodation facilities

[1] `www.airbnb.com`

in these locations. Similarly, Netflix[2] is currently considered as one of the biggest entertainement content delivery platform, while the company does not own neither produce any (or very little) of that content. Google[3] provides genuine services (e.g., Google scholar, Google flights, Google weather) to end-users that completely rely on third-party owners.

Specifically, these service providers rely on XaaS concept with features and added-value products managed and offered as services to consumers. They do operate massive infrastructures and content according to well-designed business processes and procedures to provide the prospective service consumers with the requested services while ensuring the required Quality of Service (QoS) all along. For instance, Netflix introduced the DevOps perspective to bring agility and automation prior to content delivery [6] while Google pioneered the Site Reliability Engineering (SRE) concept to enhance service availability and to automate part of the management procedures in its datacenters [7]. The ultimate goal for these providers is to bring agility, achieve scalability and optimize the cost-effectiveness while delivering the services to consumers in the context of XaaS. During the last years, telecommunications and networking providers followed the trend as well. Several telco compagnies henceforth rely on Mobile Virtual Network Operators (MVNO) and ally with cloud providers to reduce the high Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) of their investments and deliver their services according to the XaaS concept. Achieving the XaaS vision for telco and network providers is more challenging than for any other kind of providers. This is due to the proprietary environments they rely on, as well as, the strong coupling between the hardware and the software resources in their runtime. Virtualization technology and cloud computing allow to tackle these challenges. For instance, Vodafone collaborate with Amazon Web Services to achieve the so-called Telco-as-a-Service (TaaS). TaaS is a cloud-native framework that changed the way the telco services are composed and delivered to end-users. For Vodafone, TaaS allows the telco provider to dynamically auto-provision complex environments following a variable workload while optimizing the cost and the performance through DevOps techniques [8]. Generally speaking, TaaS helps telco providers to operate scalable and cost-efficient services for MVNOs. Furthermore, it brings agility to the network. The underlying infrastructures become multi-tenanted and the providers can seamlessly run any operator type (e.g., mobile, fixed-line) and support any kind of mobile services (e.g., 4G LTE, 5G) over the allocated network resources.

There has always been a close relationship between the emergence of new computing models/technologies and new applications. XaaS concept, with its underlying cloud and SOA-based computing models, enabled a myriad of novel service-based applications that range from agriculture to smart home/city, transportation and healtcare. The evolution of the specifications of the hosting providers to the so-called next-generation networks is motivated by the novel, strong and changing requirements that

[2] `www.netflix.com`
[3] `www.google.com`

emerging online applications impose on the hosting infrastructures. These applications are usually compute-intensive and latency-sensitive at the same time [9] [10]. Indeed, they often handle huge amounts of data, implement complex analytics and processing models and stand in need of reliable networking and communications infrastructures. Furthermore, most of these applications require mobility support during runtime. Unmanned systems (e.g., autonomous cars [11] [12], car platooning [13]), augmented/virtual reality [14] and adaptive streaming [14] are among the examples of emerging online applications. The fifth-generation/sixth-generation (5G/6G) wireless telco network, Content Delivery Networks (CDN) and Information-Centric Networks (ICN) are considered as next-generation networks and among the prospective hosting environments for these emerging applications.

1.2 Motivations

Existing cloud providers, whatever they are, rely on the pretty same core architecture. The limitations that prevent the proper provisioning of the emerging applications in fully cloud environments are intrinsically related to their inner architecture. Broadly speaking, they are mainly related to the strong requirements imposed by these applications on: (i) the data plane (i.e., data support and processing), (ii) communication and signaling plane (i.e., intra/inter service communications through the network) and (iii) the control plane (communication protocols and networking rules).

The first limitation is a direct consequence of the cloud ecosystems topology. Cloud providers use regional centralized datacenters to host and execute services. They all copy and/or sync up the data to a centralized clusters. This is suitable for human-centric and regular Web applications but has major deficiencies when it comes to the support of the emerging applications provisioning. Indeed, such applications are designed for machine-centric models with services implementing complex machine learning models and exchanging (i.e., producing and/or consuming) huge amount of data during runtime. For example, unmanned systems use computer-vision models to implement image classification/segmentation services and CDN use ensemble-based machine learning to enable the reliabilty of the caching services [15]. Provisioning these services on centralized cloud infrastructures suffer from numerous limitations such as storage complexity incurred while storing the extra data chunks required for machine learning models, processing complexity due the distribution of the data (e.g., collecting and routing raw data from data sources, ingesting offline data from data lakes, indexing and storing processed data in a search engine) and excess processing time.

As for the second limitation, the fact that cloud servers sit at the backhaul of the network, distant from end-users, data sources and/or from any other potential interacting services, makes the SLA management for the emerging applications difficult to achieve, taking into consideration that most of these applications are very demanding on the network appliances [16]. In particular, this applies on the networking metrics

as the cloud providers are entirely dependant on third-party network infrastructures to communicate with distant services and end-users. It should be noted that it is not just matter of optimizing the common network metrics (e.g., decreasing the latency, increasing the bandwith, decreasing the jitter, minimising the packets loss) but it is more about maintaining these metrics stable and within the required range during the whole runtime. Obviously, this cannot be guaranteed by the cloud providers as they neither own nor control the communication network in between the remote services and the cloud facilities.

Last but not least, the third limitation might impair the proper functioning of the emerging applications as their associated services, or part of their associated services, might move during runtime (e.g., augmented reality app installed on a smartphone, autonomous car riding in the city and connecting to live traffic service, CDN provider delivering live streaming on a mobile device). This means that some data sources, end-users and/or services might change location at runtime (e.g., see [12] for the case of autonomous cars). Consequently, cloud providers need to dynamically adapt and adjust the end-to-end communications, the routing rules and even the considered communication protocols/technologies in some cases (e.g., end device switching from WiFi to 4G LTE network during live streaming). This assumes that cloud providers support every single communication protocol, are able to reach any network domain and have total control on the configuration of the network in between the cloud facilities, from one side, and the remote services communicating through the network from the other side, which is not true considerating the discussion elaborated in the previous paragraph. In fact, this limitation was the major impediment to achieve the so-called Mobile Cloud Computing (MCC) in the past.

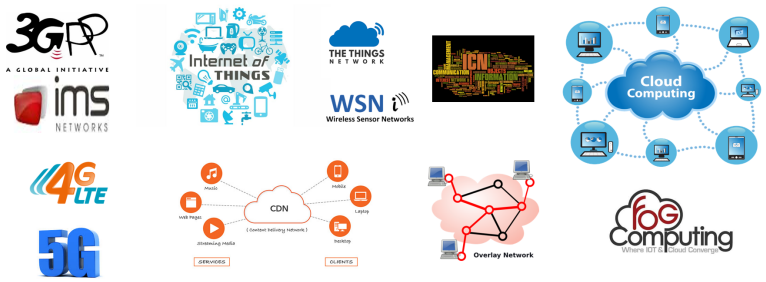


Figure 1.1: Examples of next-generation networks.

Next-generation networks represent the impending answer to address these three limitations [11] [17]. Figure 1.1 shows a non-comprehensive list of next-generation networks. Basically, most of them are part of the cloud continuum. Fog and Edge networks extend the cloud providers' capabilities and provide additional resources close to the data sources and/or end-users [16]. The Internet of Things (IoT) is a ubiquitous network of various objects connected over the Internet and uses henceforth the cloud for storage and computing purposes [18]. A dynamic overlay network implements the

necessary gateway operations (e.g., communication protocols conversion, data formatting) in between the IoT devices and the cloud resources. The telco networks (e.g., 4G LTE, 5G, 6G) rely on cloud for data storage and to host and to execute telecommunication services [19] [20]. In addition to the cloud, next-generation networks adopt novel and emerging computing paradigms and networking approaches. For instance, 5G relies on key concepts such as Network Function Virtualization (NFV), Software-defined Network (SDN), Multi-access Edge Computing (MEC) and Next-generation Protocols (NGP). Specifically, 5G specifications recommend slicing the network into several logic and functional entities that could be virtualized to enable agile and cost-effective operation [21]. The control plane is handled by SDN that supports dynamic reshape of the traffic while MEC provides computing capabilities at the edge of the 5G network [21]. MEC environment is characterized by ultra-low latency and high bandwidth, as well as, real-time access [16]. MEC resources are accessible through NGP and/or Radio Area Network (RAN).

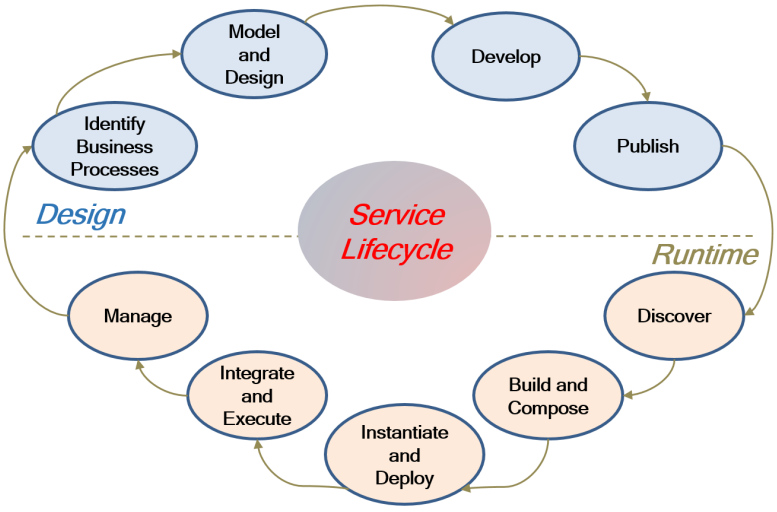


Figure 1.2: The service computing lifecycle management process.

Generally speaking, in contrast to fully-cloud environments, next-generation networks rely on highly distributed architectures and hybrid domain topologies. This inevitably impacts the lifecycle of service-based applications when they are being provisioned over these environments. Figure 1.2 depicts the several steps that make up the lifecycle of services. Provisioning service-based applications refers to the support of every single step in this lifecycle [22] [23]. During the *Design* phase, service developers identify and model the business processes describing the IT procedures that should be supported by the target service. This is followed by the implementation of the service (e.g., writing the source code, compiling, testing) and, then, the publication of the resulting archives on dedicated repositories or marketplaces. During the

Runtime phase, the requested services are first discovered from the repositories. Prior to the service instantiation and deployment, an intermediate composition step might be needed to build up complex and sophisticated service from the several elementary dicovered services. This is necessary whenever none of the elementary services could implement the requested business functionality [22]. After that, the service is configurated so that it could be integrated to the target deployment environment (e.g., initializing parameters, settling endpoint address and/or listening port). This is followed by the execution and the management steps. The management step implements a set of procedures that aim to optimize the functionning and the usage of the service during runtime. The intent of these procedures is to execute well-defined algorithms and techniques to meet the service-level objectives at runtime (e.g., reduce response time, reduce operation cost, increase availability). It should be noted that the service lifecycle is modeled as an infinite loop. Once the service is executed and being managed, one might decide to push it back to the *Design* phase and make one more iteration through the provisioning process (e.g., perform new updates on the service, integrate additional functionality to the service).

On one hand, *the emerging applications' services are getting more and more numerous, finer and mobile*. The adoption of novel design patterns, such as micro-services or Function-as-a-Service (FaaS), led to a proliferation in the number of the services and brought significant changes to their specifications. The services are henceforth finer in terms of granularity (i.e., the implemented business functionality). Furthermore, these services became completely standalone and entirely decoupled from their hosts as their required execution environments (e.g., librairies, frameworks) are built and encapsulated around the service prior to its deployment (e.g., microservices hosted and executed on Docker, serveless FaaS). Therefore, the services could be mobile during the *Runtime* phase and might migrate from one host to another within the network or even switch from one network domain to another. The prospective mobility of one or several services of the same application does not impact the end-to-end execution capability since the services are decoupled, have proper addressing schemes (e.g., Unified Address Identifier-URI) and implement distributed and common architectural style (e.g., REpresentational State Transfer-REST).

On the other hand, the target environments to provision these services, i.e., *the next-generation networks, are getting more and more distributed, hetergenous, dynamic and mobile*. The distribution of the these networks is due to the adoption of the cloud continuum. The latter consists of several domains (e.g., cloud, fog, edge, gateways) while increasing the whidening of the target environments' multi-tenancy. The strong heterogeneity of the next-generation networks is related to the wide range of its appliances in terms of capabilities (e.g., computing clusters, storage volumes, sensors, network controllers) and behavior (e.g., static servers, stationary sensors, flying drones, rolling robots). Furthermore, the topology of these networks is highly dynamic and mobile. This is due to the behavior of part of its constituents nodes. Next-generation networks involve nomad components (e.g., smartphones, smart watches, drones, con-

nected cars). While some of these components are part of the network domain, other components might join/drop the network domain with an arbitrary pattern. In fact, the next-generation networks can crowdsource the workload on these components in an opportunistic way considering their availability and location, from one side, and the service SLA from the other side. Consequently, contrary to regular networks, the topology of the next-generation networks constantly changes and evolves during runtime. This dynamicity is even further reinforced considering the potential mobility of some of the network components.

Provisioning emerging applications in the next-generation networks addresses the three previously discussed limitations related to the use of fully-cloud environment. However, the specific characteristics of their constituents services, as well as, the properties of their target environments drive fundamental changes in their associated lifecycle management. This research trial aims to study these changes, together with the necessary contributions to enable proper provisioning of these services in the next-generation networks.

1.3 Research issues, challenges and objectives

The main research question that needs to be answered in this context could be formalized as follows: "***How to properly provision emerging applications' services in next-generation networks with respect to the service computing paradigm?***". With other terms, the research question would be: "***What are the required adaptations and changes that are needed to revisit every single step in the whole service lifecycle process to support such provisioning?***". To answer this question, one should first bring anwsers to these underlying questions: "*What is the impact of adopting novel paradigms and technologies (e.g., NFV, MEC) on the service lifecycle process?*", "*How to integrate that in the service lifecycle management?*" and, above all, "*What are the requirements to support and to unlock the full potential that next-generation networks could bring to the emerging applications?*".

Provisioning emerging applications in next-generation networks inevitably affects the lifecycle of their constituents services considering the distribution, the dynamicity and the prospective mobility in the target networks. While the lifecycle phases remain naturally the same, the procedures that defines and implements these phases need to be adapted or even reinvented. To that end, several challenges need to be tackled. The first one is related to the adoption of the novel paradigms and technologies. For instance, the integration of NFV enables "*softwarising*" the network appliances and brings, henceforth, agility and cost-effectiveness to the network. The integration of MEC introduces, among others, the support of the multi-cast broadcasting and triggers dynamic netwotk topology evolutions at runtime. Obviously, the service lifecycle management process should be flexible and agile enough to catch up with these dynamic changes.

The second challenge is related to the automation of the lifecycle process man-

agement. This includes the foreseen actions at every single step of the lifecycle but also the moving from one step to the next. Similarly to the DevOps/SRE approach in software engineering or the continuous configuration and automation approach in networking, this is necessary to meet the strong requirements imposed by the services (e.g., latency-sensitiveness, location-awareness) and keep up with the dynamic changes of the hosting network (e.g., workload variation, mobility).

The third challenge is related to the switching from a "*centralized*" single cloud domain to a decentralized multi-domain provisioning environment. Indeed, managing the service lifecycle over several interacting domains should take into consideration the several information (e.g., capabilities, business model) for all involved domains. Ideally, the service lifecycle process is able to normalize and homogenize the information that come from seperate and hetergenous domains before integrating them to the foreseen actions in the process. For instance, the management phase of the lifecycle should support migrating a running service from a cloud host to an edge/fog host (and vice versa) to meet the required SLO.

For practical reasons and for the sake of efficiency, the target contributions mostly focus on NFV domain and consider the Virtualized Network Functions (VNF) as the service building block of the cloud continuum. In this work, NFV refers to the European Telecommunications Standards Institute (ETSI) initiative[4] to virtualize network services that traditionally run on proprietary and dedicated network appliances (e.g., switches, routers, DNS servers)

In fact, any other kind of services, associated to any given resource that is part of a different domain (e.g., IoT, Fog/Edge, CDN) can be considered as a network function. For instance, in the case of IoT, the gateways act as network middleboxes between the IoT devices and the IoT applications and are responsible of operations such as communication protocols conversion (e.g., from COAP/MQTT to HTTP) or message formatting (e.g., from raw data stream to well-structured CSV/JSON document). Similarly, the IoT devices (e.g., sensors) own computing capabilities and are able to some extent to host and execute VNF services (e.g., data transcoding). As for the CDN case, CDN controllers implement regular network controllers in between the primary servers, where the raw content is stored, and the surrogate servers, where this content is duplicated. As for the Fog/Edge domains, it is not uncommon to use IoT devices and/or CDN surrogate severs as Fog/Edge nodes to host and execute services. For instance, the surrogate servers keep and then deliver the content as close as possible to end-users. Besides, prior to the content delivery, CDN often provision network middleboxes, that could be implemented as VNF services, in between the surrogate servers and the end-users (e.g., location-based ads service aiming to enrich the raw content to value-added content prior to delivery).

[4] `https://www.etsi.org/technologies/nfv`

1.4 Research contributions and substantial results

The research work discussed in this manuscript introduces methodologies, models and procedures to support provisioning of services in the next-generation networks. The proposed contributions enable agile, fully automated and efficient provisioning of the services in their target environments, in accordance with the service computing paradigm. Figure 1.3 lists the main contributions and positions them in relation to the service lifecycle steps. These contributions cover every single step of the lifecycle. The two first contributions cover the *Design* phase of the service lifecycle. The third contribution involves the early steps of the *Runtime* phase. The fourth and the fifth contributions are intented to the management phase of the *Runtime* phase.

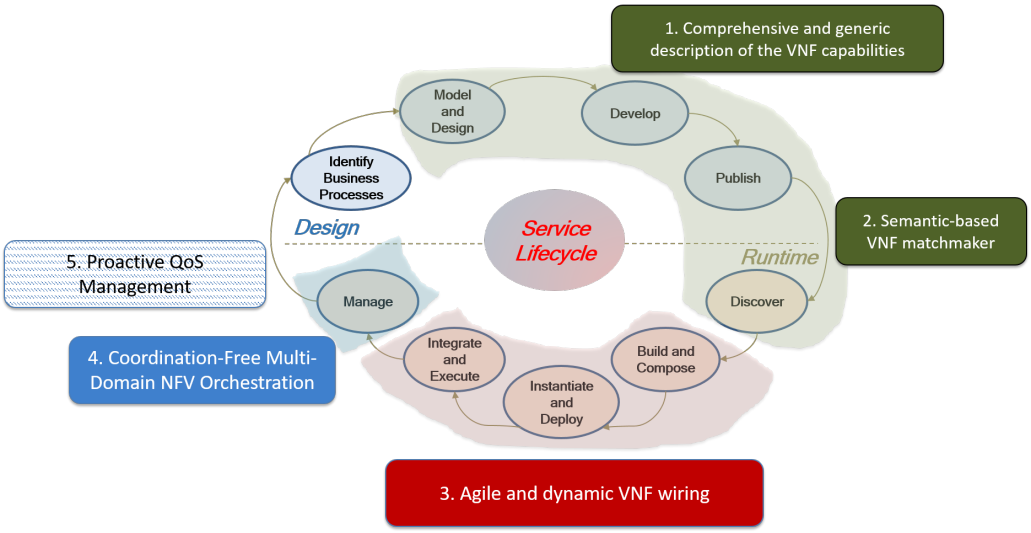


Figure 1.3: Overview of the contributions in relation to the service lifecycle steps.

As stated in Section 1.3, the contributions mainly focus on NFV. However, various illustration examples, taking into consideration other paradigms of next-generation networks (i.e., CDN, Edge and IoT), are discussed throughout the manuscript to show the genericity of the introduced approaches and procedures, as well as, their compliance with the cloud continuum ecosystem. The details of each one of these contributions are discussed in what follows:

- The first contribution ($CONTRIB1$) aims for generic VNF description and publication. It introduces a domain-independent VIrtualized networK functIoN ontoloGy (VIKING for short) that enables a comprehensive and generic description of the VNF capabilities from functional and non-functional perspectives.
- The second contribution ($CONTRIB2$) enables accurate and automatic VNF discovery. It proposes a semantic-based matchmaker that relies on VIKING to ensure the best matching between requested VNFs and published ones.

- The third contribution ($CONTRIB3$) set up agile VNF wiring (service composition) and instantiation (service deployment). It introduces a model, as well as, an exhaustive procedure to: (i) draw the composition of a set of elementary VNF, and (ii) instantiate and deploy the resulting Network Service (NS) in a terget environment.

- The fourth contribution ($CONTRIB4$) ensures coordination-free orchestration for consistent VNF reconfiguration at runtime. The proposed model supports a prospective NS deployment under multi-domain federations while taking into consideration the non-functional dependencies between the involved VNF in the orchestration.

- The fifth contribution ($CONTRIB5$) is on-going work. It lays the foundations of the research directions and future work discussed in Chapter 5. This work contribution aims to enable proactive QoS management of the services. It relies on machine learning models to predict any prospective degradation of the relevant QoS metrics. The goal is to identify and to execute the necessary actions (e.g., scale up the service, migrate the service) in order to maintain optimal QoS before even the degradation happens.

The main outcomes of this research work (2015 - 2023) are summarized in what follows:

- 6 co-supervised PhD students (graduated) and 1 supervised PhD student (ongoing)

- 16 (co-)supervised Master students

- 35 publications including 13 journal papers, 20 conference papers and 7 demonstration/short/workshop papers

- 4 running prototypes and software tools

- 7 research projects

It should be noted that part of this work was done within international collaborations with local and foreign laboratories and universities such as the University of Geoscience at Beijing in China, University Lyon 1 and University Paris Saclay in France, Concordia University and UQAM in Canada, IBM Almaden Research Center in USA, University of Tunis ElManar and University of Carthage in Tunisia and the University of Sydney in Australia, to cite a few.

The research results has been recognized by several prizes and awards:

- Doctoral and Research Supervision Award (Prime d'Encadrement Doctoral et de Recherche), 2020-2024.

- Best Research Paper Award. IEEE International Conference on Collaboration Technologies and Infrastructures, 2019.
- Best Demo Award. International Conference on Service Oriented Computing, 2018.
- Runner-up Demo Award. IEEE Consumer Communications and Networking Conference, 2017.
- Runner-up Demo Award. IEEE International Symposium on Local and Metropolitan Area Networks, 2016.

1.5 Manuscript structure

The rest of this manuscript is organized as follows: Chapter 2 discusses the two first research contributions on service design, publication and discovery. Chapter 3 presents the third contribution that addresses the NS service composition, instantiation and deployment. Chapter 4 discusses the last contributions on dynamic service management at runtime. Specifically, it proposes an approach to achieve coordination-free NS migration across multi- and hetergenous domains. Chapter 5 introduces a set of open issues and research directions that are relevant to this research study. It also describes the early obtained results on the on-going work to achieve proactive QoS management in this context. Finally, Chapter 6 concludes the manuscript.

The reader should note that this manuscript contains 2 appendixes. Appendix 1 details the list of publications while Appendix 2 details the teaching activities.

Chapter 2

Contributions on Service Design, Publication and Discovery

Contents

Service providers publish VNFs in dedicated marketplaces where network providers search VNFs and instantiate them according to a pre-established SLA. On top of being proprietary and specific to the service providers, the existing VNF description models include details on VNF deployment but fail to include VNF functional and non-functional specifictions. This alters an efficient selection of the most relevant VNFs and prevents full automation of the VNFs provisioning.

This Chapter discusses $CONTRIB1$ and $CONTRIB2$. It introduces a novel domain-independent VIrtualized Network FunctIoN ontoloGy (VIKING for short) for VNF description and publication in federated repositories [24]. It also proposes a semantic-based matchmaking algorithm to discover and select the most relevant VNFs that satisfy prospective VNF consumers' requests.

2.1 The state-of-the-art in NFV description and discovery

Several research efforts in service computing have been interested on service and user queries description. Different concepts were used among these works. However, the most important results were obtained when using semantics. Handling semantics in service discovery was largely investigated from two main matching perspectives: syntactic and semantic. The first relies on graph theory (e.g., Resource Description Framework [25] and DIANE Service Description [26]) while the second relies on ontologies (e.g., OWL-S [27] and WSMO [28]). Many works compare the syntactic ones (exemplified by information retrieval metrics) versus the semantic matching ones (exemplified by logic inference). The latter turns out more efficient than the former in terms of precision and recall. $CONTRIB1$ advocates for semantic matching for this work.

Generally speaking, in the networking domain, the use of semantics was widely used since the late eighties (e.g., [29], [30]). Semantic networks were first developed for artificial intelligence and machine translation. More broadly, the use of semantics in networks is done through declarative graphic representation that aims at representing knowledge and supports automated systems for reasoning about the knowledge. Some approaches are highly informal, but others are formally defined systems of logic. In particular, semantics was used to build and evolve network ontologies (e.g., [31]), retrieve information in networks (e.g., [32] for peer-to-peer networks), and network slicing and segmentation (e.g., [33]).

Considering the existing NFV Infrastructure (NFVI) such as OpenStack-Tacker[1] or OPNFV[2], the VNF Manager (VNFM) and the NFV Orchestrator (NFVO) require VNF-Descriptor (VNFD) for VNF instantiation and lifecycle management, and orchestration, respectively. However, the existing discovery approaches that rely on VNFD still in their early ages and much work has yet to be done for optimal VNFs

[1] `https://wiki.openstack.org/wiki/Tacker`

[2] `https://www.opnfv.org/`

provisioning. First, the existing discovery approaches remain specific to the owner providers due to the fact that the VNFD on which they rely are neither generic nor unified. Second, these VNF descriptions and publication models are not comprehensive. Indeed, they do include details on VNF deployment but fail to cover their related functional and non-functional specifications. Last but not least, consumers still need to manually select the required VNF rather than having an automated discovery mechanism. These noticed limitations are due to several reasons. In the wide landscape of VNF providers and technologies, a lack of a common understanding of VNF descriptions can be undoubtedly observed due to technologies' and providers' heterogeneity along with implicit knowledge leading to possible different interpretations. Consequently, consumers are obliged to parse a priori known sources to look for VNF candidates. This is time-consuming and often results in a very limited number of VNF candidates, not necessarily relevant with regard to consumers' initial needs. In the relevant literature, there were already few approaches that propose the use of semantics. This review mainly focus on the approaches that use semantic models at different phase(s) of the VNFs lifecycle introduced in Section 1.2. These work are classified into 2 categories: (i) the work done within the standardization bodies and research projects, and (ii) the work done within academia.

2.1.1 Standardization bodies and research projects

Besides the previously discussed ETSI VNFD model, one of the most known and used approaches is Topology and Orchestration Specification for Cloud Applications TOSCA-based, namely TOSCA-NFV [34]. TOSCA is a data model standard managed by the OASIS industry group that can be used to describe services' operations and requirements [35]. It also describes the way that services can be deployed and managed at runtime through the so-called management plans (workflows). TOSCA-NFV is the concrete implementation of the model applied to NFV for VNFs provisioning and management. However, TOSCA-NFV model assumes that the VNFs are already discovered. It proposes a model to describe topologies, dependencies, and relationships between virtual applications and simplifies the complexities of these services rather than describing the VNFs capabilities and requirements. Consequently, the main scope of TOSCA-NFV is to deliver orchestration and interoperability of VNFs. The same observation is valid for the IETF Service Function Chaining (SFC[3]) initiative. SCF in NFV setting relies on VNFD for VNFs selection and mainly focuses on the VNFs composition by matching the VNFs business (functional) operations [36].

T-NOVA is an EU-funded project [37]. It provides a VNF marketplace that: (1) helps VNF developers to describe and to store network functions, and (2) assists the consumers when browsing and selecting the network functions that match their needs. T-NOVA extends the ETSI NFV description model by applying business aspects from TMForum SID model [38]. It involves additional fields that enable business interac-

[3] `https://tools.ietf.org/html/rfc7665`

tion among the actors communicating through the T-NOVA Marketplace (e.g., SLA specification, pricing) besides the deployment information needed for the orchestrator to deploy the network services. The VNF/NS discovery process is conducted through the brokerage module [37] which permits consumers to search for VNFs/NSs while specifying their specific requirement in terms of network SLA.

Cloud4NFV [39] is a virtualized platform for VNF provisioning that aims to deliver NF-as-a-service to end customers. It is compliant with the ETSI NFV architectural specification with major contributions on the modeling and orchestration aspects. On one side, Cloud4NFV possesses frontend database that stores collections of VNFs along with high-level description (e.g., ID, name, description, location). On the other side, it handles a backend database that stores specific VNF information necessary for the VNF deployment and configuration. The platform provides only deployment and configuration information and the discovery process is manually done by the end customers.

2.1.2 Academic research works

In the academic literature, Hoyos and Rothenberg propose an NFV Ontology called NOn and a Semantic nFV Services (SnS) [40]. NOn enables the description of NFV as a high-level framework with reusable element descriptors. SnS is the concrete semantic application of NOn in the NFV domain. It uses NOn to create explicit service descriptors. It relies on agents from different domains to parse and evaluate NFV services capabilities. However, this approach imposes strong constraints on existing providers and assumes that they could support these agents on their domains. Furthermore, NOn only considers the functional capabilities of the resources.

In [41], the authors propose an ontology for NFV that enables describing the whole network resources including VNFs in terms of properties and relationships (dependencies). The resource description is considered then as reusable semantic concepts that can be used to construct additional rules for reasoning over the network. For instance, this could be useful to automate network topology design and deployment. This work focuses on the network engineering and integration efforts and does not cover the VNFs selection given a specific and precise user needs.

In [42], the authors propose a set of affinity and anti-affinity constraints useful for virtualized networks management. The validation of these rules is semantic-based. The addressed constraints are mainly related to service function chain requests. For instance, defining a valid VNFs placement strategy taking into consideration the network provider constraints and the chain request.

2.1.3 Synthesis

Table 2.1 sums up the capabilities of each considered work with regard to VNF decription, publication and discovery. The literature study shows that several works

tried to extend the VNFD proposed by ETSI with additional information using different approaches. Except VIKING and T-NOVA, none of them succeeded to cover the functional and non-functional properties of the VNFs, at the same time, in their proposed description models. However, the reader should note that the description of the non-functional properties in T-NOVA is limited. It only involves the business information (e.g., cost, SLA) necessary for interaction with other T-NOVA actors. When it comes to VNF publication, the conducted study highlights that most of the existing models require VNF publication in dedicated and proprietary repositories. T-NOVA and VIKING are the only approaches that do not impose any compatibility constraint on the provider side and enables NFV repositories federation. Since both rely on generic and unified semantic models, this eliminates dependencies related to technologies that would be used when offering the VNFs to prospective consumers. Finally, for the discovery phase, the study shows that all the existing works, except VIKING, either did not address the discovery process or propose simplistic procedures. These procedures are often characterized by the use of manual VNF selection or automated syntactic-based matchmaking between the offered VNFs and the required ones. In both cases, these discovery approaches require solid domain knowledge, are time-consuming and are not always efficient. This considerably decreases agility and cost-effectiveness that one may expect/require from a virtualized network ecosystem.

Table 2.1: VNF description and discovery in the relevant literature

Reference	**Description**		**Publication**	**Discovery**
	Functional properties	Non-Functional properties	Interoperability	Semantic matchmaking
ETSI VNFD	Yes	No	No	No
OASIS TOSCA NFV [34]	Yes	No	Yes	No
IETF SFC [36]	Yes	No	No	No
T-NOVA [37]	Yes	Partially	Yes	No
Cloud4NFV [39]	Yes	No	No	No
Hoyos et al. [40]	Yes	No	No	No
Oliver et al. [41]	Yes	No	No	No
Bouten et al. [42]	Yes	No	No	No
VIKING	Yes	Yes	No	Yes

2.2 Challenges and design considerations

NFV is at the crossroad of networking and service-oriented computing research fields for many reasons. It is advocated that VNF falls into the definition of IT services at large [43]. NFV aims at provisioning the network facilities through the VNF concept.

VNFs could be provisioned in the same way as any other kind of service such as telco services and Web services. In fact, Service-Oriented Architecture (SOA) principles (e.g., service abstraction, discoverability, and composability) [44] [45] could ensure the viability of an ecosystem of network services that are dynamically and flexibly provisioned, thereby coping with changeable network provider (i,e., the service consumer) needs and dynamic Quality of Service (QoS) requirements along with context conditions. Similarly, the VNF lifecycle phases are directly inspired from the service provisioning lifecycle detailed in [23] and discussed in Section 1.2. VNFs are first designed and developed before being published in appropriate repositories for prospective consumers. Prior to their deployment, VNFs are instantiated into the target network and are configured to be integrated as part of a specific topology. Once deployed, VNFs are executed and, when necessary, are subject to management considerations at runtime (e.g. scale up/down, migrate). The VNFs consumers rely on the information provided in the decriptors of the published VNFs to discover and select the most suitable ones that match the best to their needs and QoS requirements. This is critical step where several discovery aspects need to be considered such as the VNF's business functionality, as well as, the perfect matching between the VNF's technical requirements for its deployment and the target (hosting) environment capabilities.

Designing unified and comprehensive VNF discovery mechanisms with a service computing perspective is challenging. The fact is that, although their lifecycle remains fundamentally the same, the implementation of each provisioning phase might be specific. For illustration purposes, Table 2.2 shows the fundamental differences between VNFs and Web services operation. Novel description and discovery mechanisms should take these specificities and differences into consideration. For instance, unlike Web services [46], VNFs are not remotely invoked but are downloaded and executed locally, part of given network topology. In addition to inputs/outputs parameters, a VNF description should contain more elaborated technical details such as supported technologies and VNF settings. Moreover, the interaction is not limited to basic operations like it is the case with Web services [47], it should also include additional sophisticated operation management dedicated to each VNF. The existing service standards, studies, and frameworks do not address deployment-related issues. Moreover, they are process-driven while VNFs are more data-driven [47]. This makes existing WS-related solutions for description and discovery, including the ones that are based on semantics, inadequate for operation in the NFV setting.

Yet another challenge is related to the strong heterogeneity of the VNFs. Indeed, VNFs implement diverse and various network functions at either IP-level (e.g., firewall, NAT) or at application-level (e.g., video mixer, virtual IoT gateway) [48] [43]. Furthermore, the functionalities supported by the VNFs could belong to very different domains (e.g., Telco, IoT, cloud, big data, multimedia). This makes the design of a common description model difficult to design.

Finally, a last challenge is related to the nature of the potential environment where discovered VNF should be instantiated and deployed. In fact, the end hosting nodes

Table 2.2: Comparison between Web services and VNFs operating

Concept	Web Services	VNFs
Description	Document-oriented (WSDL)	Document-oriented (VNFD)
Specification	Limited interaction operations (RESTful)	Dedicated and extensible operations for management
Invocation	Remote Procedure Call (RPC)	Download and locally execute
Execution/ Management	Input/Output operations	Input/Output operations Additional technical or non-functional information

range from powerful computing servers to virtual machines and smartphones [49] [48]. These nodes have different and various capabilities (e.g. CPU, RAM, graphics resolution, bandwidth). This implies that an additional checking of the correct matching between the non-functional requirements of the disovered VNFs and the characteristics of their potential hostig nodes need to be integrated to the discovery procedure.

2.3 A semantic approach for VNF description

VIKING is an OWL-based (Ontology Web Language) ontology that allows describing VNFs. To design VIKING, the considered domain VIKING covers (namely, network function virtualization), for what VIKING will be used (namely, VNF description, publication, and discovery), and for what types of queries VIKING should provide answers (namely, similarity and correlation). The abstraction is then tackled by identifying the main common concepts shared by various application domains like CDNs, IoT, telco and 5G networks. *Concepts* are organized as a class hierarchy where abstract concepts are refined with more concrete ones specific to each domain application. They are also described with properties and connected to other concepts with semantic relations. To not reinvent the wheel, existing ontologies were reused, mainly for VNF deployment (e.g., [50]) and billing (e.g., [51]). Since concept refinement and instantiation are domain-dependent, they will be discussed in the illustrative use case presented in Section 2.5.1.

To assist VNF providers when creating comprehensive and consistent VNF descriptors, VIKING relies on OWL's reasoning principles. Fig 2.1 depicts VIKING's high-level skeleton that consists of two interrelated ontologies, namely VIKING-F and VIKING-NF, related to VNF's functional and non-functional properties, respectively.

On one hand, VIKING-F refers to the formal specification of what exactly the VNF can do. It revolves around 2 dimensions known as **Business** and **Model**. **Business** denotes the VNF's type, inputs (i.e., details about the content upon which the VNF will take effect along with other necessary details), and outputs (i.e., details about the

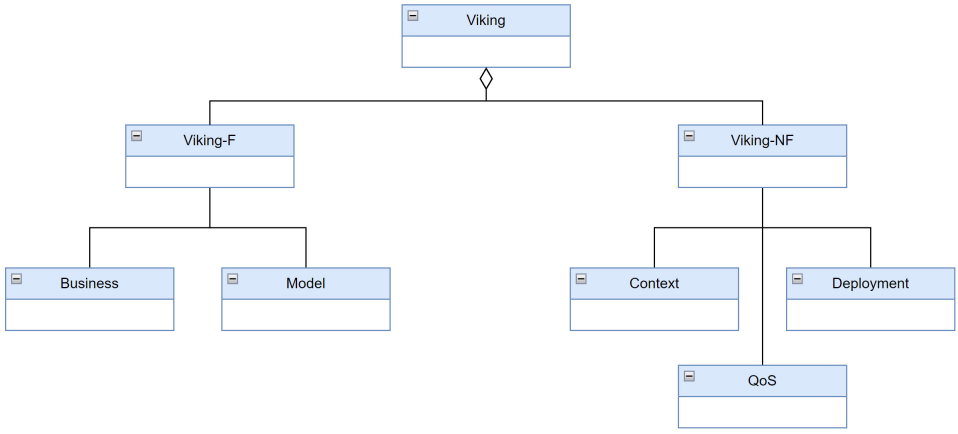


Figure 2.1: A high-level view of VIKING design

changes that will take place in the content). **Model** indicates the set of operations that ensure these inputs' conversion into outputs along with the related techniques and/or standards.

On the other hand, VIKING-NF refers to the formal specification of what exactly the VNF need/require for proper functioning. It revolves around 3 dimensions known as: **Context**, **QoS**, and **Deployment**. **Context** refers to the required runtime (e.g., operating system, specific libraries and/or system packages), as well as, device types (e.g., smartphones, TVs, desktops) upon which the VNF's outputs can be readable. **QoS** specifies common quality features offered by the VNF (e.g., response time, operation cost) and can be refined with specific-domain ones (e.g., surrogate servers locations for CDN, the bandwidth for 5G applications). Finally, **Deployment** involves VNF's artifacts and configuration parameters that are needed for VNF's execution.

Figure 2.2 shows a more detailed view of VIKING dimensions. Each dimension encompasses abstract conceptual areas that are instantiated using concrete concepts, which allows to produce a dedicated VIKING-F and VIKING-NF ontologies. These concepts, as well as, the relations between them are discussed in-depth in the rest of this Section.

2.3.1 VIKING-F ontology

As mentioned earlier, VNF's functional properties are specialized into **Business** and **Model** dimensions, described as follows.
Business. This dimension relies on existing classification standards (e.g., ISO/IEC[4],

[4]`https://www.iso.org/standard/68291.html`

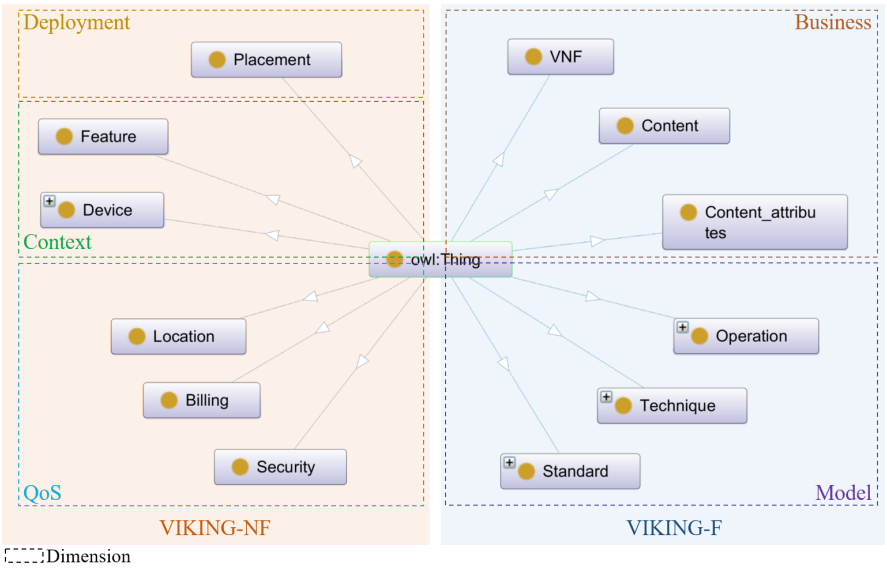


Figure 2.2: The core concepts of VIKING

ETSI NFV[5]) and leading service providers. Obviously, VNF design is always related to a target application domain. The VNF business description consists of three main concepts namely, *VNF*, *Content*, and *Content-Attribute* along with their semantic relations. *VNF* describes all necessary details on VNFs for advertisement and query-building purposes. Basically, *VNF* will be refined into concrete virtualized network functions for a given application domain. Since these functions share common concepts and semantic relations but also have their own technical specificities, they should be considered as concepts rather than as concept instances. *Content* refers to different domain-related artifact types manipulated by the VNFs. *Content-Attribute* indicates the type of content(s) supported by the VNF. More specifically, this concept represents the content's technical specification (e.g., required/supplied *Resolution* and *Quality*). It is worth noticing that *VNF*, *Content*, and *Content-Attribute* are semantically connected with relations namely, ***delivers*** between *VNF* and *Content*, and ***requires/supplies*** between *VNF* and *Content-Attribute*. The first relation states that any VNF provides some content while the second relation captures the input/output attributes upon which the VNF will act for specific content. In addition, for consistency purposes, cardinality restrictions (e.g., *at least one*) and axioms (e.g., *disjoint*) are specified, so that, concept instances are related to the right instance(s) and belong to the right concepts. To ensure a consistent instantiation of concepts, Semantic Web Rule Language (SWRL) rules (including axioms) help enforce restrictions on attribute values and semantic relations, as well. Hereafter, only SWRL rules referring to concepts are exemplefied, while those referring to instances will be discussed in Sec-

[5] `https://www.etsi.org/technologies-clusters/technologies/nfv`

Table 2.3: Relations in VIKING-F with the concept VNF

Dimension	Relation	(Target) Concept
Business	***delivers***	*Content*
	requires*/*supplies	*Content_attribute*
Model	***implements***	*Operation*
	supports	*Standard*
	applies	*Techniques*

tion 2.5.1. For instance, Equation 2.1 formally reflects the following statement: "*Any VNF (?x) that requires content-attribute (?y) should deliver specific content (?z)*".

$$\begin{aligned} & VNF(?x) \wedge \\ & \textbf{\textit{requires}}(?x, content\_attribute(?x, ?y)) \\ \rightarrow\ & \textbf{\textit{delivers}}(?x, content(?x, ?z)) \end{aligned} \tag{2.1}$$

Model. Technical aspects are relevant when making content exchangeable and adaptive in heterogeneous networks and devices (e.g., be able to read video in one digital encoding format different from the video original format). These aspects are thus considered to identify three main concepts related to **Model**, namely, *Operation*, *Standard*, and *Technique* linked to *VNF* through ***implements***, ***supports***, and ***applies*** relations, respectively. Specifically, *Operation* refers to how a VNF makes changes on some content(s) described in **Business**. *Standard* contains different standard(s) in the target application domain to foster content exchanges. *Technique* encompasses methods and procedures that a specialized VNF applies to make the necessary changes to the content. Furthermore, as in **Business**, restrictions and axioms such as "Any VNF can apply some techniques" might be defined. In addition, in some cases, mapping **Business** onto **Model**, or vice versa, is required (e.g., matching VNF requests with VNF advertisement). To this end, SWRL rules are defined to infer new semantic relations between instances during concept instantiation. For instance, Equation 2.2 formally reflects the following statement: "*Any VNF (?x) that implements operation (?y) and covers some device (?z) should supply a specific resolution (?u)*".

$$\begin{aligned} & VNF(?x) \wedge \\ & \textbf{\textit{implements}}(?x, operation(?x, ?y)) \wedge \\ & \textbf{\textit{covers}}(?x, device(?x, ?z)) \\ \rightarrow\ & \textbf{\textit{supplies}}(?x, resolution(?z, ?u)) \end{aligned} \tag{2.2}$$

Table 2.3 sums up the defined relations between the *VNF* concept and the rest of concepts in VIKING-F.

2.3.2 VIKING-NF ontology

VNF's non-functional properties are specialized into **QoS**, **Context**, and **Deployment** description parts, described as follows.

QoS. This dimension consists of three concepts namely, *Location*, *Billing*, and *Security-policy* linked to *VNF* through ***locates***, ***costs***, and ***enforces*** relations, respectively. *Location* refers to VNF's placement (e.g., network domain). *Billing* contains pricing models similar to those defined in cloud environments (e.g., time-based, volume-based, flat rate) [52]. Last but not least, *Security-policy* is related to *VNF* regardless of security mechanisms provided by the hosting platform. Indeed, the VNF should not depend on its hosting platform that can be itself a source of threats (e.g., malicious orchestrator or administrator) and thus should have its own security policies compliant to ETSI NFV SEC recommendation [53] [54]. *Security-policy* is refined into *Functional* and *Non-functional*, as depicted in Fig 2.3. The former is specialized into *Internal* and *External* to refer to the type of defense that the VNF puts in place against internal and external attacks. *Internal* and *External* both share *Authorization* that refers to control access to the VNF and its data along with capabilities, respectively, by an authorized entity in an authorized manner (e.g, Role- and Identity-based mechanisms). On top of this, *External* also covers *CIA* (stands for *Confidentiality*, *Integrity*, and *Availability*), *Authentication*, and *Trust*. *CIA* refers to protection mechanisms (e.g., cryptography, signature, and redundancy) for the VNF's capabilities along with raw/processed data against intrusions. *Authentication* refers to verification mechanisms (e.g., public key, certification, and password) for checking source's identity including traffic provenance. *Trust* refers to evaluation mechanisms (e.g., direct and collaborative trust-based) to establish trust relationships between VNFs. Finally, the latter is refined into *Auditability* and *Accountability* where *Auditability* refers to VNF examination techniques (e.g., knowledge- and behavior-based) while *Accountability* refers to internal tracking mechanisms (e.g., logging) to monitor the VNF's activities. It is worth noticing that this ontological model for capturing the VNF's security aspects can be easily enriched with more sophisticated ones based on the application domain.

Context. This dimension encompasses two main concepts, namely *Device* and *Feature*. *Device* refers to additional details related to surrounding/target appliances (e.g., hosting device) and *Feature* refers to options provided by the VNF (e.g., resize multimedia content in CDN, and switch communication protocol in IoT).

Deployment. This dimension integrates the already existing ETSI VNFD and enrich it with additional/complementary details on the resources to be allocated for VNF hosting and execution (e.g., number of required CPUs, amount of RAM). Specifically, *Placement* involves a URI of a remote enriched ETSI VNFD. It is worth noticing that *Placement* is a mandatory property during VNF discovery. Finally, it is well known that SWRL rules related to non-functional properties are domain-specific. Conse-

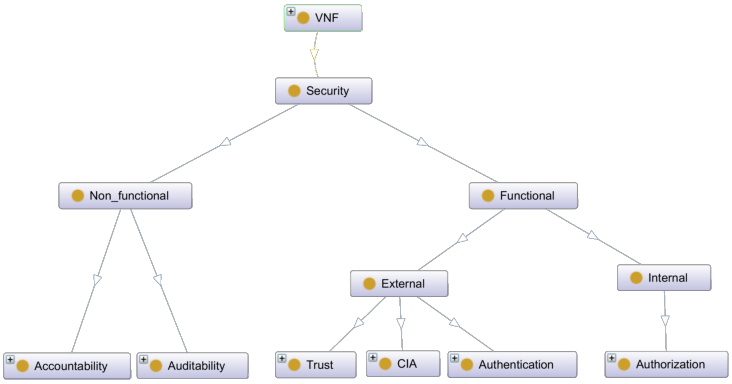


Figure 2.3: The security concept refinement in VIKING

Table 2.4: Relations in VIKING-NF with the concept VNF

Dimension	Relation	(Target) Concept
QoS	***locates***	*Location*
	costs	*Billing*
	enforces	*Security-policy*
Context	***covers***	*Device*
	offers	*Feature*
Deployment	***refers_to***	*Placement*

quently, they will be defined in Section 2.5.1.

Table 2.4 sums up the defined relations between the *VNF* concept and the rest of concepts in VIKING-NF.

2.4 A semantic-based model for VNF discovery

The proposed VNF discovery model is based on VIKING. It takes into consideration the enduser's preferences. Indeed, in accordance with the Web semantics principles, users can specify preferences among a set of functional and/or non-functional requirements, so that, the most appropriate discovered VNFs are selected. Table 2.5 contains the notation used to formalize user requirements and preferences. For the sake of simplicity, user requirements will be classified into three preference clusters namely, mandatory, high-requested, and optional.

VNF discovery process consists of two main steps both based on user preferences: user request building and semantic matchmaking. First, this process starts with assisting the user (i.e., a network provider) build his/her VNF requests in terms of what VNF capabilities are required. Afterward, it calls for a semantic matchmaking algorithm to seek for candidate VNFs offered by the providers in the appropriate repos-

Table 2.5: Notations to formalize user requirements and preferences

Symbol	Description
$\mathcal{RE}\mathcal{Q}^{F}_{i,j}$	User i's j^{th} functional requirement
$\mathcal{RE}\mathcal{Q}^{NF}_{i,k}$	User i's k^{th} non-functional requirement
$\mathsf{Pref}(F)$	Preference associated with functional requirements
$\mathsf{Pref}(NF)$	Preference associated with non-functional requirements
M-$\mathcal{CL}$	Mandatory-preference cluster
H-$\mathcal{CL}$	High requested-preference cluster
O-$\mathcal{CL}$	Optional-preference cluster
$\mathsf{Pref}(\mathcal{CL})$	Preference value associated with the preference cluster $\mathcal{CL}$
$\{\mathcal{RE}\mathcal{Q}^{F}_{i,j}\}^{\mathcal{CL}}$	Set of functional requirements labeled with the preference cluster $\mathcal{CL}$
$\{\mathcal{RE}\mathcal{Q}^{NF}_{i,k}\}^{\mathcal{CL}}$	Set of non-functional requirements labeled with the preference cluster $\mathcal{CL}$

itories. Finally, the discovery process provides the user with the most relevant VNFs based on their preferences. The algorithm and methodology that implement each one of these steps are detailed as follows.

User request building. After specifying all $\mathcal{RE}\mathcal{Q}^{F}_{i,j}$ and $\mathcal{RE}\mathcal{Q}^{NF}_{i,k}$ and labeling each requirement with either M- $\mathcal{CL}$, H- $\mathcal{CL}$, or O- $\mathcal{CL}$, the user i will define all preference values namely, $\mathsf{Pref}(F)$, $\mathsf{Pref}(NF)$, Pref(H- $\mathcal{CL}$), and Pref(O- $\mathcal{CL}$). Note that all $\mathcal{RE}\mathcal{Q}^{F}_{i,j}$ and $\mathcal{RE}\mathcal{Q}^{NF}_{i,k} \in$ M- $\mathcal{CL}$ will serve to discard irrelevant VNFs. Note that $\mathsf{Pref}(F)+\mathsf{Pref}(NF)+\mathsf{Pref}(\text{H-}\mathcal{CL})+\mathsf{Pref}(\text{O-}\mathcal{CL}) = 1$. To sum up, the user request ($\mathcal{UR}_i$) is a 2-tuple defined as follows:

$$\mathcal{UR}_i =< \{\mathcal{RE}\mathcal{Q}^{F}_{i,j}, \mathsf{Pref}(\mathcal{RE}\mathcal{Q}^{F}_{i,j})\}_{j=1,n}, \{\mathcal{RE}\mathcal{Q}^{NF}_{i,k}, \mathsf{Pref}(\mathcal{RE}\mathcal{Q}^{NF}_{i,k})\}_{k=1,m} > \quad (2.3)$$

Semantic matchmaking. Algorithm 1 reflects the matchmaking logics used to return the relevant set of candidate VNFs (Cand). It relies on VIKING when matching VNFs provided in a given repository (Rep), with user requirements.
Algorithm 1 consists of two types of matching, namely, `matchAll` (Line 2) and `matchSome` (Line 7). On one hand, since the set of all mandatory requirements (i.e., $\{\mathcal{RE}\mathcal{Q}^{F}_{i,j}\}^{\text{M-}\mathcal{CL}}$ and $\{\mathcal{RE}\mathcal{Q}^{NF}_{i,k}\}^{\text{M-}\mathcal{CL}}$) **should** be fulfilled, `matchAll` checks if the VNF exactly matches this set (i.e., true or false). Indeed, any VNF should be either kept or discarded in/from Cand depending on the result provided by `matchAll`. On the other hand, `matchSome` is applied to the rest of user requirements. For each VNF in Cand, `matchSome` returns a set of matched capabilities (M-Cap) that could be empty if there is no matching at all. Finally, the Cand list will be ranked based on VNF scores.

VNF matchmaking algorithm

```
1  VNF-MATCHMAKING(UR_i, Rep)
2    foreach VNF_i ∈ Rep do
3      if matchAll(VNF_i,{REQ^F_{i,j}}^{M-CL}, {REQ^{NF}_{i,k}}^{M-CL}) then
4        append(VNF_i, Cand);
5    foreach VNF_i ∈ Cand do
6      score(matchSome(VNF_i,{REQ^F_{i,j}}^{H-CL} , {REQ^{NF}_{i,k}}^{H-CL},
         {REQ^F_{i,j}}^{O-CL}, {REQ^{NF}_{i,k}}^{O-CL}), M-Cap)
7    rank(Cand )
```

2.5 Proof of concept

For validation and evaluation purposes, VIKING have been implemented in the context of Content Delivery Networks (CDN). CDN refers to a group of geographically distributed servers interacting with each other to enable fast content delivery to end-users over the Internet [55] [56]. Akamai[6], Swarmify[7] and Netflix Open Connect[8] are among the examples of CDN providers. In addition to the basic video services, CDN providers provision value-added content. Additional services such as media management (e.g., transcoding, ad insertion, and content protection), dynamic site acceleration, and front-end optimization are injected to the raw content prior to its delivery to end-users [57]. Enriching raw content with value-added services requires the provisioning of the so-called middleboxes in between the media server, that hosts the content, and the end-user [58]. Middleboxes implement network functions that perform the required transformations on the raw content depending on the needs (e.g., end-users requests and preferences) and context (e.g., location and monitor capabilities). The CDN carries raw content through these middleboxes to get the needed enrichment (e.g., inject location-based ads, apply user-specified filters) before serving it to the end-users.

In legacy environment, CDN middleboxes are provisioned as physical building blocks, at fixed network locations and on a proprietary and dedicated hardware [59]. The shortcomings of this traditional mode of middleboxes provisioning are widely known. It is subject to a lack of automation, dynamicity, and flexibility when deploying and managing the services. Actually, for planned events (e.g., worldwide sport events such as the Olympic games or the soccer world cup), CDN can anticipate the most common prospective user requests and, consequently, can adjust/predict and provision in advance the required middleboxes that are needed when processing these requests. However, in case of unplanned events, when, for instance, some videos go

[6] https://www.akamai.com/
[7] https://swarmify.com/
[8] https://openconnect.netflix.com/en/

viral, CDNs might get short in time and not being able to provide the appropriate middleboxes for a specific content/location.

CDN providers leverage NFV to re-architect their traditional system architecture and to provision value-added services as VNFs in agile and cost-effective way. According to the business model introduced in [60], CDNs could interact with third-party VNF providers to get the required middleboxes. Each VNF provider handles a set of repositories where VNFs are published and stored by owners through proprietary specifications and description model. Although most of the existing VNF descriptors include the VNFD information about the VNF instantiation and deployment in the target CDN, they barely describe the VNF's business functionality (e.g., video mixing, text translation). Furthermore, they completely fail in describing the end-user preferences and devices' capabilities (e.g., available bandwidth, supported format, terminal type, and screen size). This actually adds more complexity to the VNF selection process and affects the relevance of the considered middleboxes with regard to the real CDN needs. Specifically, identical VNFs offered by different providers could be described with different terms and characteristics. Worse still, VNFs could be described with identical terms but having totally different capabilities. For instance, the "media mixer" description might refer to a middlebox with text mixing capability or with sound/video mixing capability.

2.5.1 VIKING in content delivery networks

Table 2.6 depicts an excerpt of VIKING's refinement that results into VIKING-CDN. For **Business** and **Model**, *VNF* is refined into *VNF4CDN* that refers to a representative set of VNF capabilities like *VConverter* and *VMixer*. For instance, *VConverter* will be specialized into `VTranscoder`. VNFs implement operations acting upon content, support standards, and apply techniques. For instance, `VTranscoder` can operate on `Audio`'s and/or `Video`'s Format. For **Context**, *Feature* is refined into `Feature4CDN` while *Device* into *Device4CDN* then *SmartDevice* that is specialized into `Smartphone`, for instance.

Table 2.7 shows an excerpt of VIKING-CDN's population. There are 2 instance types. The first refers to CDN technologies (e.g., `ffmpeg` and `rotate`) while the second refers to CDN applications (e.g., `newsbroadcast` and `mooc`).

As stated in Section 2.3.1, SWRL rules are defined to infer new semantic relations between instances during VIKING-CDN's population. For instance, Equation 2.4 states that "*Any transcoder (?x) that delivers a video in some video format (?y), implements transmuxing operation*".

$$\begin{aligned} \texttt{Vtranscoder}(?x) \ \wedge \ \textbf{\textit{delivers}}(?x, \textit{video}) \ \wedge \ \texttt{Format}(\texttt{Video}, ?y) \\ \wedge \ \textbf{\textit{supplies}}(?x, ?y) \ \rightarrow \ \textbf{\textit{implements}}(?x, \texttt{O-Transmuxing}) \end{aligned} \tag{2.4}$$

A set of SWRL rules are defined as part of VIKING-CDN to ensure consistent instantiation of concepts. For instance, Equation 2.5, formally reflects the following

Table 2.6: Excerpt of VIKING's refinement

Dimension	VIKING	VIKING-CDN
Business	*VNF*	*VNF4CDN*, *VConverter*, `VTranscoder`
	Content	*Content4CDN*, `Audio`, `Video`
	Content-Attribute	*Content-Attribute4CDN*, `Format`, `Resolution`
Model	*Operation*	*Operation4CDN*, *Conversion*, `O-Transcoding`
Context	*Feature*	`Feature4CDN`
	Device	*Device4CDN*, *SmartDevice*, `Smartphone`

Table 2.7: Excerpt of VIKING-CDN's population

Dimension	Concept	Instances
Business	`VTranscoder`	`ffmpeg`, `vlc`
	`Audio`	`newsbroadcast`
	`Video`	`mooc`
	`Format`	`mp3`, `mp4`
Model	`O-Transcoding`	$\texttt{transcoding}_1$
Context	`Feature4CDN`	`rotate`, `resize`
	`Smartphone`	$\texttt{smartphone}_1$

statement: "*Any Transcoder (?x) that implements Transsizing operation and covers an iPhone 14 device should supply a 2048p resolution*".

$$\texttt{Vtranscoder}(?x) \wedge \textbf{\textit{implements}}(?x, \texttt{O-Transsizing}) \wedge \textbf{\textit{covers}}(?x, \texttt{iphone14}) \rightarrow \texttt{Resolution}(\texttt{2048p}) \wedge \textbf{\textit{supplies}}(?x, \texttt{2048p}) \tag{2.5}$$

To sum-up, any VNF that implements some CDN (injected) service (e.g. appliance, middlebox) should be described according to VIKING-CDN. This way of doing enables unifying the VNF's description, publication, and discovery procedures. Indeed, the procedures are common and homogenized regardless of any VNF provider. Consequently, the discovery procedures are simplified and could even be automated. Furthermore, cooperation and federation could then be envisaged between the providers.

2.5.2 The Mastermyr chest tool

As part of the validation process, a prototype implementing the proof-of-concept was developed. The name of the prototype is Mastermyr Chest. Its name refers to the tool chest that was found in Mastermyr[9] on the island of Gotland, Sweden in 1936. This chest contained more than two hundred objects used by Viking carpenters. Similarly, the Mastermyr Chest prototype contains several tools useful for VNFs description, publication, discovery, and so on. Figure 2.4 depicts Mastermyr Chest's tools, as well as, the main interactions between them. The reader should note that the Mastermyr Chest was designed and implemented with modular fashion so that it can be easily extended with additional tools in the future. VIKING-CDN was implemented with Protégé 2000 ontology editor[10] while Mastermyr Chest's tools were developed with Java. Source codes are available on the GitHub repository[11].

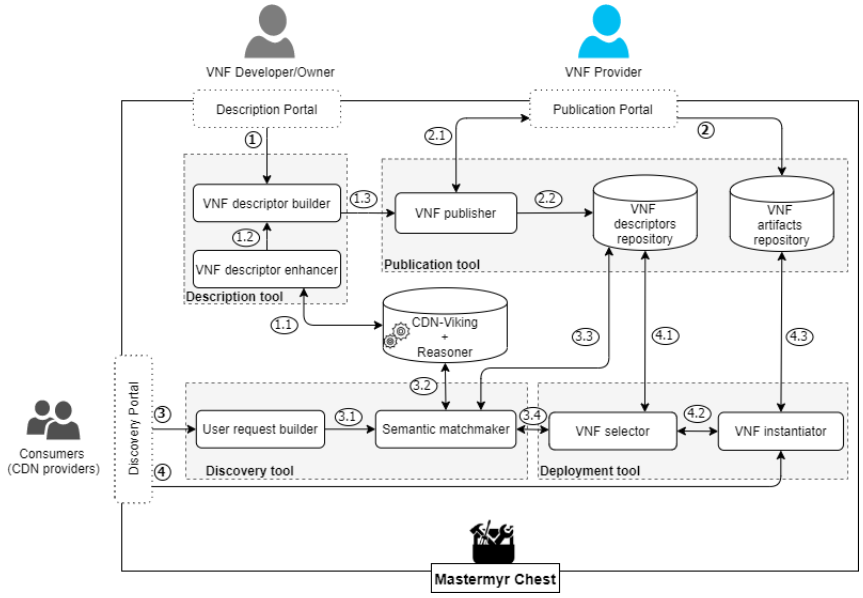


Figure 2.4: The Mastermyr chest tool box architecture

The *description tool* assists VNF developers (possibly, could be VNF owners) to semantically describe the VNFs that are relevant to the CDN context (action 1). VNF descriptors can be enriched with QoS details (e.g., price) by using the *VNF descriptor enhancer* (action 1.1). In accordance with the model introduced in Section 2.3, some information are mandatory and others are not. Then, the *VNF descriptor builder* generates the VIKING-CDN-compliant descriptors of the VNFs (action 1.2) and forwards them to the *VNF publisher* (action 1.3). The VNF descriptors are implemented as

[9] `https://en.wikipedia.org/wiki/M\%C3\%A4stermyr_chest`
[10] `https://protege.stanford.edu/`
[11] `https://github.com/NourelhoudaNouar/VNF-Description-Discovery`

OWL files. Snapshots of the *description tool* are shown in Figure 2.5.

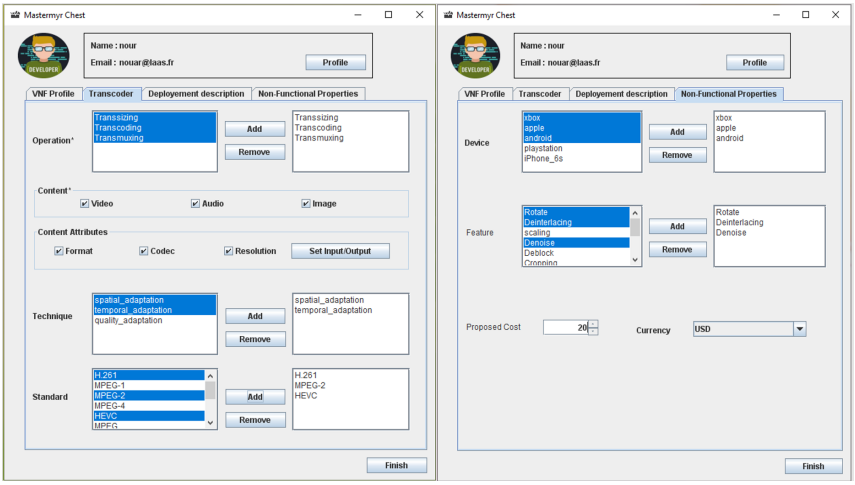


Figure 2.5: Snapshots of the description tool interfaces

The *publication tool* enables publishing the VNF artifacts (deployable) in the *VNF artifacts repository* to make them available to CDN providers (action 2). The *VNF publisher* requests the VNF artifacts' Unified Resource Identifier (URI) (action 2.1). After that, it annotates the VNF descriptor file with this URI and saves it in the *VNF descriptors repository* (action 2.2). For the current prototype, the concrete VNF artifacts, that implement one of the following middleboxes, are considered:

- **A multimedia mixer** that enables mixing several multimedia contents and returns a resulting content (e.g., adding voice to a video, adding ads banner to an image/video),
- **A multimedia compressor** that enables compressing the size and quality of multimedia content (e.g., degrading a high-definition video quality to save storage space or to decrease delivery time),
- **A multimedia transcoder** that converts original multimedia content to other formats using appropriate codecs (e.g., converting MP4 video to AVI).

The FFmpeg[12] open-source solution was used to implement these three middleboxes as VNFs. FFmpeg involves a suite of codecs, libraries and programs for handling video, audio, and other multimedia files and streams. Several and various FFmpeg instances with different configurations and packaging are implemented, in accordance with the characteristics and capabilities mentioned in the VNF descriptors. In turn, VNF providers store their instances into the *VNF artifacts repository* as Ubuntu-based virtual machines appliances.

[12] `https://www.ffmpeg.org/`

The *discovery tool* allows the VNF consumers (i.e., CDN providers in this specific case) to build their requests to calculate the matchmaking between required and offered VNFs (action 3). First, the *user request builder* assists the VNF consumers to define a proper and VIKING-CDN-compliant request based on their functional and non-functional needs and preferences. The request is then forwarded as required VNF descriptor to the *semantic matchmaker* (action 3.1). The *semantic matchmaker* enriches it through SWRL rules using VIKING's reasoner (action 3.2). Then, it applies Algorithm 1 to calculate the matching scores of the requested VNF with regard to the offered VNFs descriptors published in the *VNF descriptors repository* (action 3.3). Finally, the *semantic matchmaker* transmits the obtained ranked list to the *VNF selector* (action 3.4) (e.g., see the snapshot in Figure 2.5). The *semantic matchmaker* relies on OWL API[13] and Jena[14] plug-ins to parse OWL files and to perform the OWL reasoning.

The *deployment tool* enables the provisioning of a published VNF in a target network topology (action 4). First, the *VNF selector* downloads and parses its VNF descriptor. Obviously, following a discovery procedure, it selects and processes the VNFs descriptor with the highest matching score (action 4.1). Then, it forwards its URI to the *VNF instantiator* (action 4.2). The latter is responsible for downloading the VNFs, deploying it in the target CDN network, configuring it and integrating it to the existing topology (action 4.3).

2.6 Benchmark and evaluation

As for validation, a test collection for the CDN context was generated and a comprehensive comparative study was performed. The considered evaluation metrics are performance and robustness.

2.6.1 Test collection

To conduct experiments on VNF semantic discovery, the test collection creation are first created. This collection includes 3 items:

1. An exhaustive set of valid VIKING-compliant VNFDs ($\mathcal{D}$) that covers conversion, mixing, and/or compression functions in the CDN domain,

2. A set of test queries ($\mathcal{Q}$) that challenges the target semantic matchmakers in terms of false positive/negative outcomes,

3. A set of relevant VNFs per query ($\mathcal{V}_{\mathcal{Q}}$) that denotes all true positive outcomes.

These three items are detailed in the rest of this Section.

[13] `http://owlapi.sourceforge.net/`
[14] `https://jena.apache.org/documentation/ontology/`

2.6.1.1 Illustrative VNFDs for the CDN use case

To achieve an exhaustive coverage for $\mathcal{D}$ (i.e., all possible valid VNFDs), the following procedure is implemented. First, the association rules, listed in Table 2.8, are defined. These rules map VIKING's tree structure with a semantic parsing onto VIKING's hypergraph structure (see Figure 2.6) with syntactic parsing.

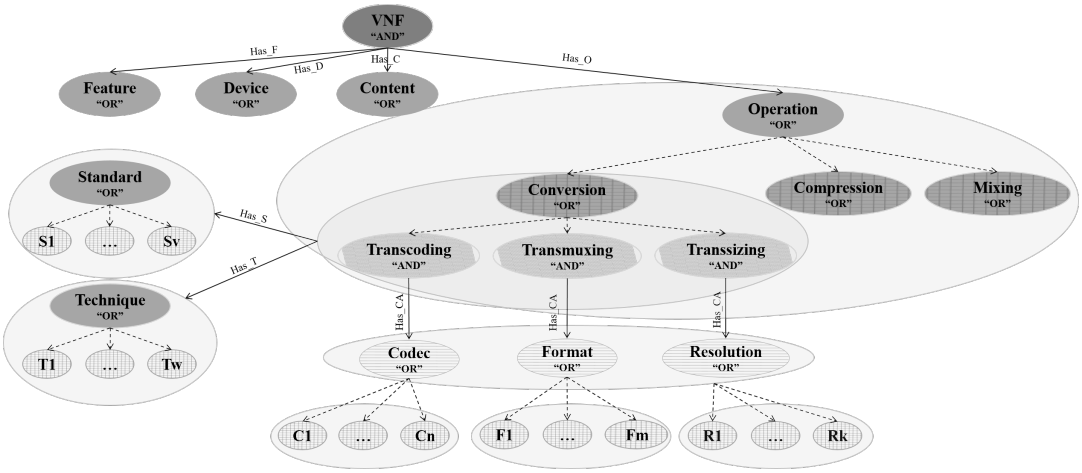


Figure 2.6: A partial syntactic representation of VIKING

Formally, this hypergraph $\mathcal{G}$ is defined as a 3-tuple $< \mathsf{T},\ \mathsf{NT},\ \mathsf{H} >$ where

- T denotes the multiset of terminal nodes that correspond to VIKING's instances where each set ($\mathsf{T}(\mathsf{c})$) refers to specific concept c.

- NT denotes the set of non-terminal nodes that correspond to VIKING's abstract and concrete concepts for CDN domain (e.g., Operation and Transmuxing). non-terminal are refined into OrNode and AndNode to represent inheritance (INH) and object properties (OP) among concepts (c_j), respectively. $\{\mathsf{R}_i\}_{i=1,3}$ reported in Table 2.8 indicate when and how to create OrNode and AndNode.

- H represents the multiset of labeled hyperedges that refers to a set of OrNode and AndNode. Formally, H is defined as a 2-tuple $< \mathsf{NT}, \mathsf{L}, 2^{\mathsf{NT}} >$ where L refers to a set of labels like OP and/or OR (Figure 2.6). $\{\mathsf{R}_i\}_{i=4,6}$ reported in Table 2.8 indicate when and how to create HyperEdge.

To generate $\mathcal{D}$, the well-known Depth First Search (DFS) is adapted so that possible valid VNFDs are built incrementally during visiting nodes. Algorithm 2 reflects this

[15] For AndNode, same as OrNode_2.

Table 2.8: Association rules mapping VIKING tree structure to VIKING hypergraph

Rule	Condition	Action
R_1	$\mathsf{c} \in \mathsf{NT}$ & $\mathsf{T}(\mathsf{c}) \neq \emptyset$	$\mathsf{createOrNode}_1(x, \mathsf{T}(x))$
R_2	$\lvert\{\mathsf{INH}(x, y_i)\}\rvert \geq 2$	$\mathsf{createOrNode}_2(x,\{y_i\})$
R_3	$\lvert\{\mathsf{OP}[x, y_i]\}\rvert \geq 2$	$\mathsf{createAndNode}(x, \{\mathsf{OP}, y_i\})$
R_4	$\mathsf{OP}[x, \mathsf{OrNode}_1(y_i, \{y_{i,j}\})]$	$\mathsf{createHyperEdge}(x, \{y_{i,j}\})$
R_5	$\mathsf{OrNode}(x, \mathsf{OrNode}_2(y_i, \mathsf{OP}, \{y_{i,j}\}))$	$\mathsf{createHyperEdge}(x, \{y_{i,j}\})$
R_6	$\mathsf{AndNode}(x, \{\mathsf{OP}, \mathsf{OrNode}_2(y)\})$[15]	$\mathsf{createHyperEdge}(x,$ $\{\mathsf{OP}, \mathsf{OrNode}_2(y)\})$

adaptation. This algorithm associates each visited node with some partial VNFD template where field names refer to all the parent nodes' names. In Lines 4-10, it splits OrNode's childs into a set of nodes, each corresponding to some combination of childs. In Lines 11-16, it concatenates AndNode's childs partial VNFD templates. As a result, $\mathcal{D}$ contains 695 VIKING-complaint VNFDs for the CDN use case.

Algorithm for generation of $\mathcal{D}$

```
1  Adapted-DFS(G, queue, current_t, template, D) ; // G is the hypergraph
   ; // queue is initialized to G.root
   ; // current_t refers to the current node in G
   ; // template refers to a certain VNFD template
   ; // D is the set of all VNFD produced
2    if queue ≠ ∅ then
3        current_t = queue.pop()
4        template.add(current_t)
5        if leafNode(current_t) then
6            D.add(template)
7        else if OrNode(current_t) then
8            foreach subset_t ∈ current_t do
9                queue.push(subset_t)
10                Adapted-DFS(G, queue, current_t, template, D)
11       else if AndNode(current_t) then
12           foreach child_t ∈ current_t do
13               queue.push(child_t)
14               Adapted-DFS(G, queue, current_t, template, D)
```

2.6.1.2 Sample queries

To challenge the target semantic matchmakers, $\mathcal{Q}$ is built by using 2 types of query ambiguity introduced in [61]. These authors classify Web queries into broad but clear and ambiguous where the former refers to queries that cover diverse subtopics but a narrow topic and the latter refers to queries that have more than one meaning. For experimentation purposes, broad-but-clear and ambiguous are refined as follows. In the first, user requirements are specified with implicit (or hidden) terms that can be inferred by VIKING's reasoner, only. In the second, user requirements include same naming for different instances but only the user can remove ambiguity. We, thus, expect that semantic matchmakers with broad-but-clear/ambiguous user queries as inputs and few knowledge about the user preferences, will provide VNFs candidates that would correspond to probably inconsistent interpretations (i.e., false positive/negative outcomes).

Table 2.9 depicts a sample of 2 test queries, each described by operation (Op), input(s) (I), and outputs (O). For each query, the source of ambiguity is identified. For instance, the knowledge of device permits to discard any incompatible resolution while the knowledge of I does not permit to determine Codec. In preparation of building $\mathcal{V}_\mathcal{Q}$ (i.e., the set of relevant VNFDs per query), the expected outcome for any q_i is expected as a VNF template referring to some required concepts (or instances) to satisfy the query like resolution and Codec.

Table 2.9: Sample of test queries for VIKING-CDN validation

Query Type	Query	Source	Expected Outcome
broad-but-clear	Op: Conversion I: Video O: XBOX one	O - Device	Op: Transsizing C: Video CA: Resolution
ambiguous	Op: Conversion I: AAC, AC3, MP3 O: DTS, EAC3	I - Codec/Format	Op: Transcoding C: Audio CA: Codec

2.6.1.3 VNFD relevance

To obtain $\mathcal{V}_\mathcal{Q}$, $\mathcal{D}$ is automatically annotated with binary relevance values. A VNF_i's relevance ($\mathcal{R}$) to a certain query (q_j) refers to at what extent this VNF's VNFD_i would

fullfil the VNF template (t_j) required to satisfy q_j (i.e., user satisfaction degree). Formally, Equation 2.6 computes $\mathcal{R}$ as follows.

$$\mathcal{R}^{q_j}(VNF_i) = \begin{cases} 1 & \text{if } |\{t_j.(\mathsf{c}_k|\mathsf{inst}_p) \in \mathsf{VNFD}_i\}| \geq \sigma_{\mathsf{c}_k},\ \forall c_k \in t_j \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where

- $t_j.(\mathsf{c}_k|\mathsf{inst}_p)$ refers to p^{th} instance of the concept c_k in the template t_j.
- σ_{c_k} denotes the minimum number of c_k's instances that should be included in any satisfactory VNF_i.

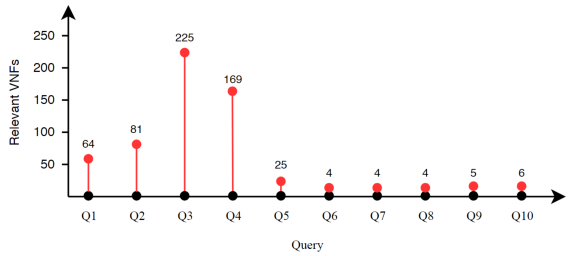


Figure 2.7: Probability distribution in $\mathcal{D}$ over $\mathcal{Q}$

This annotation was performed on every VNFD in $\mathcal{D}$ for all test queries ($\{q_j\}_{j=1,10}$). Figure 2.7 shows how VNFs annotated with 1 (see Equation 2.6) are distributed over $\mathcal{Q}$. A non-uniform distribution can be observed over $\mathcal{D}$. For instance, the set of generated VNFs that satisfy queries related to conversion ($\{Q_j\}_{j=1,5}$) is more represented than the other sets. This is due to a significant number of possible conversion-related capabilities compared to other operations like mixing.

2.6.2 Comparative study

To demonstrate the proposed approach's added-value, VIKING-based matchmaker (Mastermyr chest) is compared with a well-known OWL-S-based matchmaker like OWLS-MX [62]. To this end, both matchmakers are challenged using the same test collection and performance metrics.

2.6.2.1 OWLS-MX in brief

OWLS-MX relies on logic-based reasoning and information retrieval techniques (i.e., non-logic-based) for OWL-S service profile matching [62]. OWLS-MX puts emphasis on

service I/O matching and ignores service's preconditions and effects. OWLS-MX's reasoner can infer 5 semantic matching degrees between a given pair of query-service descriptions, listed as follows:

- exact: both I/O descriptions perfectly match with respect to logic-based equivalence of their formal semantics
- plug-in: all service I concepts are matched by more specific ones in the query
- subsumes: all service O concepts are more specific than those requested
- subsumed-by: all service O concepts are more general than those requested
- nearest-neighbor: all service I/O concepts have (to some degree) text similarity with those in the query

Compared to OWLS-MX, Mastermyr chest relies on VIKING's SRWL rules to logically verify/infer given/new semantic relations between concepts/instances used to describe VNF semantics. Thus, some mapping from VIKING to OWL-S with respect to the test collection is required. Table 2.10 depicts mapping rules where the rationale behind them is to translate VNF capability into OWL-S service business/profile (i.e., what VNF does) and model (i.e., how VNF works). Since a VNF can implement one or many operations, the corresponding service will be described in term of either atomic or composite process. Note that content and content attribute are both mapped onto I/O parameters. To put OWLS-MX and Mastermyr chest on the same equal footing, SWRL rules related to VIKING-F concepts are translated into OWL-S rules, exemplified with Equation 2.7.

Table 2.10: Mapping from VIKING to OWL-S

Dimension	**VIKING**	**OWL-S**
Business/Profile	*VNF* name	Service name
	Content	I/O parameter
	Content_ attribute	I/O parameter
Model	*Operation*(s)	Process (atomic or composite)

$$\begin{aligned}\text{Service}(?x) \wedge \texttt{presents}(?x, ?y) \wedge \texttt{has\_process}(?y, ?z) \\ \wedge\, \texttt{hasInput}(?z, mp4) \rightarrow \texttt{hasInput}(?z, Video)\end{aligned} \tag{2.7}$$

2.6.2.2 Performance metrics

To compare the proposed matchmaker and OWLS-MX, 3 well-known performance metrics in Semantic Web community are used namely, recall ($\mathcal{R}$), precision ($\mathcal{P}$), and response time ($\mathcal{RT}$). These metrics are defined as follows:

- $\mathcal{P}$ refers to the ratio between the number of true positive (TP) and the total number of **retrieved** VNFs including true positive and false positive (FP) (Equation 2.8).

$$\mathcal{P} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FP}} \tag{2.8}$$

- $\mathcal{R}$ refers to the ratio between the number of true positive and the number of **relevant** VNFs including true positive and false negative (FN) (Equation 2.9).

$$\mathcal{R} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FN}} \tag{2.9}$$

- $\mathcal{RT}$ indicates the amount of time necessary to get a response from the system following a discovery request sent by an end user.

2.6.2.3 Measurement and results interpretation

The experiments that challenge the proposed matchmaker and OWLS-MX with the same test collection and measure their respective performance in terms of precision, recall, and response time. The obtained results are discussed in what follows.

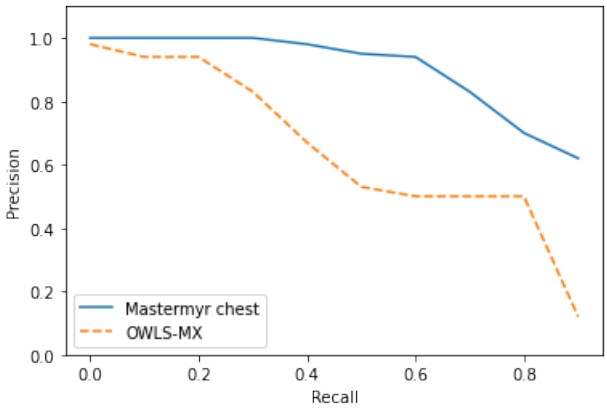


Figure 2.8: Recall/Precision ratio (VIKING matchmaker versus OWLS-MX)

Figure 2.8 shows better $\mathcal{R} - \mathcal{P}$ results for the proposed matchmaker than for OWLS-MX. Both matchmakers may return irrelevant VNFs to ambiguous queries but it turns out that the impact is more important for OWLS-MX. A possible explanation is that OWLS-MX only focuses on I/O matching and overlooks semantics that brings VNF's technical aspects like supported techniques, standards, and devices.

Figure 2.9 shows slightly better $\mathcal{RT}$ results for the proposed matchmaker than for OWLS-MX. The reader can also observe that $\mathcal{RT}$ varies depending on the VNF type. For instance, conversion type takes longer time than compression and mixing types. This is somehow expected due to the complexity (in terms of the number of concepts and instances along with semantic relations) in conversion type's semantics compared to the 2 other types. This complexity, thus, induces additional time for matching calculation. Overall, for this specific CDN use case, the proposed matchmaker remains more competitive than OWLS-MX. The reader should also note that the proposed matchmaker always responds within reasonable delays, even for the complex queries.

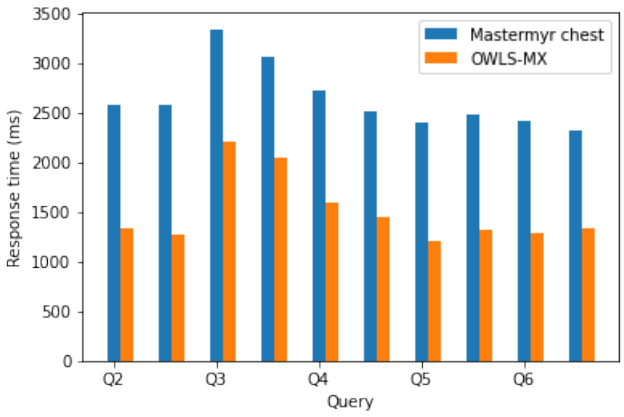
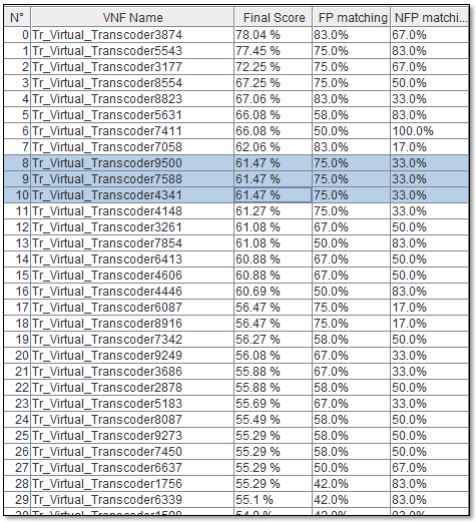


Figure 2.9: Response time measurement (VIKING matchmaker versus OWLS-MX)

2.6.3 Robustness evaluation

The robustness of the proposed matchmaker is also evaluated in terms of consistency. For a given query, the obtained VNFs and their associated calculated matching scores are examined and validated considering a specific predefined set of published VNFs. For instance, the matching results for the VTranscoder are shown in Figure 2.10. The reader should note that VNFs ranked from 8 to 10 have the same final score. This would mean that these VNFs are either identical or equivalent in terms of satisfying the query and its preferences. Let us parse the following capabilities:

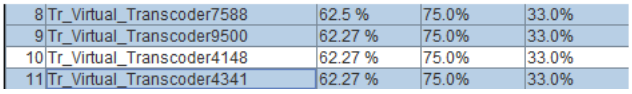
- $\{\mathsf{CAP}_{vt7588}{}^{\mathsf{F}}\}^{H}$=$\{$`0-Transmuxing`$\}$ & $\{\mathsf{CAP}_{vt7588}{}^{\mathsf{F}}\}^{O}$=$\{$`Std-MPEG_4`$\}$



N°	VNF Name	Final Score	FP matching	NFP matchi...
0	Tr_Virtual_Transcoder3874	78.04 %	83.0%	67.0%
1	Tr_Virtual_Transcoder5543	77.45 %	75.0%	83.0%
2	Tr_Virtual_Transcoder3177	72.25 %	75.0%	67.0%
3	Tr_Virtual_Transcoder8554	67.25 %	75.0%	50.0%
4	Tr_Virtual_Transcoder8823	67.06 %	83.0%	33.0%
5	Tr_Virtual_Transcoder5631	66.08 %	58.0%	83.0%
6	Tr_Virtual_Transcoder7411	66.08 %	50.0%	100.0%
7	Tr_Virtual_Transcoder7058	62.06 %	83.0%	17.0%
8	Tr_Virtual_Transcoder9500	61.47 %	75.0%	33.0%
9	Tr_Virtual_Transcoder7588	61.47 %	75.0%	33.0%
10	Tr_Virtual_Transcoder4341	61.47 %	75.0%	33.0%
11	Tr_Virtual_Transcoder4148	61.27 %	75.0%	33.0%
12	Tr_Virtual_Transcoder3261	61.08 %	67.0%	50.0%
13	Tr_Virtual_Transcoder7854	61.08 %	50.0%	83.0%
14	Tr_Virtual_Transcoder6413	60.88 %	67.0%	50.0%
15	Tr_Virtual_Transcoder4606	60.88 %	67.0%	50.0%
16	Tr_Virtual_Transcoder4446	60.69 %	50.0%	83.0%
17	Tr_Virtual_Transcoder6087	56.47 %	75.0%	17.0%
18	Tr_Virtual_Transcoder8916	56.47 %	75.0%	17.0%
19	Tr_Virtual_Transcoder7342	56.27 %	58.0%	50.0%
20	Tr_Virtual_Transcoder9249	56.08 %	67.0%	33.0%
21	Tr_Virtual_Transcoder3686	55.88 %	67.0%	33.0%
22	Tr_Virtual_Transcoder2878	55.88 %	58.0%	50.0%
23	Tr_Virtual_Transcoder5183	55.69 %	67.0%	33.0%
24	Tr_Virtual_Transcoder8087	55.49 %	58.0%	50.0%
25	Tr_Virtual_Transcoder9273	55.29 %	58.0%	50.0%
26	Tr_Virtual_Transcoder7450	55.29 %	58.0%	50.0%
27	Tr_Virtual_Transcoder6637	55.29 %	50.0%	67.0%
28	Tr_Virtual_Transcoder1756	55.29 %	42.0%	83.0%
29	Tr_Virtual_Transcoder6339	55.1 %	42.0%	83.0%

Figure 2.10: The total ranked list of the discovered VNFs

- $\{\mathsf{CAP}_{vt9500}{}^{\mathsf{F}}\}^{H}$=$\{$`O-Transcoding`$\}$ & $\{\mathsf{CAP}_{vt9500}{}^{\mathsf{F}}\}^{O}$=$\{$`Std-MPEG_2`$\}$
- $\{\mathsf{CAP}_{vt4341}{}^{\mathsf{F}}\}^{H}$=$\{$`O-Transcoding`$\}$ & $\{\mathsf{CAP}_{vt4341}{}^{\mathsf{F}}\}^{O}$=$\{$`Std-MPEG_4`$\}$



8	Tr_Virtual_Transcoder7588	62.5 %	75.0%	33.0%
9	Tr_Virtual_Transcoder9500	62.27 %	75.0%	33.0%
10	Tr_Virtual_Transcoder4148	62.27 %	75.0%	33.0%
11	Tr_Virtual_Transcoder4341	62.27 %	75.0%	33.0%

Figure 2.11: Excerpt of a list of relevant VNFs following a requirement change

The reader should note that `O-Transmuxing` and `O-Transcoding` are both functional requirements and are associated with the same preferences. This explains the equivalence between these VNFs. To evaluate robustness, the preference values for `O-Transcoding` are changed to O- $\mathcal{CL}$. Based on the updated list of scores shown in Figure 2.11, the reader should note that the final score changes. The list is refined to better satisfy the H- $\mathcal{CL}$'s requirements. Contrary to the previous list, the reader should note that *Tr_ Virtual_ Transcoder7588* is ranked before *Tr_ Virtual_ Transcoder9500* in the updated list.

Chapter 3

Contributions on Service Instantiation and Deployment

Contents

The European Telecommunications Standards Institute (ETSI) established the key reference architectural framework for Network Function Virtualization (NFV) [53]. The specification associated to this framework decribes the NFV instantiation, deployment and execution procedures. VNFs are instantiated into the target network and are configured to be integrated as part of given network topology. Once deployed, VNFs are executed and, when necessary, are subject to management considerations at runtime (e.g., scale-up/down and migrate). Furthermore, the ETSI specification introduces the Network Service (NS) concept while its associated framework enables its design, deployment and execution. According to ETSI [53], an NS is a composition of

VNFs. These VNFs are arranged as a set of functions according to a Network Connectivity Topology (NCT) or without any connectivity specification between them. Similar to Service Component Architecture (SCA) in service-oriented architecture specifications [44], the NS concept enables developers to provision complex and sophisticated network functions, made up of elementary VNFs, at a higher-level of abstraction. Furthermore, coupled with SDN, NFV provides higher flexibility in NS management. SDN enables flexible data forwarding among the VNFs. It places forwarding rules in network elements to transmit data through one of the possible paths prepared in advance by developers at design time [63]. This capability ensures dynamic control of VNF chains by facilitating data forwarding across VNFs. The forwarding elements route the traffic to follow an overlay path so that several VNFs are visited.

However, unlike SCA, ETSI NFV does not provide a specific control technique that dictates the execution order and the composition logic of VNFs in a given NS. This reflects negatively on the NS provisioning methodology and procedures. Indeed, at design time, developers must entirely rely on SDN, along with its inherent service function chaining concept, to design and configure the routing rules between the involved VNFs. Consequently, they have to foresee and plan all possible and practical composition scenarios in the NS. Moreover, they should instantiate, deploy, configure, and manage all these potential paths at runtime , including the ones that will rarely be or never be used. On top of being costly, tedious, and time-consuming, this way of proceeding contradicts NFV spirit, specifically virtualization, promoting agility and cost-effectiveness in networking applications. Yet another limitation concerns the routing path models supported by SDN. In fact, SDN allows only sequential service function chains to be provisioned where VNFs are tied in a linear way. Therefore, more complex and sophisticated chains are still not supported in the NS.

This Chapter discusses *CONTRIB*3. It proposes a novel approach that enables agile and dynamic NS provisioning as an extension to the ETSI NFV architectural framework [64]. Broadly speaking, this research initiative introduces a novel technical type of VNFs, called *routing* VNFs, with efficient re-configurable wiring capabilities for NS. This initiative distinguishes domain-specific aspects from the connectivity ones in the NS definition. Domain-specific aspects are implemented with regular VNFs, while connectivity aspects are supported by the newly introduced *routing* VNFs. By doing so, developers will be able to change and update the NS's composition logic at runtime, given specific criteria (e.g., message type, data size, QoS metrics).

3.1 Connectivity management in ETSI NFV architecture

ETSI provides developers with the NFV architectural framework as an open environment where VNFs can be interoperable [65]. This framework includes one fundamental building block, namely, NFV MANagement and Orchestration (MANO), that interfaces with three others, namely, Element Management (EM), Operations and Billing Support System (OSS/BSS), and Network Functions Virtualization Infras-

tructure (NFVI). MANO has three functional blocks namely, the Virtualized Infrastructure Manager (VIM), VNF Manager (VNFM), and NFV Orchestrator (NFVO), described as follows:

- VIM manages the NFVI resources (i.e., compute, storage, and networking) to provision VNF instances like provisioning hosting Virtual Machines (VM) based on compute/storage needs and migrating VMs, to cite a few.
- VNFM manages the VNFs life-cycle with the assistance of EM, like instantiating the VNFs and scaling up/down the VNFs, to cite a few.
- NFVO performs service and resources orchestration to satisfy OSS/BSS requests like activating or updating NS according to the procedures specified in the NS descriptor management.

ETSI NFV enables developers to model and compose VNFs as NS to deliver sophisticated network functions for prospective consumers such as CDN or telco providers [66]. Figure 3.1 depicts the core concepts related to connectivity, including the NS, as part of the ETSI NFV management system.

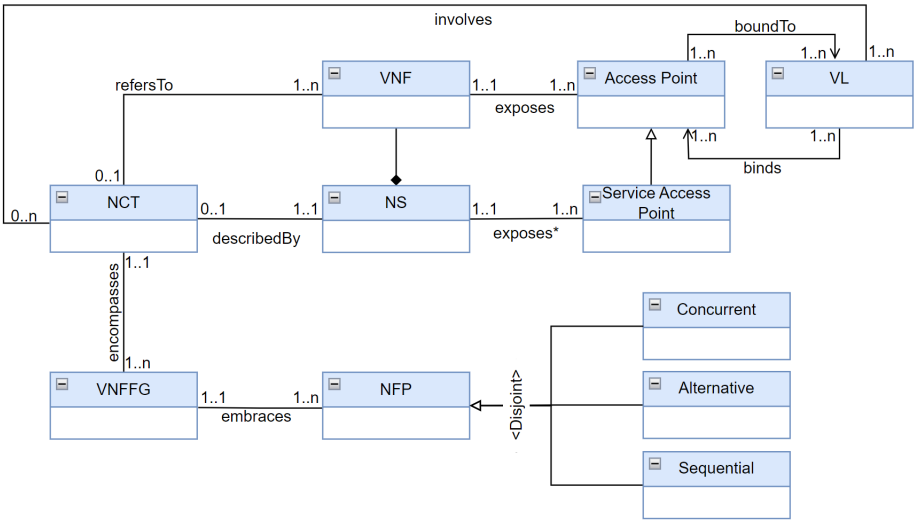


Figure 3.1: Connectivity modelling in ETSI NFV

The NS exposes service access points (i.e., specific connection endpoints) that define the NS interfaces. Similarly, the VNFs that composed the NS are bound to each other by Virtual Links (VL) that connect their associated access points. Altogether, it represents the Network Connectivity Topology (NCT). The NCT formalizes a high-logical view of connectivity between VNFs in the NS. The reader should note that constituent VNFs can be arranged in NS with unspecified connectivity between them [53].

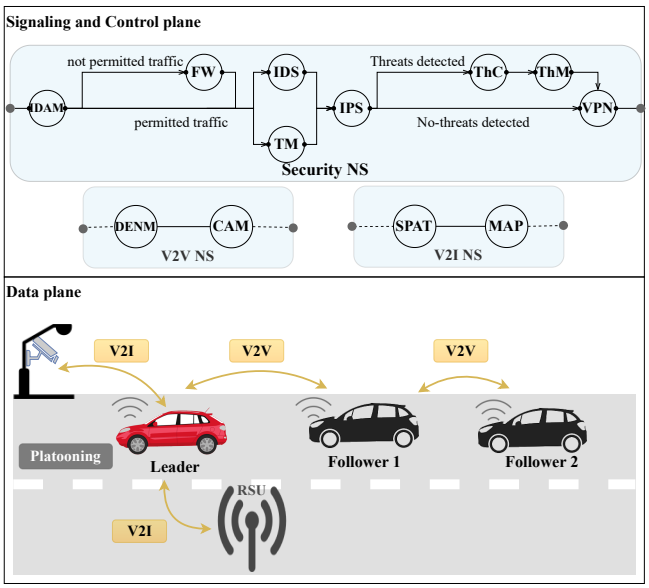


Figure 3.2: Network services and connectivity for vehicles platooning in 5G

NCT encompasses different VNF Forwarding Graphs (VNFFG). A VNFFG embraces one or more sequential, alternative, and/or concurrent Network Forwarding Paths (NFP) where a given NFP implements the concrete network path for the actual traffic flows in a VL. The NS concept, as well as, the related management entities are illustrated in the following motivating use case.

3.2 Motivating use case: Platooning of vehicles

Platooning technology made significant advances to mitigate traffic congestion and reduces vehicle emissions. For a safe and green journey, a platoon consists of a *leader* vehicle that controls the speed and direction and *follower* vehicles that fit with the *leader*'s movement in terms of acceleration and braking. The *leader* also communicates with the networking infrastructure to retrieve details about signalization and road conditions. Platooning relies on vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) communications [67] [68]. Notably, the fifth-generation (5G) of mobile telco networks turns out an excellent candidate to achieve this kind of communication. Indeed, the 5G network is expected to deliver ultra-low latencies and ultra-high reliability to enable efficient, safe, and reliable car platooning. To reinforce mobility and allow V2X communications, ETSI-compliant 5G specifications recommend NFV, SDN, Multi-access Edge Computing (MEC), and Next-Generation Protocols (NGP) as key concepts [12].

Figure 3.2 depicts the data plane and the signaling & control plane for car platooning.

The former refers to V2V and V2I communications. The latter involves three Network Services (NS), namely, V2V, V2I, and security. The V2V NS ensures the communication services between the vehicles and encompasses VNFs such as Decentralized Environmental Notification Messaging (DENM) and Cooperative Awareness Messaging (CAM). The V2I NS ensures communications between vehicles and infrastructure, and it encompasses VNFs such as Signal Phase and Timing Information (SPAT), road sign information, and road topology information (MAP). Finally, the security NS secures the whole system, including the V2V and V2I communications. While the V2I and V2V NSs reflect sequential "execution flows", the security NS provides several prospective "execution flows" depending on the case study and the data/request types to be secured. In practical terms, the traffic arrives first to the VNF named IDentity and Access Management (IDAM) that is responsible for verifying the vehicle's identity and distinguishes if the source vehicle is trustful (e.g., already part of the platoon) or not. The untrusted traffic must necessarily pass by the FireWall (FW) VNF before traveling through the rest of the NS. The FW VNF filters the network packets based on specific criteria (i.e., source, destination addresses, ports) and blocks the suspicious traffic. Next, the traffic simultaneously goes through the Intrusion Detection System (IDS) VNF and the Traffic Monitor (TM) VNF. The IDS VNF compares the received packets to a knowledge database to identify potential threats. In parallel, the TM VNF parses the network packets to detect any threats and/or malicious content. The execution results are then aggregated by the Intrusion Prevention System (IPS) VNF. When no threats are detected, the traffic is forwarded to the Virtual Private Network (VPN) VNF that makes safe tunneling with the network destination. Otherwise, the traffic should first travel through a specific network branch that consists of Threats Classifier (ThC) VNF and Threats Mitigation (ThM) VNF before reaching the VPN VNF. The ThC VNF classifies the intrusion within pre-defined classes/families of threats while the ThM VNF reduces the extent of the intrusion by either isolating or containing it until the problem is fixed.

Following the ETSI specification, provisioning the security NS necessarily implies deploying all the involved VNFs and configure the connectivity as depicted in the associated execution workflow. Therefore, NS developers should anticipate and cover all the execution alternatives. Specifically, this means that developers need to design, deploy and configure all the possible NFP and end up with heavy, complex, and static NCT handling during every single phase of the network entities' life-cycle. Wiring constituent VNFs could be easily done for simple NS but can turn out tedious and costly tasks for complex NSs with a considerable number of VNFs and sophisticated workflows. Furthermore, depending on the traffic, some of the pre-deployed/pre-configured wires will be rarely or never used. To address all these limitations, $CONTRIB3$ advocates for deploying a single execution alternative that could dynamically be reconfigured and adjusted, depending on the need (e.g., request type, data type), during runtime.

3.3 The state-of-the-art in dynamic management of networks' connectivity

In the relevant literature, several surveys (e.g., [69], [70], [71]) investigated and discussed the current challenges in NFV. Most of them argue that there is still a need for more appropriate and suitable VNF chains management. Moreover, they claim that VNF chains should be efficiently modeled, draw on dynamic business policies, and consider the network context to ensure its efficient operation and to better fit with the evolving users' requirements. With this regard, various approaches have been proposed to tackle these challenges focusing on enabling VNF dynamic chaining. Some of them rely on SDN to ensure the dynamicity of the control plane following a service function chain update, while others propose a fully NFV-based solution.

3.3.1 Literature review

In [72], the authors introduce an SDN orchestrator for service chains to handle congestion events and SLA violations. After collecting traffic statistics from switches, the orchestrator detects overloaded ones and then changes impacted service chain paths using other switches, if available.

In [73], the authors define chaining policies to build and manage dynamic service function chains (SFC). They propose a high-level specification language for policies and SFCs. Examples of policies are Service Level Agreement (SLA) requirements. However, SFCs are considered as a sequence of network functions.

In [74], the authors ensure dynamic chaining and flexible traffic routing for network functions in hybrid cloud/edge networks. This approach dynamically configures forwarding rules in SDN switches (OpenFlow) based on network traffic conditions (e.g., bandwidth) and the users' SLA.

In [75], the authors propose an orchestrator to dynamically provision and readjust VNF chains. This solution reuses existing VNFs to deal with the new users' requests. For cost-effectiveness purposes, the orchestrator periodically readjusts the chains by either migrating the VNFs over new datacenters or replacing them following the user requests.

In [76], the authors define some heuristic bandwidth-aware algorithm to create multiple dynamic service chains given a pre-defined set of allowed VNFs. This algorithm seeks acceptable solutions for service chaining where the candidate network links are less-heavily loaded than possible.

In [77], the authors enhance SDN orchestration to support dynamic VNF chaining as a preventive solution to service degradation. This prevention relies on network congestion prediction using traffic statistics. Afterward, alternative path redirections are computed so that chain recovery can be avoided.

In [78], the authors propose an SOA approach to discover and orchestrate VNFs using SDN-based traffic management automatically. When network service degrada-

tion is raised, the proposed solution adapts active paths with pre-established ones based on resource usage (e.g., switch load).

In [79], the authors propose the ESCAPE framework to build customized VNF chains in an SDN environment using Mininet, POX, ClickOS, and NetCONF tools. In addition, this framework supports traffic steering across VNFs based on specific policies and scrutinizes real-time information collected from running VNF instances.

3.3.2 Synthesis

The literature review highlights that most of the existing work rely on SDN to ensure a dynamic control plane following the update of service function chaining. Furthermore, it is remarkable that all the reviewed research work have tried to bring agility and dynamicity in VNF chains using various and different approaches (e.g., QoS monitoring, statistics, heuristics) and at several phases of their life-cycle (e.g., deployment and execution). None of the proposed solutions cover the entire life-cycle (i.e., from design to *on-the-fly* management at runtime). Moreover, all of them consider very simplistic VNF chains where VNFs are sequentially composed and do not tackle more complex and more sophisticated composition scenarios, as this might be the case in advanced NS.

Finally, the emerging Intent-Based Networks (IBN) research field could be relevant in the future. Simply put, IBN can be defined as a novel concept that incorporates: (i) SDN to manage the network control plane and (ii) machine learning and artificial intelligence to automate administration tasks across the network [80]. Recently published IBN-based work aim at automating network routes management during runtime (e.g., see [81], [82], [83]). However, there are still no studies currently that evaluate and compare business and operational costs of integrating IBN to the ecosystem of network providers. IBN is still at its early stages while NFV is becoming more broadly adopted nowadays by network providers (e.g., CISCO Systems, Netflix, Ubicity, VMware, Nokia, Intel Corporation, Huawei Technologies, IBM, Brocade, Vnomic, NetCracker).

3.4 Dynamic network wiring and execution

The proposed approach overcomes the limitations highlighted at the end of Section 3.2. It introduces an extended specification of the ETSI NFV reference architecture and promotes a novel and specific kind of "technical" VNF called *routing* VNF. The resultant architecture addresses the set of requirements that developers might face during design time, deployment or even the management changes that could be necessary to adjust the network's infrastructure.

3.4.1 Requirements and foundations

The conducted study of the ETSI NFV reference architecture enabled identifying a set of requirements that represent the necessary actions/characteristics for NS provisioning (i.e., design, deploy, execute, and manage). These requirements could be classified and associated to each phase of the NS life-cycle when considering MANO:

1. **NS description phase:** When designing the VNFFG (introduced in Section 3.1), all the necessary NFPs among a given number of VNFs need to be prepared in advance for a prospective operation, assuming that a complete/comprehensive knowledge is unfortunately unrealistic.

2. **NS deployment phase:** All the necessary elements, such as the virtual links that make up the previously mentioned NFPs, need to be established appropriately. This includes the virtual links that eventually will never be used and, thus, leads to excessive resources consumption.

3. **NS execution and management phases:** Not all the management operations are possible when maintaining the service delivery (e.g., adding new VNFs to the NS descriptor at runtime). Only the pre-defined paths that connect VNFs in the NS's descriptor can be subject to updates during these phases. This would overlook newly added paths.

To address the aformentioned requirements/limitations, the proposed approach advocates for flexible and dynamic wiring among VNFs in a given NS using the separation-of-concerns design principle to separate domain-specific aspects from connectivity aspects. Basically, the associated specification relies on three novel concepts, namely, *routing-AND*, *routing-OR*, and *routing-XOR*. Each concept supports wiring capabilities that enable the NS designer to define a given partial order among the domain-specific VNFs (i.e., all, some, or single). These novel wiring capabilities allow to obtain non-linear (i.e., tunable) NCTs. The prospective traffic when crossing VNFs triggers a particular *routing* logic as represented in Figure 3.3:

- *routing-AND* triggers **all** the "execution branches" by forwarding the entry traffic to all the outgoing VNFs.

- *routing-OR* triggers **some** of the "execution branches" based on traffic conditions, either conveyed in the traffic itself or reported by NS consumer, by forwarding the entry traffic to selected outgoing VNFs.

- *routing-XOR* triggers a **single** "execution branch" based on traffic conditions, either conveyed in the traffic itself or reported by NS consumer, by forwarding the entry traffic to the selected outgoing VNF.

Algorithm 3 implements the *routing* logic as follows. This algorithm first takes *routing* VNF id, NS descriptor, triggering condition, and traffic as inputs. Then,

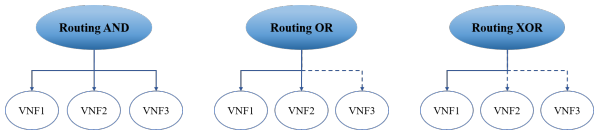


Figure 3.3: Network traffic representation per routing logic

it parses the NS descriptor to retrieve all NFPs ($\{\text{NFP}_k\}_{k=1,r}$) associated with the *routing* VNF id. Afterward, $\{\text{NFP}_k\}$ are split into 2 disjoint sets namely, $\{\text{NFP}_i\}_{i=1,r'}$ and $\{\text{NFP}_j\}_{j=r'+1,r}$ corresponding to desired wiring capabilities to enable when the triggering condition is true/false (lines 3-6 & lines 8-11, respectively). It is worth noticing that r' would be equal to r in case of *routing-AND*. Finally, the algorithm duplicates traffic as many as either $\{\text{NFP}_i\}$ or $\{\text{NFP}_j\}$. This results in multiple traffic to forward through these NFPs.

Routing logic algorithm

```
   Input: VNF id, NS descriptor, condition, traffic
 2 {NFP_k} = parse(VNF id, NS descriptor);
 4 if check(condition) then
       ; // whether routing-AND or routing-OR
 6     if {NFP_i} ≡ {NFP_k} ∨ {NFP_i} ⊂ {NFP_k} then
 8         foreach NFP_i do
10             forward(duplicate(traffic), NFP_i);
       ; // routing-XOR
12     else if |{NFP_i}| = 1 then
13         forward(traffic, NFP_i);
15 else
17     if |{NFP_j}| = 1 then
18         forward(traffic, NFP_j);
20     else
22         foreach NFP_j do
24             forward(duplicate(traffic), NFP_j);
```

In NFV setting, the proposed approach advocates for specific types of VNFs that would implement the afore-mentioned *routing* concepts. Figure 3.4 depicts a conceptualized view of VNF types. VNFs are specialized into *operative* for domain-specific and *routing* for connectivity. *Routing* VNFs could be implemented as either *swing* or *proxy* VNFs, both discussed in Section 3.5. Provisioning NSs endowed with *routing* VNFs would make their associated NCT generic and customizable for modeling all

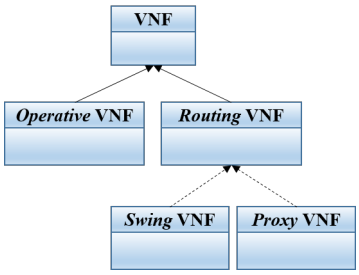


Figure 3.4: The novel introduced VNF types

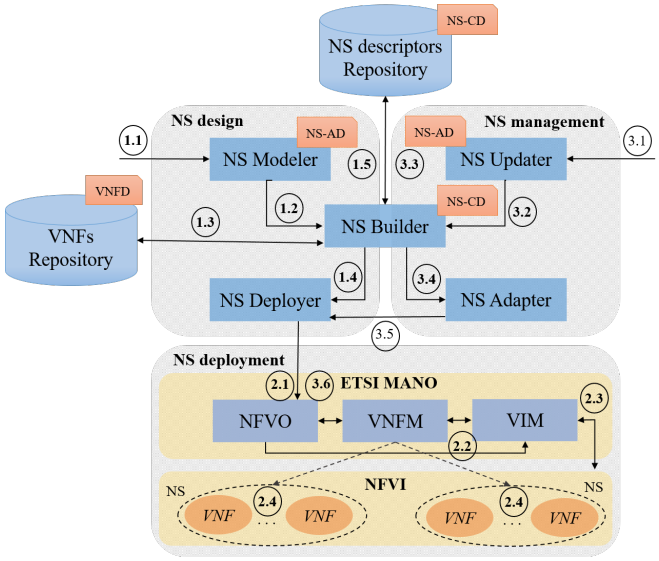


Figure 3.5: An overview of the proposed system architecture for NS provisioning

the possible connections in a given VNFFG. Thus, the NCT can be modified at runtime following user requirements, business policies, and/or network context. The following section discusses the way this NCT is defined, deployed, and managed at runtime.

3.4.2 High-level architecture

Figure 3.5 depicts an overview of the system architecture. This architecture aims to address the requirements discussed in Section 3.4.1. It ensures the proper end-to-end provisioning of NS made up of *operative* and *routing* VNFs. The proposed design consists of 3 main layers: `NS design`, `NS deployment`, and `NS management`. Each layer implements a given phase of the NS life-cycle:

- `NS design phase`: The NS Modeler enables designing a given NS with a set of

graphical elements to represent the NS's execution workflow, including *operative* VNFs, *routing* VNFs, as well as, the connections between them (Figure 3.5, (1.1)). The output is the NS Abstract Descriptor (NS-AD) that will be forwarded to the NS Builder ((1.2)). The latter parses the files and selects, from the VNFs Repository, the needed domain-specific VNFs that are bound to *operative* VNFs ((1.3)). Then, it produces the NS Concrete Descriptor (NS-CD) that refers to the concrete deployment details (e.g., VNF images), associated NFPs for a given VNFFG along with the constituent VNFs, and policies related to traffic management such as the classifiers. Finally, the NS Builder forwards the NS-CD to the NS Deployer ((1.4)) while saving a copy of it in the VNF Descriptors Repository ((1.5)).

- `NS deployment phase`: The NS deployment relies on the ETSI MANO. It uses its three components (i.e., NFVO, VNFM, and VIM) described in Section 3.1. Considering a given NS-CD, NFVO receives queries from the NS Deployer ((2.1)). Then, NFVO requests the VIM for necessary resources to instantiate both *operative* and *routing* VNFs ((2.2)). VIM communicates with NFVI to allocate and instantiate the requested resources ((2.3)). Moreover, the NFVO also asks VIM for the required networking resources to create and maintain virtual links (VLs) for all associated NFPs. VNFM will proceed with the concrete deployment of the VNFs over the NFVI resources ((2.4)). VNFM also interacts with NFVO to manage them.

- `NS management phase`: The architectural components for this phase ensure the service delivery and support the necessary on-the-fly updates. Indeed, at runtime, the NS designer or NS consumer may request functional changes such as adding/removing VNFs to a running NS, or simply modifying the wiring within a running NS ((3.1)). The NS Updater forwards the NS-AD associated with these requested changes to the NS Builder ((3.2)). The latter retrieves the corresponding NS-AD from the VNF Descriptors Repository ((3.3)) and revises it according to the necessary modifications. The revised NS-AD is then forwarded to the NS Adapter ((3.4)). The NS Adapter applies the new changes on the NS (e.g., add/delete NFPs) before notifying the NS Deployer to communicate with the ETSI MANO and reflect these changes on the running NS ((3.5)).

3.5 Proof of concept

To prove the feasibility of the proposed approach, a Proof-of-Concept (PoC) implementing the proposed architecture was carried out and integrated to the Mastermyr chest tool box introduced in Section 2.5.2. Moreover, a running prototype implementing the platooning use case, discussed in Section 3.2, was develped for validation purposes.

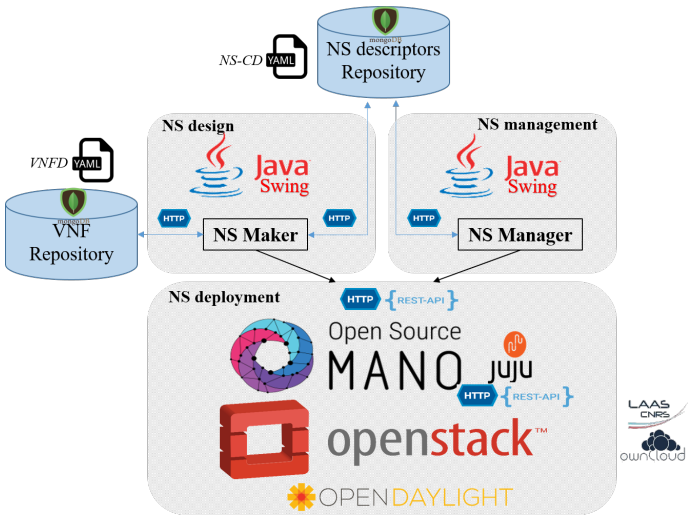


Figure 3.6: The DYVINE tool architecture

3.5.1 Software architecture

The designed PoC is called DYVINE (DYnamic forwarding graphs for VIrtual NEtwork functions for short). DYVINE's source code is available on a GitHub repository[1]. Figure 3.6 depicts DYVINE's software architecture that implements the system architecture discussed in Section 3.4.2. Basically, DYVINE tool enables NS provisioning while supporting the three main phases (i.e., design, deploy, and manage) of their life-cycle. DYVINE is coded with JAVA and incorporates two main modules, namely, NS Maker and NS Manager.

On the one hand, the NS Maker implements both NS Modeler and NS Builder at the NS Design layer. Thanks to the JAVA Swing framework, this module provides the NS developers with graphical interfaces that enable setting up NCT by drawing a given NS structure composed of *operative* VNFs and *routing* VNFs. Figure 3.7 shows a snapshot of DYVINE with the security NS's perspective representation of the platoon car use case. The generated descriptors are implemented with YAML and stored in MongoDB, a NoSQL database.

On the other hand, the NS Manager implements the architectural components at the NS Management layer namely, the NS Updater and the NS Adapter. This module displays the graphical representation of the NS to be updated from the NS descriptors repository and applies changes requested by the NS developer, to the corresponding NCT structure.

In addition to these modules, an integration of OSM[2], an open-source MANO

[1] `https://github.com/NourNouar/DYVINE`
[2] `https://osm.etsi.org/`

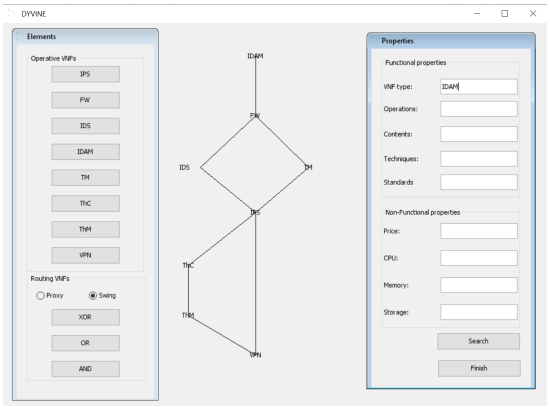


Figure 3.7: A snapshot of DYVINE depicting the security NS design

implementation, to DYVINE was performed to handle the concrete NS deployment, orchestration, and management. Specifically, the VNF Configuration Adapter (VCA) module of OSM was used to support Day-2 configurations and support the dynamic (at runtime) reconfiguration of the *routing* VNFs. In line with the foundations and the requirements discussed in Section 3.4.1, the VCA enables:

- Replacing the existing NFP identifier with a new one for *routing-XOR* VNFs,
- Updating a list of NFP identifiers for *routing-AND* VNFs, and
- Creating/deleting one or several NFP identifiers for *routing-OR* VNFs.

As for the infrastructure, both OpenStack[3] platform, to implement the NFVI, and OpenDaylight[4], with Open vSwitch (OVS) [84], are used to support the dynamic reshaping of the network control plane. OSM, OpenStack and, OpenDaylight are deployed over the CNRS-LAAS laboratory resources. More specifically, the NS design and NS management layers along with OSM were executed over a single laptop featured with Intel(R) Core(TM) i7-8650U CPU 1.90GHz under Ubuntu 18.04 LST. Regarding Openstack and Opendaylight, both were hosted on the Lab's private cloud. Finally, OVS (version 2.13.1) was used to support the Network Service Header (NSH) encapsulation with VXLAN tunnels.

Finally, the reader should note that DYVINE was integrated as part Mastermyr chest tool box. It supports the deployment tool depicted in Figure 2.4.

3.5.2 Running prototype

The developed prototype emulates the use case reported in Section 3.2. *Leader* vehicle and *attackers* were implemented as data streaming providers while the 4 *follower*

[3] `https://www.openstack.org/`
[4] `https://www.opendaylight.org/`

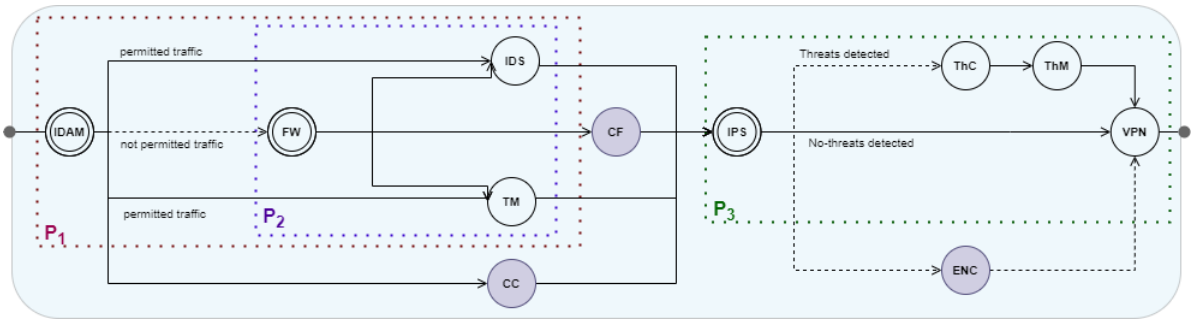


Figure 3.8: Option 1 - The security NS design with intrusive swing VNFs

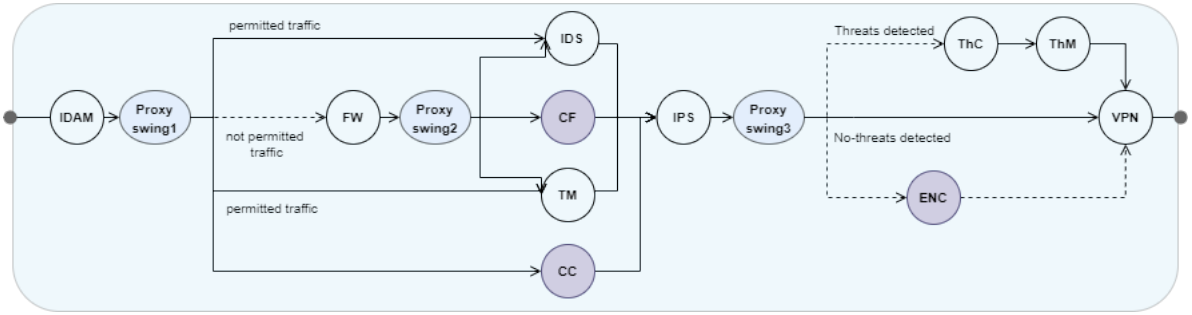


Figure 3.9: Option 2 - The security NS design with non-intrusive proxy VNFs

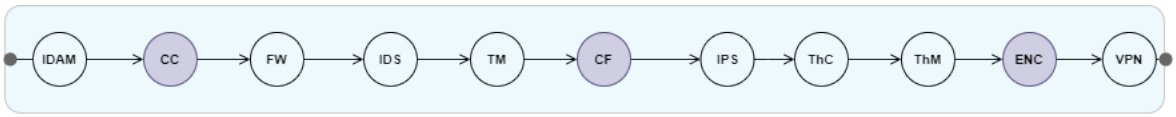


Figure 3.10: Option 3 - The security NS design with SFC (fully-SDN)

vehicles play the role of consumers, all deployed over OpenStack VMs. To enable V2V communications, 1 virtual network mapped onto the Lab's cloud network is created. Any network service including VNFs deployed over OpenStack VMs is connected to this virtual network through connection points. The reader should note that each VM has the following features: 1 CPU, 1 GB memory, and 10 GB storage.

When it comes to the security NS implementation, the three options were explored. While Option 1 relies on *routing* VNFs implemented through the so-called *swing* VNFs, Option 2 considers *proxy* VNFs. As a baseline, Option 3 consists of linear service function chaining and refers to a fully SDN-based approach for VNF wiring within the NS. The reader should note that Option 3 is referred to as the *fully-SDN* option. The VNFs source code associated to the security NS implementation are available on GitHub repository[5]. A live demo of the running prototype is available through the

[5] `https://github.com/NourNouar/Routing-VNFs`

following link: `https://youtu.be/7emJ6tnpvLE`.

On one side, *swing* VNFs support a static and particular logic. Each *swing* VNF implements a given *routing* gateway (i.e., *routing-XOR*, *routing-AND*, or *routing-OR*). With *swing* VNFs, depending on the network traffic conditions, (a) particular "execution branch(es)" will be triggered in the NS execution path. Specifically, *operative* VNFs, coupled with their respective intrusive *swing* VNF, are both encapsulated in the same VM. These particular VNFs are represented with a double ring in Figure 3.8.

On the other side, *proxy* VNFs are dynamic and implement abstract wiring mechanisms that could be reconfigured at will during runtime. Specifically, *proxy* VNFs encompass the 3 *swing* VNF types' logic and need to be configured to select the one to be used at deployment/running time (Figure 3.9). Packet exchanges between *operative* VNFs and *proxy* VNFs refer to Context Header 4 field's NSH Metadata (Type 1) [85]. This offers higher flexibility to NS at runtime.

Finally, for the need of benchmarking, a security NS's, wired with *fully-SDN* option is implemented (Figure 3.10). This solution was developed using OpenDayLight.

3.6 Validation and evaluation

Several experiments were performed to: (i) validate the proposed approach while demonstrating the agility and adaptive capabilities of the newly introduced VNFs and (ii) to evaluate its overhead and performance with regard to the classical routing approaches. To that end, the 3 implemented options presented in Section 3.5 were tested in terms of cost and performance.

3.6.1 Testbed settings

The experimental scenarios rely on the communication platooning strategies discussed in [86]. This work reports realistic communication rates between the *leader* and any prospective *follower*, as well as, between successive *followers* (i.e., 10 packets per second) with packet size varying over [200B, 1200B]. In addition, the experimental scenarios also refer to Denial of Service (DoS) attacks against *followers*. Any *attacker* floods the target *follower* with excessive data packets over time (i.e., 20 packets per second). The number of *attackers* varies over [1, 10], depending on the conducted experiment.

3.6.2 Validation scenarios

The first set of experiments aim at demonstrating the approach's effectiveness to support flexible NSs with dynamic wiring capabilities. To do so, the traffic throughput is measured over the security NS in case of attacks issued by a single *attacker*. This set of experiments exclusively focus on Option 1. To perform fine-grained analysis, the

throughput is probed over the NS's 3 partitions (dashed lines in Figure 3.8) namely, P_1, P_2, and P_3, each delimited with intrusive *swing* VNFs. Since P_i contains a set of VNFs, distinct throughput is measured over each VNF. 2 time-windows ($[0, t_1[$ & $[t_1, t_2]$), referring to pre- and post-attack, are defined. The reader should note that t_1 refers to start-time of attacks while t_2 indicates the end of both data streaming.

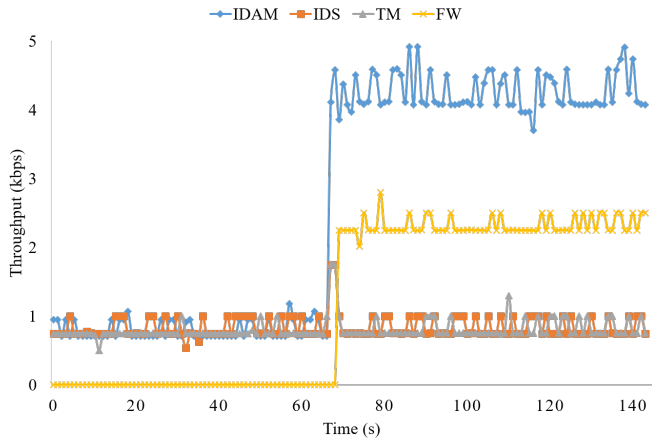


Figure 3.11: Throughput variation during pre- and post-attack (Option 1 - P_1)

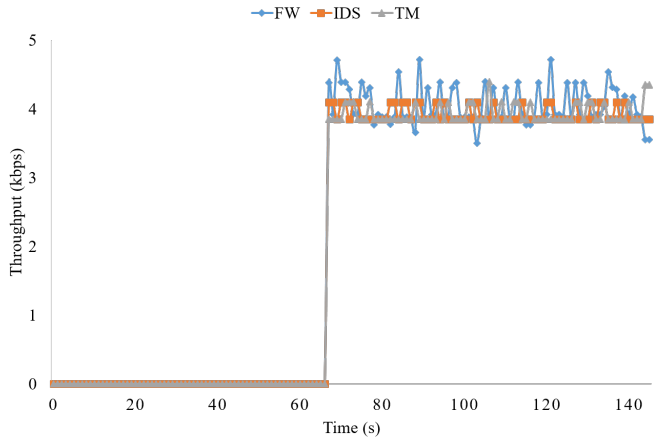


Figure 3.12: Throughput variation during pre- and post-attack (Option 1 - P_2)

Figure 3.11 shows that, during $[0, t_1[$, it is noticeable that P_1's VNFs have the same throughput except for FW where the traffic is null. During $[t_1, t_2]$, the throughput over IDAM and FW increases while it remains constant over IDS and TM. This demonstrates that IDAM enriched with *swing-OR* capability works properly by redirecting the *attacker*'s traffic to FW, only. Figure 3.12 highlights a null throughput

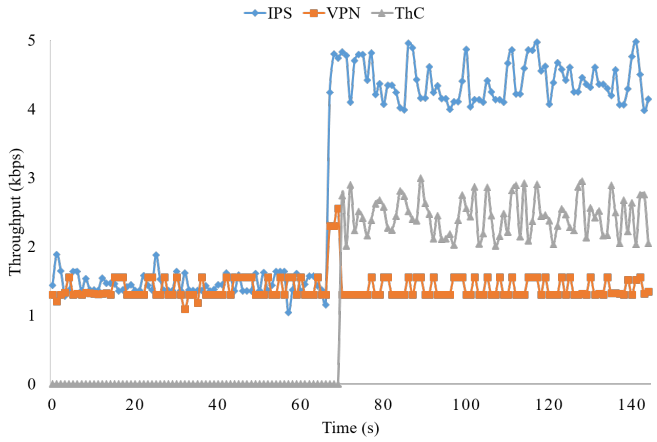


Figure 3.13: Throughput variation during pre- and post-attack (Option 1 - P_3)

for P_2's VNFs during pre-attack is observed whereas the throughput increases significantly during post-attack. This reveals that FW enriched with *swing-AND* capability works properly by passing on *attacker*'s traffic to both IDS and TM. Finally, Figure 3.13 depicts throughput over P_3's VNFs where it is similar for both IPS and VPN and null for ThC during $[0, t_1[$. During post-attack, the throughput over IPS and ThC increases significantly while it remains constant for VPN. This denotes that IPS, enriched with *swing-XOR* capability, fulfills its duty by sending *attacker*'s traffic to ThC. Finally, the 3 experiments highlight that the necessary switching time for the traffic route between pre- and post-attacks is instantaneous, confirming the agility and the dynamicity of the proposed approach.

3.6.3 Evaluation

The first evaluation experiment aims to estimate the average time related to one-way latency $\mathcal{L}$ (i.e., from the first bit sent to the last bit received) through a given NS. To measure $\mathcal{L}$, the network traffic is firstly emulated as a fixed number of packets (i.e., 100) with different sizes varying between 200B and 1200B including a certain rate of attacks varying from 10% to 100%. This traffic passes through the 3 options implementing the security NS. To avoid biases, the experiment are repeated 10 times before calculation of $\mathcal{L}$. Figure 3.14 represents the curves associated with $\mathcal{L}$(Option 1), $\mathcal{L}$(Option 2) and $\mathcal{L}$(*fully-SDN*) when increasing the rate of malicious packets. Overall, it is particularly noticeable that $\mathcal{L}$(Option 1) and $\mathcal{L}$(Option 2) achieve better performance than $\mathcal{L}$(*fully-SDN*) by $\approx$ 78% and $\approx$ 27%, respectively. When there are no attacks, $\mathcal{L}$(Option 1), $\mathcal{L}$(Option 2) and $\mathcal{L}$(*fully-SDN*) have $\approx$ 6.8ms, 8.2ms, 14ms, respectively. The difference is due to the fact that the packets need to go through various VNFs from one option to another. In fact, with regard to Option 1, the additional *proxy* VNFs in Option 2 increases latency. However, the reader should note that the latency

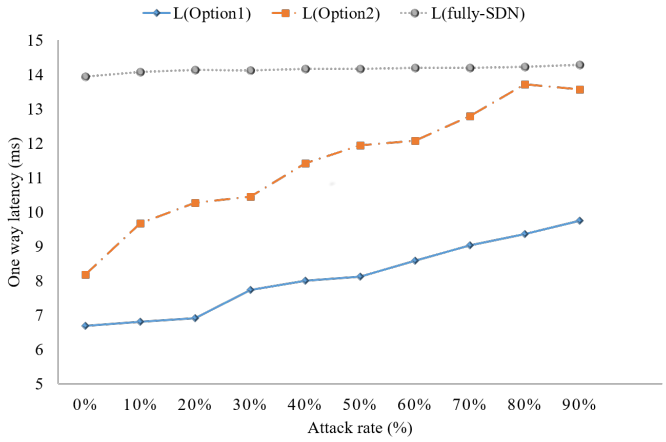


Figure 3.14: Variation of (one-way) latency while increasing the rate of attacks

is still less substantial for these 2 options with regard to *fully-SDN* option. Indeed, in Option 1 and Option 2, the packets go through the specific and selected path (i.e., limited number of VNFs), while in *fully-SDN* option, packets necessarily need to go through all VNFs in the NS, one after the other. When increasing the rate of attacks, $\mathcal{L}$(Option 1) increases slightly to reach $\approx$ 9ms whereas $\mathcal{L}$(Option 2) rises significantly to reach $\approx$ 13ms. This is due to additional processing for attack packets by intrusive *swing* VNFs and *swing* proxies, where, the latter dedicates more time to process wiring information communicated by *operative* VNF in packet meta-data. Also, the reader should note that $\mathcal{L}$(*fully-SDN*) remains constant because *fully-SDN* option deals with both normal and attack packets in the same way.

The second evaluation experiment aims at demonstrating the proposed approach's efficiency in terms of effective service continuity during when updating the network service during runtime. For instance, due to some changes in user requirements at runtime, existing/new VNFs can be removed/added from/to the network service. Say, any vehicle that requires additional defenses against tampering attack (i.e., false safety messages), malware (i.e., unreadable data format), and cipher attack (i.e., known plaintext) [87]. As counter-measures, the security NS should be updated with 3 new VNFs namely, Coordination Control (CC), Content Filter (CF), and Encryption (Enc), each against one of the afore-mentioned attacks (see Figure 3.8, Figure 3.9 and Figure 3.10). To enable this update, the security NS's options 1 & 2 proceed as described in Section 3.5. Regarding the security NS's *fully-SDN* option, any NS update requires OSM to redeploy this NS and SDN to reconnect all the VNFs. To quantify the Mean Time-To-Operation (MTTO), deployment time, for each new VNF, is measured in case of Option 1 and Option 2 and redeployment time for all VNFs in case of *fully-SDN* option. During this measurement, the number of new VNFs to be added to the security NS is varied. Figure 3.15 shows that the deployment/redeployment time

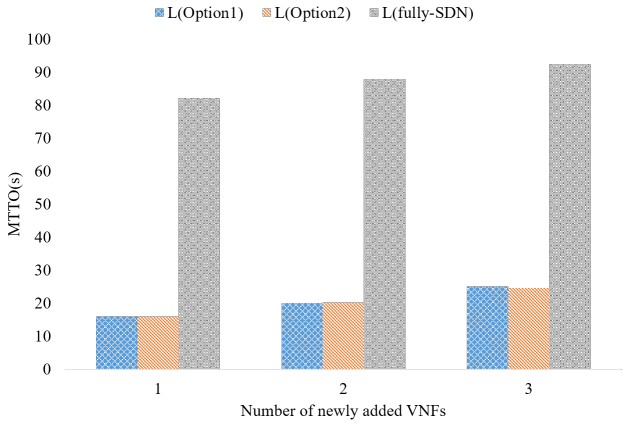


Figure 3.15: Mean Time-To-Operation comparison for the 3 deployment options

over the 3 Options rises in a linear way. It also should be noted that MTTO_1 and MTTO_2 are relatively equal. The ratio between $\mathsf{MTTO}_{i=1,2}$ and MTTO_3 approximates 72% (i.e., 6.5 times more). This can be explained by the fact that only the newly added VNFs are deployed while *wiring* VNFs operate to establish connectivity with these VNFs.

The last experiment aims to estimate the necessary time when some updates are required for a given NS. The update scenario consists of adding new number VNFs (VNF number varies from 1 to n) to the NS for each iteration. In this experiment, n was fixed to 3. For each Option, the NS was updated with 3 VNFs (see Figure 3.8, Figure 3.9 and Figure 3.10) namely,:

- Coordination Control (CC). This VNF checks coordination information in the car platooning after IDAM's operation.
- Content Filter (CF). This VNF analyzes the packets after content filtering from FW.
- Encryption (ENC). This VNF ensures a secure communication to reinforce IPS.

Figure 3.15 represents the histograms when increasing the number of newly added VNFs for the 3 implemented options. Generally speaking, this experiment shows a significant decrease using the proposed approach (Option 1 and Option 2) compared to *fully-SDN* option that requires from OSM to redo all necessary actions as a new NS. The time difference stands as high as $\approx$ 72%. Indeed, OSM takes from [80s-100s] to re-deploy the NS including amount of time to deploy all VNFs and establish the corresponding NCT to form the newly updated NS. For Option 1 and Option 2, the time required to update the NS falls into [18s - 22s]. This can be explained by the fact that only the newly added VNFs are deployed while *wiring* VNFs operate to establish connectivity with these VNFs.

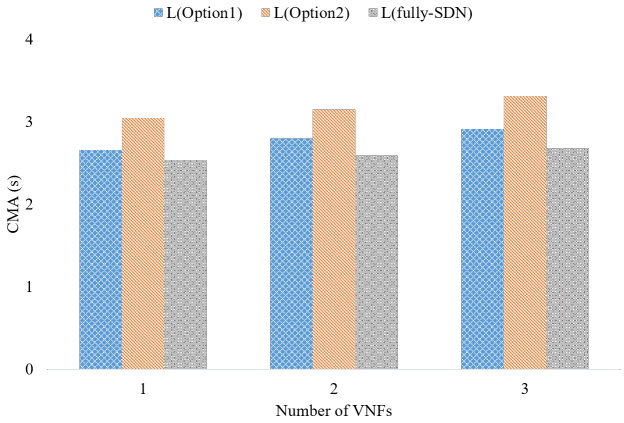


Figure 3.16: Cumulative moving average comparison for the 3 deployment options

The last experiment aims to assess the time required to create a new NFP for the different options. To this end, the cumulative moving average (CMA) is computed over 30 iterations per update when the new aforementioned VNFs are incrementally added to the NS. Figure 3.16 shows that CMA weakly increases. CMA(Option 1) and CMA(Option 2) with CMA(*fully-SDN*) are compared using ratio. The first approximates 7% while the second is close to 16%. Therefore, Option 1 gives acceptable CMA contrarily to Option 2.

3.6.4 Observations

The first lesson learned is that *routing* VNFs bring added value to the NS provisioning procedures. Compared to the service function chaining in SDN, the use of *routing* VNFs considerably reduces complexity in NFV connectivity management. Unlike SDN procedures, where developers need to handle all the network routes, they will be henceforth able to focus on particular branches of the same network. This induces a decrease in the network's operating cost. Generally speaking, the *routing* VNF concept fits well with virtualization principles. Actually, it brings cost-effectiveness and agility to the network. Also, it enables the dynamic update of the NS's composition logic given specific criteria (e.g., message type, data size, QoS metrics). This capability seems to be appropriate to tackle the emerging needs of the next-generation service providers (e.g., mobile communication systems) with challenging constraints on the network business model ecosystem. Examples of constraints are the mobility support (e.g., autonomous vehicles [12]), dynamic QoS management (e.g., adaptive and multicast streaming [88]), and the tactile Internet (e.g., haptic communications [89]), to cite just a few.

The second lesson learned refers to the way the *routing* VNFs can be implemented and the subsequent trade-off between flexibility and performance. At runtime, *swing*

VNFs (Option 1) show better performance in terms of end-to-end latency and the necessary update time with regard to *proxy* VNFs (Option 2). Nevertheless, *proxy* VNFs are easier to adopt and integrate during design time. The advantage of Option 2 is to take over the routing logic management and to not require any adaptation efforts on the network provider side. Regarding *swing* VNFs, their use would impose to integrate the routing logic (i.e., *routing-XOR*, *routing-OR*, *routing-OR*) within the *operative* VNFs. This adds more complexity to `NS` developers that will be in charge of injecting the routing logic within *operative* VNFs. Furthermore, this practice is not always possible because `NS` developers do not necessarily own all the VNFs that compose the `NS`. By contrast, *proxy* VNFs are compliant with the separation of concerns principle since domain-specific and connectivity aspects are decoupled in the `NS`. Also, this alleviates the `NS` design burden by delegating the *proxy* VNFs operation to the network provider. Although in line with the virtualized ecosystems' business model, this implementation option imposes upstream integration of this specific kind of VNFs as part of the technical provider's capabilities. To conclude with this, one can say that both alternatives are useful and it would be up to the `NS` developers to decide about the most suitable option for their specific needs (e.g., considered use case, supported data format, and SLA).

Chapter 4

Contributions on Service Management

Contents

Managing applications and services at runtime in dynamic and mobile environments is challenging. The management operations to be executed on the services aim at optimizing their use and performance given a predefined set of criteria/requirements. Basically, these requirements are expressed as SLA and it is the role of the service provider to manage these SLA and keep the inherent metrics within the

required operating range (e.g., resize a storage disk to increase the data storage capacity following a dataset enrichment, scale up an application instance to decrease the response time following a workload increase). The challenge lies in, not only on identifying the most appropriate management actions to be executed, but also maintaining the service availability and the service consistency when these actions are being executed. As for the service availability, the performed management actions should not interfere with the proper execution of the services, and consequently, the applications that rely on them. There are several techniques in the literature to maintain the service availability such as checkpointing, load balancing, and redundancy [90]. When it comes to the service consistency, the performed management actions should be able to determine the required cascading actions and ensure their proper execution while making sure to keep the system consistent all along. The reader should note that, as highlighted in Chapter 1, the target hosting environment, as well as, the considered services are highly distribued which significantly increase the risk of inconsistencies when operating management actions (e.g., reconfiguring a shared service, migrating a concurrent resource).

Specifically, for the case of ETSI NFV, the distributed NS are described through Forwarding Graphs (VNF-FG). As discussed in Section 3.1, VNF-FG embraces one or more sequential, alternative, and/or concurrent Network Forwarding Paths (NFP) where a given NFP implements the concrete network path for the actual traffic flows. In the cloud continuum, a multi-domain provisioning is common [91] [92]. This can be motivated by several reasons such as optimizing the performance, cost or security/privacy (e.g., provisioning the data-oriented services in one service provider domain and the compute-intensive services in another service provider domain, migrating data-oriented services to local domain prior to the processing of sensitive and confidential data). Similarly, in NFV, the VNFs that make up the NS are often distributed over multi-domain NFV infrastructures for the very same reasons [93]. Moreover, these VNFs are often shared between several and distinct NS at the same time. Each domain has specific and dedicated orchestrator that is responsible of: (i) composing the VNF with each other to support the end-to-end execution of the NS and (ii) supporting the required management operation on the VNFs during runtime (e.g., resize, migrate, restart).

Figure. 4.1 depicts an example of composite services with shared NS implementing basic funtionalities of a CDN provider, necessary to manage multimedia content (e.g. video mixer, video transcoder). The shared NS consists of 4 administrative domains. These domains has connections and dependencies between them to enable the end-to-end services (e.g., video streaming, video transcoding). The reader should note that CDN providers are highly dynamic with changing conditions during runtime (e.g., the number of end-users vary during runtime, random failures might happen in primary/-surrogate servers, some content might become viral and then popular). Consequently, the management actions are recurrent and are often inter-domains. From one side, each domain is managed by a local orchestrator and rely on shared VNFs and their

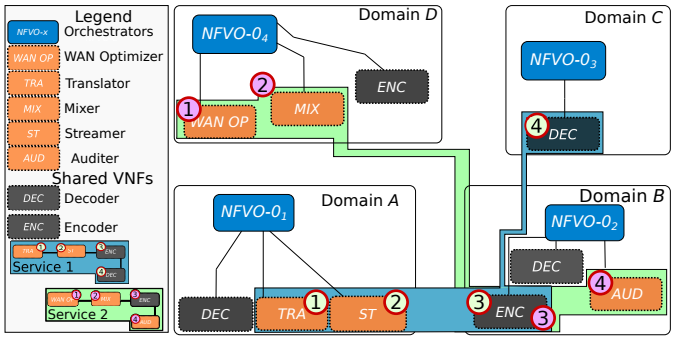


Figure 4.1: Example of a shared NS in a CDN provider

replicas. This generates dependencies in the associated VNF-FG. On the other side, as highlighted in Figure. 4.1, the VNF orchestrators are domain-specific. Their awareness remains local and it does not exceed their related domain limits. Indeed, the orchestrators have limited information, as they do not know the resources and the topologies operated by the other providers [53] [94]. Therefore, the several orchestrators that are involved in a multi-domain provisioning environment and sharing the same NS interact and collaborate with each other to coordinate the cascading management actions and to avoid any prospective inconsistencies in the associated VNF-FG. Generally speaking, when an orchestrator reconfigures a shared services' VNF-FG in a multi-domain environment, the orchestrator has to ensure that all orchestrators, that also use the shared service, have the same information (i.e., the replicas of the VNF-FG are consistent). With other terms, the reconfiguration of the VNF-FG implies the update of the data structure for classification rules and rendered service paths [95]. If one VNF-FG replica has different values than others, there is then a risk of inconsistencies created by conflicting operations made by different orchestrators. Specifically, Figure. 4.2 lists the several relevant concepts for the reconfiguration and shows the relatioships between them. Federations are composed of many domains that share services managed by different orchestrators. Services can be of type deicated and composite. The former have internal dependencies (e.g., VNFs) while the latter have internal dependencies but also external dependencies such as remote services. Dependencies can be both functional and non-functional. Moreover, each of these dependencies is connected with other VNFs or services, as specified in the VNF-FG associated with the service. The VNF-FG contains a list of matching attributes and connection points that detail how the dependencies are connected and how network traffic needs to be processed. Since services are shared, orchestrators need to handle the service's replicas, and by extension, the service's VNF-FG. Thus, reconfiguring a VNF-FG for shared services means that the replicas (and their dependencies) need to agree on the new value for both the matching attributes and connection points.

The problem of consistent VNF-FG reconfiguration involves updating or extending

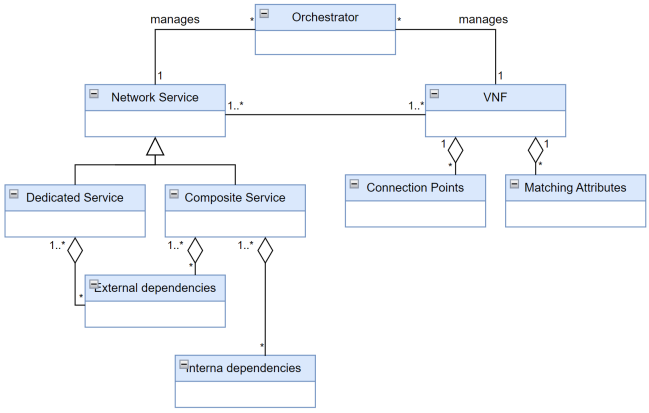


Figure 4.2: Distributed multi-domain orchestration system model

a VNF-FG responding to new demands [96] [97]. Since the VNF-FG defines a logical order of execution for each dependency of a service [98], the orchestrator can change the order of execution by updating connection points or classifier rules [65]. To ensure the other orchestrators apply the same updates, the orchestrator that changed the VNF-FG notifies the other orchestrators of the changes. In an ideal scenario, the orchestrators' VNF-FG replicas always achieve a consistent state. However, since there is no shared global reference between the orchestrator replicas, it is possible to concurrently update a shared VNF-FG with two conflicting updates. Moreover, since the orchestrators share services, the external dependencies involves non-functional dependencies to prevent unwanted effects of reconfiguration. For example, updating the connection point of a VNF-FG can optimize the latency in a given administrative domain but it may not be the case for replicas as these replicas can be used by other services. In other words, orchestrators can reject new changes from incoming replicas and decide only to reconfigure if their non-functional dependencies are satisfied after the reconfiguration. Thus, the combination of concurrent reconfiguration, limited knowledge, and non-functional dependencies introduces conflicts that must be fixed to achieve a consistent state at the end of reconfiguration.

This Chapter discusses $CONTRIB4$. it focuses on the consistent VNF-FG reconfiguration in multi-domain environments while considering VNF-FG non-functional dependencies [99]. The proposed approach consistently reconfigures the VNF-FG of a shared network service without a coordination phase between the orchestrators. Skipping the coordination phase paves the way for dynamic federations where orchestrators could dynamically join and leave the system. In addition, the proposed approach supports non-functional dependencies that have not been addressed so far in the literature for VNF-FG reconfiguration [100]. The orchestrators can negate ongoing reconfigurations for their VNF-FG replica unlike current orchestration approaches. Two

variants of the associated algorithm were designed, implemented and evaluated: (i) a preventive variant that ensures no transient inconsistent states could happen and (ii) a corrective variant that tolerates contingent inconsistent states, as reconfigurations are executed as soon as they arrive

4.1 Consistency models in distributed multi-domain orchestration

Consistency is a critical in distributed multi-domain orchestration. Generally speaking, distributed systems need to ensure consistency to manage concurrent operations on shared data [101]. Based on the literature study, the consistency could be either sequential or eventual. This Section discusses both types of consistency as well as the so-called strong eventual consistency, which reconcilite the trade-off between consistency guarantees and performance.

Sequential consistency was proposed to make the illusion of having one logic single-system image. Under sequential consistency, there is a single execution that follows a specific order. However, like in any other distributed systems, the execution in multi-domain federations run on top of multiple and independent nodes that do not have global knowledge. Since these nodes communicate over a faulty network, non-deterministic conditions bring conflicts when these nodes try to interact with each other and concurrently modify the state of a specific node. To prevent inconsistencies in sequential consistency process, the concept of solving consensus was introduced. Consensus is the convergence to a common value among all nodes [102]. It achieves sequential consistency for distributed systems. However, the high complexity of implementing consensus [103] and its low performance [104] make it a bottleneck for distributed systems. To improve performance on non-critical services, eventual consistency was proposed [105]. Eventual consistency is stated in Definition 1 introduced in [106]. Basically, eventual consistency guarantees that if no additional updates are made to a given data, all reads to that item will eventually return the last updated value [107].

Definition 1 (Eventual Consistency)
Eventual delivery: *An update delivered at some replica i is eventually delivered to all replicas: $\forall i, j : f \in c_i \implies \Diamond f \in c_j$, where f is an update, $\Diamond$ is a random and finite amount of time, and c_i, c_j are replicas of the same node c.*
Convergence: *Replicas that have delivered the same updates eventually reach an equivalent state: $\forall i, j : c_i = c_j \implies \Diamond s_i \equiv s_j$, where $\Diamond$ is a random and finite amount of time, s_i is the state of the ith replica.*
Termination: *All method executions terminate.*

Under eventual consistency, all nodes eventually converge, even though the model does not specify which value is eventually chosen neither the necessary time to achieve

the convergence [105]. The replicas can execute an operation without synchronizing *a priori* with other replicas, making data available at any given moment. Despite the consensus being moved off critical paths of services, reconciliation is still complex to achieve [108]. This led to the introduction of yet another concept, i.e., Strong Eventual Consistency. This concept is presented in Definition 2 introduced in [106].

Definition 2 (Strong Eventual Consistency (SEC))
An object is strongly eventually consistent if it is Eventually Consistent and:
Strong Convergence: *Replicas that have delivered the same updates have equivalent state:* $\forall i, j : c_i = c_j \implies s_i \equiv s_j$.

To achieve SEC, Conflict-Free Replicated Data Types (CRDTs) can ensure that there are no conflicts, hence, no need for consensus-based concurrency control [106]. CRDTs refers to the data types in which operations commute. In the literature, there is a portfolio of CRDTs for counters, registers, sets, and graphs that act as a building stone for more complex algorithms [108].

4.2 Consistent VNF forwarding graph reconfiguration in multi-domain environments

The design of an efficient and reliable consistent VNF-FG reconfiguration in federated environment is tedious. This has been done progressively. The early version only describes the consistent reconfiguration where many orchestrators manage shared services and their related VNF-FGs. Then, the final version supports the negation of the ongoing reconfigurations by the non-functional dependencies. Table 4.1 shows the relevant notation for the proposed cosnsitent reconfiguration model.

4.2.1 VNF forwarding graph reconfiguration

The reconfiguration of a VNF-FG involves changing the list of connection points and matching attributes [65]. This change can be done by updating the values either via changing a connection point or matching attribute and adding/removing more elements to the lists.

Let g be a VNF-FG that belongs to the set of the federation's VNF-FGs G. Each g has a pair of classifier rules c_g and rendered service path x_g. The classifier rule c_g has a list of matching attributes $[ma_1, ma_2, \ldots, ma_u]$. And the rendered service path x_g has a list of connection points $[p_1, p_2, \ldots, p_v]$. The function ϕ computes the state of the VNF-FG g as defined in Equation 4.1.

$$\phi(g) = (c_g, x_g) = ([ma_1, \ldots, ma_u], [p_1, \ldots, p_v]) \tag{4.1}$$

Each matching attribute $ma \in c_g$ has the protocol, IP, and ports to be visited by incoming traffic. The connection points $p \in x_g$ have both input and egress points. A

Table 4.1: The variables notations for the VNF-FG consistent reconfiguration model

Variable	**Meaning**
$O = \{o_1, o_2, \ldots\}$	The set of orchestrators
$G = \{g^1, g^2, \ldots\}$	The set of VNF-FGs each numbered
c_g	Classifier rule of VNF-FG g
x_g	Rendered service path of VNF-FG g
ma	Matching attribute, that belongs to a classifier rule
p	Connection point of a rendered service path
δ	The dependency relation
Δ	A VNF-FG Reconfiguration operation
L_θ	Pending operations for the the orchestrator θ
h_g^θ	The heap of accepted values for VNF-FG g from θ
$\Diamond$	A random and finite amount of time
s_i	State of the ith replica
$.id$	The identifier of a given entity, such as a VNF-FG
l_g^θ	The list of negated values for VNF-FG g from θ
ϵ	The initial value for a data structure.
$\phi(x_i)$	The state of the i-th VNF-FG x replica
r_o	Reply to update coming from orchestrator o
k_o	Orchestrator identifier
k_o^*	Highest identifier in the federation
x_i	the i-th replica of a VNF-FG x
$C(x_i)$	Causal history of the i-th replica of a VNF-FG x
$<_d$	The delivery order for reconfigurations
τ	The top operation

reconfiguration of the VNF-FG changes multiple values by either a matching attribute or connection points. The reconfiguration operation Δ between a pair of VNF-FGs g, g' in defined in Equation 4.2.

$$\begin{aligned}\Delta : g \rightarrow g' = \exists p \in x_g, p' \in x_{g'}, p.id = p'.id \mid \phi(p) \neq \phi(p') \\ \exists ma \in c_g, ma' \in c_{g'}, ma.id = ma'.id \\ \mid \phi(ma) \neq \phi(ma').\end{aligned} \tag{4.2}$$

The reconfiguration for a VNF-FG problem is named as VNF-FGR. For a *dedicated* service, the reconfiguration is trivial as the service's orchestrator manages all the resources and easily resolves conflicts. However, for *shared* services, the chief interest is that all affected orchestrator replicas have the same view after a reconfiguration. This is the goal of the consistent VNF Forwarding reconfiguration problem discussed in Section 4.2.2.

4.2.2 Consistent VNF forwarding graph reconfiguration

Following the orchestrator changes to a given copy of a *shared* VNF-FG $g \in G$ by updating the order of a connection point in the rendering service path, The other orchestrators will apply to all their related copies of the VNF-FG the very same changes (e.g., $\forall g' \in G | g.id = g'.id$). The consistent VNF-FG reconfiguration is formalized using Equation 4.3.

$$\Delta(g) : \forall g' \in G | g.id = g'.id \implies \phi(g) = \Box\phi(g). \tag{4.3}$$

where $\phi(g)$ is the state of the VNF-FG g and $\Box$ is a random and finite amount of time. An inconsistency occurs if, after a VNF-FG reconfiguration Δ on a *shared* service g, the state of a replica of g' do does not match. The consistent VNF-FG reconfiguration problem is named as CVNF-FGR.

Back to the CDN provider use case depicted in Figure. 4.1, due to a sudden surge in a *shared* service containing a *decoder* VNF, areconfiguration action is necessary to maintain the service response time within the required SLA. However, this service has been replicated in a multi-domain federation with three different providers that have three different orchestrators o_1, o_2, o_3. Consequently, the 3 involved orchestrators must reconfigure it along with its VNF-FG. The performed management operation is illustrated through scenario (A) in Figure. 4.3. The target management operation consists on changing the connection point of the *shared decoder* VNF represented by value a (e.g., changing the input point). First, all the VNF-FG replicas start in the same state o (Figure. 4.3, A-step 1). Then, the second orchestrator o_2 updates its VNF-FG replica with value a and sends a notification to other orchestrators (A-step 2). Finally, the other orchestrators receive the notification and also update their VNF-FG replicas (A-steps 3-4). At the end of the reconfiguration, all the replicas have the same value a.

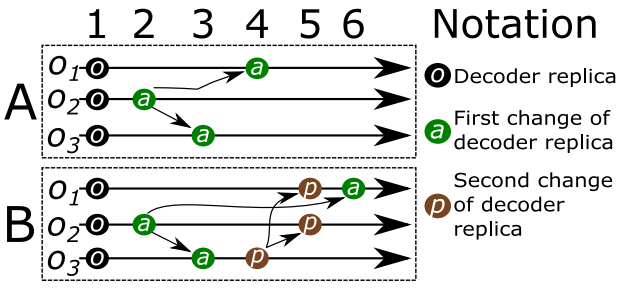


Figure 4.3: Two different scenarios of VNF-FG reconfiguration

In ideal conditions, consistency can be achieved by sending the notification to all affected replicas as depicted in scenario (A). But, non-deterministic network conditions, such as a random latency and packets loss can lead to an inconsistent reconfiguration. Scenario (B) in Figure. 4.3 highlight a possible inconsistent VNF-FG reconfiguration.

In this scenario, concurrent updates can be done. Such reconfigurations brings partial or total failures and are costly to fix. First, all orchestrators start with an initial value o (B-step1). Then, the second orchestrator o_2 updates its VNF-FG replica with value a and sends a notification to the other orchestrators (B-step2). This is followed by the third orchestrator receiving the notification and updating its replica (B-step3). After that, the third orchestrator o_3 updates again the replica, based on the previous update, to a new value p (e.g., updating a different input point) and sends a notification to all the others (B, step4). Two concurrent tasks are executed (B-step5). On one hand, the first orchestrator o_1 receives the second update from o_3 with value p and updates its VNF-FG replica. On the other hand, the second orchestrator o_2 also updates the replica with value p. Finally, the o_1 receives the second update from o_2 with value a (B-step6). Since the orchestrators have local knowledge of their administrative domain only, it updates its replica. In the end, VNF-FG replicas have different values which leads to inconsistent VNF-FG reconfiguration. The problem can be even more challenging when considering the non-functional dependencies. This is discussed in Section 4.2.3.

4.2.3 Consistent VNF Forwarding Graph reconfiguration with non-functional dependencies

In some cases, the orchestrators might reject update requests on the replicas related to their VNF-FG. This happen when the orchestrators consider that the execution of the changes will inevitably affect the non-functional requirements of their managed services (e.g., latency, data privacy) and then might lead to SLA violation. In fact, to stay consistent, all orchestrators need to consider the negation by non-functional dependencies. This problem is named as CVNF-FGR-NF. CVNF-FGR problem, discussed in Section 4.2.2, remains particular case of the CVNF-FGR-NF where orchestrators always accept the reconfiguration proposed by replicas coming from other orchestrators. The reader should note that the CVNF-FGR-NF problem is divided into two classes according to the wanted behavior in terms of consistency. If the reconfiguration to be applied is always validated by all orchestrators, then the problem does not support fault-tolerance (i.e., the case for critical applications). This is the preventive approach. If the reconfiguration operations can be undone, then the problem is considered ad supporting fault-tolerance. This this the corrective approach.

Figure. 4.4 resumes the scenario A introduced in Figure. 4.3 and extends it with the capability of the orchestrators can refuse a reconfiguration. In this scenario, four orchestrators (o_1, o_2, o_3, o_4) from different administrative domains manage two *shared* VNF encoders g^1, g^2 and two *shared* VNF decoders g^3, g^4. The first orchestrator o_1 manages the encoder g^1, the second orchestrator o_2 manages the encoder g^2, and so on. At the beginning, all *shared* encoders and decoders start with initial values represented by o, and $*$ as shown in the bottom of Figure. 4.4 (box A- Replicas). In addition, the encoders and decoders are related since decoder configuration depends of the encoder

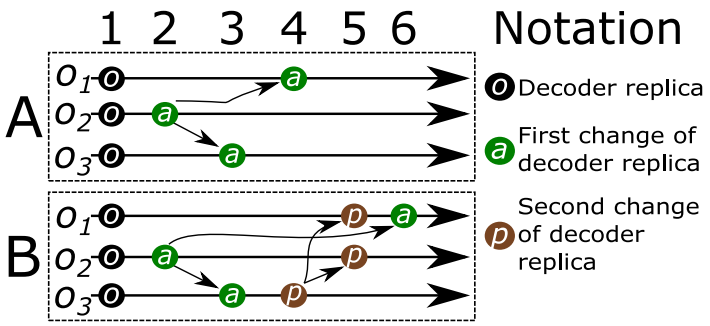


Figure 4.4: Example of an inconsistent VNF-FG reconfiguration scenario

configuration, as shown in the bottom of Figure. 4.4 (box B - Dependencies). This means, that in order to reconfigure a decoder, both orchestrators managing g^1, and g^2, respectively, have to accept the changes. The reconfiguration scenario is as follows: o_3 and o_4 change the value of their respective *shared* decoders g^3, g^4 with different values a, p, respectively (step1). Therefore, o_1 and o_2 update their *shared* encoders. o_1 verifies the proposed reconfiguration, accepts and proceeds with the change of the value of the encoder g^1 to a new value y (step2). As for o_2, it also accepts the changes and updates the encoder value to y (step2). After that happens three concurrent tasks (step3). Firstly, o_1 gets the notification from o_4 to reconfigure the VNF-FG. o_1 does not accept the reconfiguration, keeps the value y, and sends a negative reply to o_4. Secondly, o_2 gets the same notification from o_4. Since it accepts this most recent update, the orchestrator updates the value of the dependency to x; then, it replies positively to o_4. Thirdly, the concurrent task is the positive reply from o_1 to o_3, which updates the first of the two required answers (step3). Then, both o_3 and o_4 get a positive reply from one of their dependencies (step4). o_3 receives a positive reply from o_2, and updates the first of two required answers; while, o_4 from o_2, also updating the second answer. Since o_3 received both positive replies from its dependencies, it will notify o_4 to change the value of the VNF-FG replica. Two concurrent events happen as messages arrive to o_4 (step5). Firstly, the instruction to update the value of the VNF-FG replica arrives from o_3. Secondly, the negative reply from o_1 arrives. Thus, o_4 can choose between doing the reconfiguration or remaining in the initial state. In this scenario, a non-deterministic output creates an inconsistency. The first alternative is that o_4 updates the value of the VNF-FG replica to a; however, the dependencies have different values (box C - End state). The other alternative is to remain in the initial state; but the replicas have different values (box C End state). Moreover, because orchestrators can share the decoder among other orchestrators with limited knowledge, conflicts will arise as the replicas diverge further and further in every single concurrent reconfiguration.

4.3 The state-of-the-art in VNF-FG reconfiguration

There are several work in the relevant literature that studied the managementof the VNF-FG. The most common operations for VNF-FG management are embedding, reconfiguring and composing the forwarding graph [109], the major focus being on the embedding operation [110]. The embedding operation aims to support the selection of the virtual network instances and their connection links by the orchestrator [110]. The reconfiguration operation aims to determine the most appropriate action to update the VNF-FG (e.g. changing the order of the VNFs, extending the forwarding graph) while ensuring the properties of the service, such as availability [111].

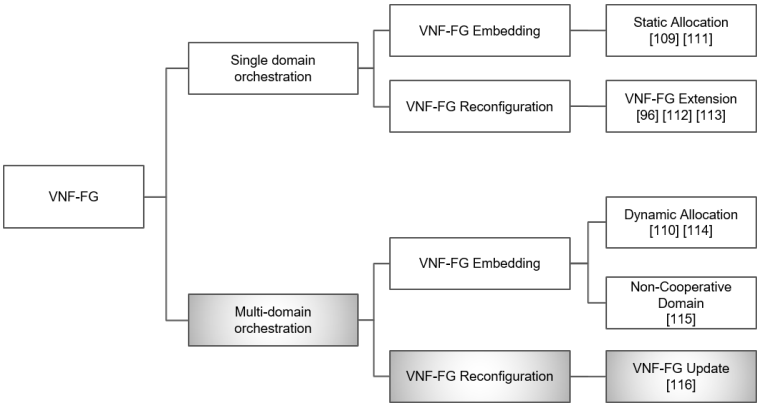


Figure 4.5: Classification of the related work on VNF-FG reconfiguration

The conducted literature review focus on the reconfiguration problem, since composing and embedding only consider static deployment of the VNF-FG. Moreover, if the orchestrator extends the VNF-FG, the reconfiguration can also include the embedding and the composing operations. Figure. 4.5 depicts the classification of the reviewed related work. The VNF-FG reconfiguration could be either in single network provider domain or over a multi-domain federation. $CONTRIB4$ implements the highlighted (lower) branch of the classification tree.

4.3.1 VNF-Forwarding Graph reconfiguration in single domain environment

The reviewed work for single domain environment supports either static VNF-FG reconfiguration or dynamic VNF-FG reconfiguration. The former considers a unique orchestrator where the managed VNF-FGs stay static. The primary goal of these work is to optimize metrics (e.g. latency [112], revenue [113], and energy [114]) while provisioning the VNF-FG. Since optimizing the placement is NP-hard, the authors often propose heuristics to solve the problem in larger instances while relying on restrictive assumptions such as: (i) federation is known in advance, static and fixed during

runtime and (ii) the single global orchestrator hav full knowledge of the underlying domain, such as topology or network policies. These assumptions are too simplistic and do not represent a true reflection of reality.

The dynamic VNF-FG reconfiguration in single domain environment tries to remedy the limitations of the static reconfiguration by allowing online changes in the VNF-FG embedding algorithms. For instance, the authors in [96] and [115] focus on the VNF placement. In these work, the orchestrator extends the VNF-FG by adding VNFs and links to respond to new demands. The optimal extended embedding problem is then solved through a decomposition algorithm in [96] and a Steiner Tree-based algorithm in [115]. Other work use bi-directional chaining [116]. This allows adding optional VNFs instances for deployed services [116]. While these works aim to reconfigure the VNF-FG, they do not consider the problem of inconsistency when multiple orchestrators concurrently try to update the VNF-FG. In case of a change in service usage, the global (central) orchestrator computes the new place to instantiate a VNF or it can update the VNF-FG by changing the VNFs' execution order. In this context, the consistency in the VNF-FG management here is not problematic, as the global orchestrator has all the required knowledge. Thus, the orchestrators synchronize the updates according to the global orchestrator. Despite this ease of consistency, these works do not scale well for multi-domain federations, as the fundamental assumption of complete knowledge is costly to implement. Additionally, the service providers prefer to keep their key information private and are generally refractory to sharing operating information with other service providers [94].

4.3.2 VNF-Forwarding Graph reconfiguration in multi-domain environments

VNF-FG management in multi-domain environments uses a a decentralized approach where multiple orchestrators jointly manage network services. There are only 2 work that contributed to VNF-FG management in multi-domain environments. The first one is part of the ETSI NFV standard [65]. It proposes a reconfiguration algorithm where the orchestrators coordinate with each other through grants to maintain the network service consistency. This work supports and manages concurrent reconfigurations. In the second work, the authors supports coordination enforced with causal consistency. This is achieved by making the orchestrators wait until they receive the appropriate reconfiguration instruction, while tracking the causal information using vector clocks [117]. This work only support non-concurrent reconfigurations.

In addition to these work, there are additional work that focused on the embedding problem. In these work, several techniques were used in the literature to avoid the repetetive calculation of the new VNFs placement following a reconfiguration operation. For instance, the authors in [118] propose a deep reinforcement learning technique to learn the dynamic behavior of the federation and predict the VNFs placement. Another work supports the migration of existing VNFs using an adaptive

centralized/decentralized orchestration algorithm to reallocate VNF-FG [111]. The last reviewed work considered a close and competitive environment where orchestrators hide their infrastructure from the other orchestrators [119].

4.3.3 synthesis

The conducted literature study highlights that there are only 2 work that contributed to VNF-FG management in multi-domain environments [65] [117]. In these work, when concurrent updates generate conflicts, the orchestrators must resolve conflicts/inconsistencies by solving consensus; thus, sacrificing performance over consistency. Solving consensus forces the orchestrators to have recourse to executing a coordination phase to prevent inconsistencies. Moreover, these work do not consider non-functional dependencies that might negate ongoing reconfigurations.

$CONTRIB4$ introduces the very first coordination-free orchestration algorithm to achieve consistent reconfiguration of the VNF-FG in a multi-domain federation. The proposed orchestration algorithm considers the consistency reconfiguration of a VNF-FG by updating either the connection points or classification rules. Futhermore, the algorithm supports the non-functional dependencies inherent in sharing network services in distributed multi-domains.

4.4 Coordination-free orchestration algorithm for multi-domain environments

As stated in Section 4.2.3, the goal in the CVNF-FGR-NF problem is to ensure that all the replicas of the VNF-FG, as well as, their dependencies have the same values. However, non-deterministic network conditions (e.g., out-of-order delivery of messages, latency fluctuation, loss of packets) from one side and concurrent updates, requested by other orchestrators, from the other side can lead to inconsistencies while reconfiguring the VNF-FG. Adding to that, prospective dependencies might negate part of the requested updates, amplifying the negative effects of the previous conditions.

A dependency relation δ is described as binary when it cannot have a dependency with itself. This means that δ takes as input two different VNF-FG ($g \in G, g' \in G'$). As the VNF-FGs have replicas, they must have the same values and state all the time. Thus, if the relation $\delta(g, g')$ holds, this implies that every element in G has a dependency with every other element in G'. This could be formalized as follows: $\exists\delta(g, g'), g \in G, g' \in G' \implies \delta(G, G')$. If δ is a bi-directional relationship, then it also implies that all elements in G' have a dependency relation (i.e. $\delta(G', G)$). This means whenever a reconfiguration takes place, the orchestrator that does the update has to notify the other orchestrators that manage a VNF-FG in the super-set $G \cup G'$.

When reconfiguring VNF-FG with a dependency relation, orchestrators can have different values for the VNF-FG. To prevent inconsistencies, a conflict resolution mechanism is required. Traditionally, in distributed systems, conflict resolution is done by

consensus among all orchestrators to achieve sequential consistency (see Section 4.1). The problem is that managing consensuses will generate an overhead that hinders the applicability of such solutions, especially considering the recurrent low latency requirement in the ETSI NFV specification. An intuitive solution to this drawback would be relaxing the consistency model as it is the case with the Strong Eventual Consistency model (see Section 4.1). Such model achieves the optimal trade-off between consistency, availability, and partitioning.

Consistent-free Replicated Data Types (CRDTs), introduced in Section 4.1, can offer an even better consistency model. In fact, CRDTs implement an automatic conflict resolution mechanism. The idea is to design an orchestration procedure using data structures that support strong eventual consistency to avoid the coordination phase. For the proposed approach, whenever an orchestrator executes a reconfiguration, it will change either the matching attribute and/or the connection points, as specified by the ETSI NFV standard information model [95]. This means changing the data structure that holds all the required information for them. For example, to change the matching attribute, the orchestrator will set a new value for the protocol recognized, the source and destination addresses, and ports.

The proposed approach proposes 2 variants, i.e., preventive variant and corrective variant. The preventive variant aims to prevent any inconsistencies by applying updates only when all replicas and dependencies have asserted an updated proposal. This means that an orchestrator managing a given VNF-FG $g \in G$ with a dependency $g' \in G'$ sends a reply to all other orchestrators in the set $G \cup G'$. Whenever an orchestrator managing a dependency of the VNF-FG g receives the proposal, it will reply either positively or negatively. The change is delayed and will not be applied until all answers are received. If any is negative, the reply will be discarded. The corrective variant is more permissive, as it allows updates to take place at the moment the notification arrives. Similar to the first variant, the orchestrator managing the VNF-FG $g \in G$ must send to all orchestrators managing a VNF-FG in the set $G \cup G'$. However, the receiving orchestrator will only notify a negative answer to the others. Indeed, when an orchestrator receives a negative answer, it must make the required changes to be in the most promising consistent state.

4.4.1 Preventive variant

The preventive variant of the proposed approach is designated as *CF-P*. It consists of a set of designed algorithms. When an orchestrator tries to update a given VNF-FG, it first executes Algorithm 4. This algorithm was introduced in 5 and 6.

First, the orchestrator applies the reconfiguration to a copy of the VNF-FG g (Algorithm 4, line 3). Then, it increases the counter for the reconfiguration to assign a unique identifier to it (line 4). After that, it computes the set of orchestrators managing replicas of theAlgorithm 4 (line 5). Next, the orchestrator creates a list to store the replicas' replies (lines 6, 7). Finally, it appends the reconfiguration to a list

Update the VNF-FG algorithm - the preventive variant

```
1  VNFFGUpdate()
3      update_vnfforwarding_graph_copy()
4      create_unique_identifier_for_reconfiguration()
5      get_list_of_orchestrators_to_send_message()
6      create_empty_list_to_store_answers()
7      initialize_list_with_positive_value()
8      add_new_list_to_pending_operations()
9      create_message_to_notify_update()
10     send_message_to_all_affected_orchestrators()
```

of pending operations and sends the instruction to reconfigure to other orchestrators (lines 8-10).

When an orchestrator receives a notification, it will execute Algorithm 5. First, the orchestrator will compute the set of orchestrators that have replicas of the VNF-FG (line 3). Then, it creates a temporary answer that will change depending if the change will be accepted or not (line 5). If the orchestrator has not previously received a reply to this reconfiguration, it will create a new empty list and add it to its pending operations (line 7). After, the orchestrator checks the feasibility of the reconfiguration (i.e., the replica and its dependencies satisfy non-functional properties); if valid, the orchestrator will mark the reconfiguration as accepted (line 13); otherwise, as false (lines 23, 25). If the orchestrator accepted the reconfiguration and the counter of the new operation is greater than the current one, the orchestrator applies the changes and it updates the counter of the VNF-FG (lines 17, 19).

Receive a notification to update the VNF-FG algorithm - the preventive variant

```
1  VNFFGReceive(Δ)
3      get_list_of_orchestrators_to_send_reply()
5      create_answer_and_set_as_false()
7      if_first_create_list_and_append_to_pending_operations()
9      if check_feasibility(Δ) then
11         mark_entry_as_positive()
13         update_answer_to_positive_reply()
15         if all_entries_are_positive_and_greater_counter then
17             reconfigure_the_vnffg()
19             update_the_counter_with_new_one()
21     else
23         mark_entry_as_negative_in_list()
25         add_entry_as_negative_in_list()
```

When an orchestrator receives a reply, it will execute Algorithm 6. First, the orchestrator stores the reply in the list for the VNF-FG reconfiguration (line 3). If all entries are positive, the orchestrator will apply the update (lines 7, 9).

Receive a reply message algorithm - the preventive variant

```
1 VNFFGReceive()
3     mark_entry_with_answer()
5     if all_entries_of_list_are_positive_and_counter_is_greater then
7         reconfigure_the_vnffg()
9         update_the_counter_with_new_one()
```

Consider the same reconfiguration of a shared VNF-FG as in Section 4.2.3 where four orchestrators o_1, o_2, o_3, o_4 manage two VNF encoders g^1, g^2 and two VNF decoders g^3, g^4, respectively. The first orchestrator o_1 manages g^1, the second orchestrator o_1 manages g^2, and so on. Figure. 4.6 shows the execution for the reconfiguration to ensure all replicas are consistent by having the same values. For ease of readability, the figure only shows a single value that changes whenever a reconfiguration happens. However, in reality the proposed approach considers the whole data structure of the VNF-FG to be compliant with to the ETSI information model.

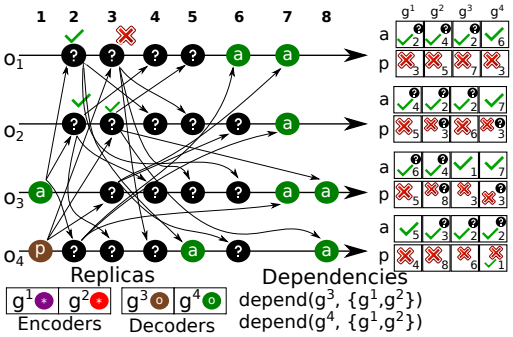


Figure 4.6: Example of the coordination-free VNF-FG reconfiguration - preventive variant

o_3 and o_4 try to concurrently update the VNF-FG (Figure. 4.6, step1). o_3 proposes a new value a for its VNF-FG; while, the o_4 updates its VNF-FG to p. Both store the values in their respective lists and send the proposal to all affected orchestrators (in this example all the other orchestrators). Then, three concurrent tasks need to be executed (step2). Firstly, o_1 receives the proposal from o_3. After validating this proposal, it stores it in a list of pending reconfigurations as shown in the right side of the figure, where a question symbol is stored in the entries for g^1 and g^3, respectively. After that, o_1 sends notification to all affected orchestrators. Secondly, similarly to o_1, the o_2 accepts, stores, and sends notifications for value a. Thirdly, o_4 receives the proposal from o_3 and also does the three operations as the others. In Step3, four

concurrent operations are being executed. The following description only covers the first two as the very same Three operations apply to the others aswell. For the first task, o_1 receives the proposal from o_4. The orchestrator verifies if the reconfiguration is valid; however, it decides not to accept it. o_1 then adds the proposal as negative, as shown in the right side of the Figure. All subsequent notifications of proposal for value p will be automatically negated. For the second task, o_2 validates the proposal for value p. Steps 4 and 5 shows the notifications arriving to the orchestrators. As for the notification from o_1 to o_4, since all values are already validated by replicas, the o_4 finally can reconfigure its VNF-FG replica. This is highlighted in Figure. 4.6 by the change of color and value of the g^4 VNF decoder. Eventually, all notification and proposals arrive with Steps 6-8. At the end of the reconfiguration, all VNF-FG replicas have the same values.

4.4.2 Corrective variant

The corrective variant of the proposed coordination-free approach is named *CF-C*. It reconfigures a VNF-FG when a notification arrives with no delays. Moreover, it does not send a notification to other orchestrators. Only when a dependency does not accept a reconfiguration due to violating non-functional requirements, the orchestrator will send a negative notification to the others. Specifically, whenever an orchestrator receives a negative notification, it will reconfigure its VNF-FG to a provisional state after merging with the notification. Algorithm 7, Algorithm 8, and Algorithm 9 implement The procedures of the *CF-C* variant.

When the orchestrator reconfigures a VNF-FG, it starts by executing Algorithm 7.

Update the VNF-FG algorithm - the corrective variant

```
1  VNFFGUpdate()
3      apply_reconfiguration_to_vnffg()
5      increase_the_vnffg_counter()
7      get_list_of_orchestrators_to_send_notification()
9      add_reconfiguration_entry_to_heap()
11     create_update_notification_message()
12      send_notification_to_all_affected_orchestrators()
```

First, the orchestrator updates the VNF-FG (line 3). Then, it increases the counter to assign a unique identifier to the operation (line 5). After that, the orchestrator computes the list of orchestrators that have replicas of the VNF-FG (line 7). Next, it adds the reconfiguration to the heap of accepted reconfigurations (line 9). Finally, the orchestrator creates a message and sends it to the list of orchestrators (lines 11, 12). When an orchestrator receives this message, it executes Algorithm 8.

For Algorithm 8, the orchestrator first checks if the reconfiguration is already stored in the list of negated reconfigurations; if not, it continues (line 3). After, the

Receive a notification to update the VNF-FG algorithm - the corrective variant

```
1  VNFFGReceive(Δ)
3      pass_if_entry_is_negated()
5      if check_feasibility(Δ) then
7          if counter_greater_equal_greater_identifier then
9              apply_the_reconfiguration_to_the_vnffg()
11             increase_the_counter_for_the_vnffg()
13         add_entry_to_heap()
15     else
17         get_list_of_affected_orchestrators()
19         add_reconfiguration_to_list_of_rejected_operations()
21         create_negative_reply_message()
22          send_negative_reply_to_affected_orchestrators()
```

orchestrator checks if the reconfiguration is feasible (i.e., replicas and dependencies' non-functional properties will be satisfied after reconfiguration). If it is valid, the orchestrator checks if both counters are greater than the current ones before proceeding with the VNF-FG reconfiguration. The top counter is then updated (lines 9, 11). Otherwise, if the condition on the counters is not satisfied, it is added to the heap in the correct place by using the counters as identifiers (line 13). If the reconfiguration is not accepted, the orchestrator computes the list of affected orchestrators, adds the reconfiguration to the list of rejected operations, creates a negative reply message, and sends it to all affected orchestrators (lines 17-22). When an orchestrator receives the negative reply, it executes Algorithm 9.

Receive a reply message algorithm - the corrective variant

```
1  VNFFGReceive()
3      remove_operation_from_heap()
5      add_reconfiguration_to_list_of_rejected_operations()
7      apply_new_update_from_consistent_state()
```

For Algorithm 9, the orchestrator removes the operation from the heap (this could be the top or any other position), adds the reconfiguration to the list of negated operations, re-orders the heap, and applies the reconfigurations starting from the initial consistent state (lines 3-7).

Consider the same reconfiguration of a shared VNF-FG as in Section 4.2.2 where four orchestrators o_1, o_2, o_3, o_4 manage two *shared* VNF encoders g^1, g^2 and two *shared* VNF decoders g^3, g^4, respectively. The first orchestrator o_1 manages g^1, the second orchestrator o_1 manages g^2, and so on. Figure. 4.7 shows the execution for the re-

configuration to ensure all VNF-FG replicas are consistent by having the same values. For ease of readability, the figure only show a single value that changes whenever a reconfiguration happens. However, in reality, the proposed approach considers the whole data structure of the VNF-FG as sipulated by the ETSI NFV information model.

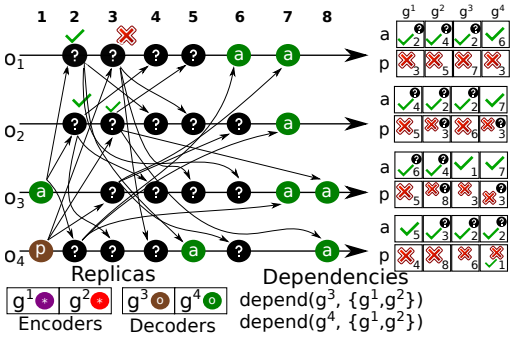


Figure 4.7: Example of the coordination-free VNF-FG reconfiguration - corrective variant

In Step1, o_3, o_4 concurrently update the VNF-FG. o_3 updates its VNF-FG with a new value a; while o_4 with p, respectively. Both add the value to their heap and then send the proposal to the affected orchestrators. In Step2, two concurrent tasks are being executed. Firstly, o_1 receives the proposal from o_3. After validating this proposal, it applies the reconfiguration to VNF-FG g^1 and adds the state to the heap. This is shown on the right side of the figure. Secondly, similarly to o_1, o_2 accepts, reconfigures, and saves the state. In Step 3 three concurrent tasks are being executed. Firstly, the proposal from o_4 arrives to o_1. The orchestrator verifies if the reconfiguration is valid; however, it decides not to accept it. o_1 adds it to the list of negative proposals and notifies all the affected orchestrators (in this example, all the others). Secondly, the proposal from o_4 arrives to o_2; unlike o_1, o_2 accepts the proposal and reconfigures the VNF-FG g^2 to match the state of value p. This is shown in Figure. 4.7 where the top now is p; unlike in the previous Step2. Thirdly, the proposal from o_3 arrives at o_4 that accepts it. However, because its reconfiguration takes precedence, it will not apply the reconfiguration. In Steps 4 and 5, the rest of the notifications arrive. Whenever an orchestrator receives a negative reply, it reconfigures again to another state. The value in the heap is removed and added to the list of negated proposals. This is shown in steps4 and 5 in Figure. 4.7. At the end of reconfiguration, all VNF-FG replicas have the same values.

4.5 Implementation and evaluation

The proposed coordination-free orchestration approach was implmented to validate and evaluate its findings. The running prototype supports the 2 designed variants (i.e., preventive variant and the corrective vraiant). The evaluation experiments were

performed in a public distributed cloud infrastructure over a multi-domain federation. The aim of the performed experiments is to evaluate the proposed approach versus the ETSI standard orchestration algorithm for consistent VNF-FG reconfiguration based a set of well-defined metrics.

4.5.1 Proof of Concept

The developed prototype implements the preventive variant (*CF-P*), as well as, the corrective variant (*CF-C*) of the proposed approach in Python. For evaluation purposes, the ETSI standard procedure (*NCF-E*) was implemented too. As a reminder, the associated algorithm use eventual consistency. It enables VNF-FG reconfiguration by applying updates the moment they arrive.

As for the deployment, the prototype was provisioned in Microsoft Azure[1], a public cloud provider infrastructure. To set up a representative multi-domain federations, several and remote domains of the service provider were selected from the following locations: North Europe, West US, South Korea, East US, and the UK. The network federation topology is generated randomly. Connections are achieved either by internal or external dependencies. The network domains are connected to each other over the Internet, such that average latency is representative of real conditions. Every single domain has its own orchestrator and policies. Identical virtual machines with the same configuration (i.e., 2 CPU cores, 30GB of hard drive, 4GB of RAM, and Ubuntu 18.04-LTS) were created in each domain to host and execute the local orchestrators. The network policies and topology of each domain are described with JSON-based descriptor. The descriptors are randomly generated and lists information such as location of the domain, types and number of running VNFs, their related VNF-FG and any other required information for VNF-based network services (e.g., communication protocols, data storage). The VNFs were implemented and hosted within Docker[2] containers. In addition, yet another descriptor was designed to implement and describe the workload and the required concurren changes at runtime. The source code of the prototype is available at: `https://zenodo.org/record/5336614`.

In addition to the prototype, for evaluation purposes, a multi-domain orchestrator implementing the ETSI standard orchestration algorithm for consistent VNF-FG reconfiguration was developed in Python. The several existing open-source ETSI NFV orchestrators (e.g., OSM[3], Open Baton[4]) do no implement the required interfaces to support a federation.

[1] `azure.microsoft.com`
[2] `docker.com`
[3] `osm.etsi.org`
[4] `openbaton.github.io`

Table 4.2: The parameters for the VNF-FG reconfiguration prototype

Parameter	Range
Maximum Delay (Max_D)	[1, 10, . . . , 100000] ms
Number of Reconfigurations (Nm_R)	0, 150, 300, . . . , 1800
Type of Reconfigurations (T_R)	Concurrent
Probability to Negate (Pb_N)	0 - 5 - 10 - 20 - 40 - 80 %

4.5.2 Evaluation scenarios and considered metrics

The prototype is measured under different experimental scenarios and considering several parameters for each experiment. Table 4.2 lists the defined parameters for the experiments. Each configuration parameter creates different test scenarios from the ideal to the worst case. For instance, an ideal scenario would have zero delay (i.e., **Max_D**=0), accept all reconfigurations (i.e., **Pb_N**=0), and non-concurrent reconfigurations (i.e., **Nm_R**=0). A worse scenario, compared to the ideal, would have the greatest delay (i.e., **Max_D**=100ms), negate all reconfigurations (i.e., **Pb_N**=20), and have concurrent reconfigurations (i.e., **Nm_R**=1200).

The list of the considered metrics for the evaluation is as follows:

- **Total reconfiguration time** is the time in milliseconds necessary for for each algorithm to process a VNF-FG reconfiguration . A lower value is preferred.
- **Number of reconfigurations** is the number of times a particular VNF-FG is reconfigured. Some algorithms allow a certain period of inconsistency, thus this metric measures the extra cost associated with more flexibility in terms of quick reconfigurations. A lower value is preferred.
- **Latency per operation** is the time difference in milliseconds between a VNF-FG reconfiguration request and its concrete reconfiguration. A lower value is preferred.
- **Inconsistencies** refers to the number of updates that are different for each replica. A lower value is preferred since inconsistencies directly translate to cost for providers and lower performance for users.
- **Overhead per messages** refers to the amount of data each orchestrator sends to other replicas and dependencies when executing an algorithm. A lower value is preferred. This metric is associated with a cost for CRDT-based algorithms.

All combinations of the configuration parameters, listed in Table 4.2, were used for the experiments. Each experiment is performed thirty times and the average values is calculated for each metric previously described. The running prototypes of *CF-C*, *CF-C* and *ETSI-C* were evaluated considering the same scenarios, test collections and metrics. The experiments mainly focus on the concurrent scenarios.

Two variables were varied and monitored during the experiments, i.e., probability of negation and network delay. The probability of negation is selected in four scenarios (i.e., 0%, 5-10%, 20-40%, and 80%). A negation of 0% means that the replicas and the dependencies always accept changes. A negation of 5-10% means that replicas almost always accept changes. In fact, higher values of negation incu more complexity for the algorithms considered, since they have to ensure no inconsistencies happen. For low probability negations, the service providers have sufficient resources to apply changes and will accept the reconfigurations. Higher probability negations mean there is a lack of resources for providers to reconfigure.

As for the network delay, the values were varied to evaluate different scenarios. This Section mainly discusses the scenarios with delays of 100ms and 1000ms. The data test collections, the comprehensive experiment scenarios and their results are available at:`https://zenodo.org/record/5336614`.

4.5.3 Obtained results

The obtained measurements are depicted in the figures of this Section. The proposed preventive variant (*CF-P*) is represented with a green axis/diamond for delays for 100 and 1000ms, respectively. The proposed corrective variant (*CF-C*) is represented with a blue right/left triangle for 100 and 1000ms, respectively. Finally, the ETSI algorithm (*NCF-E*) is represented with an orange up/down triangle for 100 and 1000ms, respectively. The reader should note that, for all the obtained results, a confidence interval was drawn at 95% using the Seaborn Python library[5] that implements a bootstrapping algorithm.

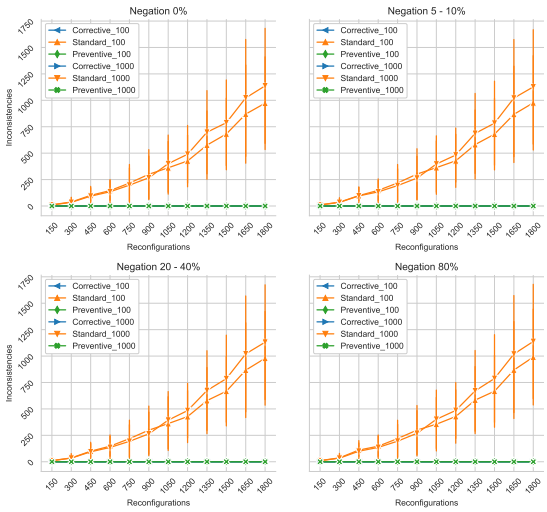


Figure 4.8: Number of inconsistencies per number of performed VNF-FG reconfigurations

[5] `seaborn.pydata.org`

Figure. 4.8 depicts the evolution of the number of inconsistencies per number of performed VNF-FG reconfigurations (lower is better). These inconsistencies happen because the replicas or dependencies have different values. The ETSI standard algorithm does not prevent inconsistencies. It has the worst performance, as the number of inconsistencies increases with more concurrent reconfigurations. Both variants prevent inconsistencies for any type of scenario; thus, negation and delay factors do not have an impact on the variants. The obtained results for both variants are equal independently of the probability of negation. This is explained by the fact that both variants enforce strong eventual consistency, and replicas converge to a consistent state irrespective of the number of negations and network delay. The results remained the same for the three algorithms when the probability of negation were varied.

Figure 4.9: The latency variation per VNF-FG reconfiguration operations

Figure. 4.9 measures the latency per VNF-FG reconfiguration operations (lower is better). This represents the time needed to wait before reconfiguring a VNF-FG. Obviously, the standard behaves the best as it applies the reconfiguration when it receives the request. The preventive variant is the worst since it must wait. With 1800 concurrent reconfigurations and 0% negation probability, the average latency per operation is about 1 minute for the preventive algorithm. However, when the negation probability increases (e.g. 20-40%), the latency reduces. This behavior is consistent with the way the preventive algorithm performs. If one entry in the table is false (because either a dependency or replica did not accept the changes), the preventive algorithm can abort the reconfiguration without waiting for the rest of the answers. Thus, when orchestrators negate more updates, the latency for the preventive is reduced. Moreover, one minute of latency per operation might seem a high number compared to the corrective and standard but this latency is still lower compared to the latency per transaction of consensus solutions (e.g. 10 minutes per transaction [120]). The depicted latency

is representative for all the parameters' combinations.

Figure. 4.10 shows the amount of the necesary messages that the orchestrators need to send to resolve conflicts dunring VNF-FG reconfigurations (lower is better). The preventive variant gets the worst performance of all. The corrective algorithm sits in the middle of the preventive and standard algorithms. However, with a high number of negations (i.e. $\geqq 20\%$), the corrective algorithm behaves like the preventive, as seen in how the corrective trend line moves towards the preventive one. In the worst-case scenario (i.e., 100% negation probability), the corrective variant will send the same number of messages. Delay has a slight impact on the number of messages sent. It widens the interval but the behavior remains the same.

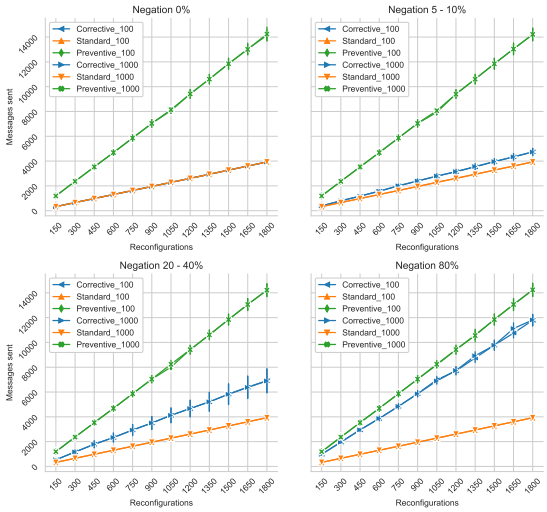


Figure 4.10: Number of generated messages to resolve conflicts

Figure. 4.11 shows the overhead (the extra) per message that need to be exchanged between the orchestrators to achieve consistent VNF-FG reconfigurations (lower is better). Similar to the number of messages sent, the preventive variant gets the worst performance of the other variants. However, the negation probability seems to have a lesser impact on the corrective algorithm. For instance, with negations between 40% and 80%, it only doubles the amount of data sent.

Figure. 4.12 shows the number of extra VNF-FG reconfigurations per algorithm (lower is better). The preventive variant is obviously the better since it is getting zero extra reconfigurations, while the corrective variant behaves worst. The corrective variant is sensitive to the negation probability. For instance, if all reconfigurations are accepted, the corrective variant does not have extra reconfigurations. However, with a greater probability to negate, the corrective algorithm must reconfigure more services, as shown in the number of reconfigurations done with 5-10% (i.e., reconfigurations 1000) and 20-40% (i.e., 3000 reconfigurations).

Figure. 4.13 shows the average reconfiguration time for the VNF-FG. Negation prob-

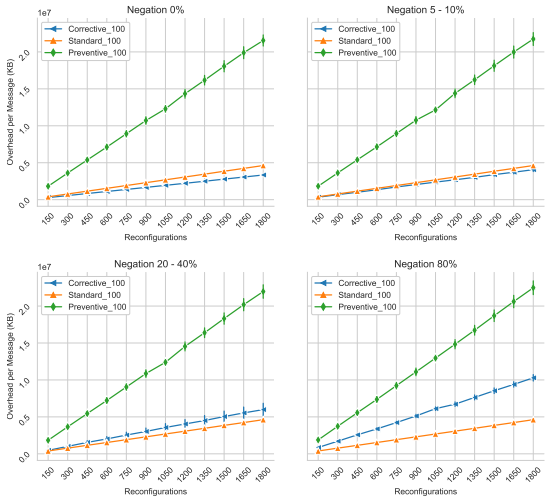


Figure 4.11: Overhead per messages during a VNF-FG reconfigurations

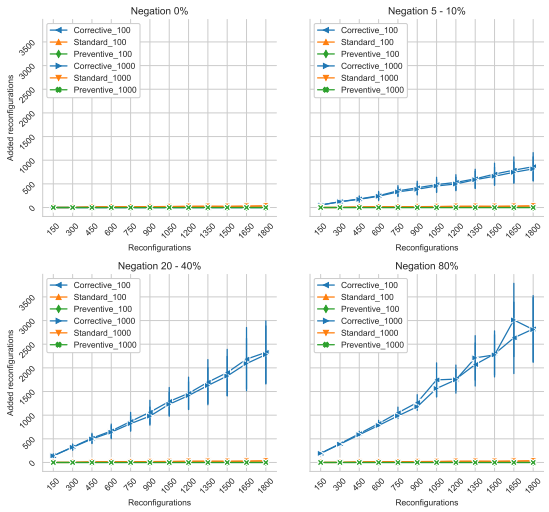


Figure 4.12: The number of extra VNF-FG reconfigurations.

ability has greater impact than delay. With no negation, the corrective variant achieves the fastest reconfigurations; while the preventive variant takes more time. This is because the preventive variant sends more messages than the corrective in this case. With more negations, the preventive variant has achieved a faster reconfiguration time; while the standard algorithm takes more time. This is because, as previously mentioned, it only takes a single negated entry into the list of changes for the preventive variant to abort the reconfiguration. With higher negation probabilities (80%), the standard has the slowest reconfiguration.

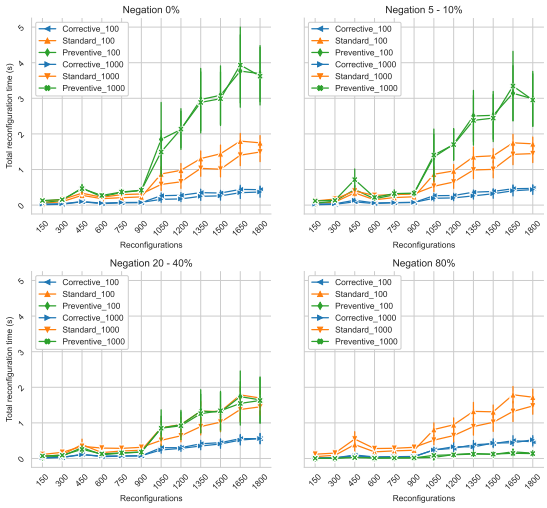


Figure 4.13: Reconfiguration time for the VNF Forwarding Graph.

4.5.4 Observations

The network delay does not seem to be of a great impact compared to the negation probability. From the obtained measurement, this could be seen when two charts of the evaluated algorithms overlap which happens for most of the experiments. However, in some scenarios, the network delay has some impact, such as in the evolution of the number of inconsistencies for the standard algorithm. Moreover, considering more concurrent reconfigurations, the variance increases for all metrics. Yet, for some experiments, such as the evaluation of the number of inconsistencies and the added reconfigurations for the corrective variant, it has a greater impact. These results enable refining the optimal configurations the applications can rely on considered each one of the implemented algorithms. For resources-constrained federations (e.g., IoT virtualized networks [121]), where consistency is a priority, extra reconfigurations are expensive, memory is limited but latency is not a constraint. In such environments, service providers should select the preventive variant over the corrective. On the other hand, the service providers would use the corrective variant in more specialized environments where network speed and response time are required, memory is available, reconfiguration (in terms of resource consumption) is cheap, sending messages is costly, and the probability to negate reconfigurations is low. NFV satellite networks are among the examples for these applications [122]. Furthermore, the corrective variant does not require knowing in advance the number of orchestrators in the federation, thus, it is possible to use it in open federations where new orchestrators can temporally join and leave.

Chapter 5

Open Issues and Research Directions

Contents

Over the years, computing paradigms have evolved from distributed, parallel, and grid to cloud computing. Although cloud computing has been in the limelight for decades now, the COVID-19 pandemic has given significant and impressive boost to its rolling-out in all application domains (e.g., telco, teaching, visio conference, multimedia). In fact, the cloud evolved from "just about" on-demand computing and storage capabilities to a comprehensive ecosystem enabling novel opportunities and providing reliable solutions for meeting the ever-changing needs of the business. Furthermore, the cloud is no longer centralized with a siloed architecture. Instead, it evolved to a complex continuum of capabilities and features.

Many organizations are reimagining their business models by migrating to the cloud continuum. The ultimate goal for them remains the optimization of the CAPEX and OPEX but this obviously entails evolving their applications and services. The research work presented in Chapter 2, Chapter 3 and Chapter 4 discuss a set of methodologies and models that assist and help these organizations to support the challenging actions and phases that make up the applications/services lifecycle over the fully distributed, dynamic and mobile cloud continuum resources.

To further square the circle, there is still need to investigate, design and sketch additional procedures and models that would automate the support of the whole lifecycle process introduced in Figure 1.2. For instance, similarly to the DevOps/continuous integration methodologies in the software engineering, there is a need of appropriate and novel procedures that would enable the automatic, or even the autonomic - when this can be possible, switch from one lifecycle step to another. In a further analogy,

the MLOps methodologies, applied in machine learning engineering, can be source of inspriration in this context and considered as a starting point to lay out the premises for a automatic/autonomic analytics models in the cloud continuum. Both research directions combined would require to step away from the classical QoS management patterns and emphasize the need for proactive QoS management techniques instead of the classical reactive QoS management that is commoly used nowadays in the cloud continuum. Section 5.1 introduces a common use case for cloud continuum (i.e., autonomous cars) and motivates the needs for introducing and supporting proactive QoS management. The end goal of this research statement is to achieve the so-called "Tactile Internet" within a few years.

5.1 Motivating use case: towards the next-generation autonomous cars

Autonomous cars are self-driving vehicles that could ride without the intervention of a humans. It gather and process data in the cars' neighborhood and autonomously plan and execute the right control actions to make in efficient and safe way (e.g., brake the car, change driving lane, avoid obstacle).

The next-generation of autonomous cars aim at (i) outsourcing part of the car computation over external resources and (ii) enabling the communication between the car and the remote services [11] [12]. For instance, during a ride from location **A** to location **B**, the car could intercat with other remote entities and services in the smart city to wind an optimal itenerary (e.g. the fastest, the closest). Consequently, the autonoumous services are location-aware and latency-sensitive. These services would help the car to be aware about the prospective events that might happen along the way and, when necessary, dynamically anticipate alternate iteneraries. Two kind of events are considered: planned events such as scheduled construction (Figure 5.1, event1) and unplanned events such as traffic congestion or accidents (Figure 5.1, event2) that might happen during the trip. While it is relatively straightforward to take into consideration the planned events in the itenerary calculation, updating the itenerary, on-the-fly, considering the unplanned events is challenging for the autonomous cars. In this scenario, the neighboring entities (e.g., involved cars in the accident, traffic outdoor cameras, and witnesses with their smartphones) could forward the information to the cars heading to the accident zone.

Obviously, existing navigation applications such as Google Maps[1] or Waze[2] are able to provide human drivers with these information in "real time" and dynamically adapt/adjust the navigation itenerary based on the application's configuration and/or the end-users preferences. Figure 5.1 depicts the fully-cloud architecture associated to this scenario. The information are first aggregated (communication delay 1) and pro-

[1] `maps.google.com`
[2] `waze.com`

cessed (processing delay) in remote Web servers before being transmitted to end users with some time shifting (communication delay 2). This inevitably causes considerable delays that could be tolerated for regular cars, but not for the autonomous ones.

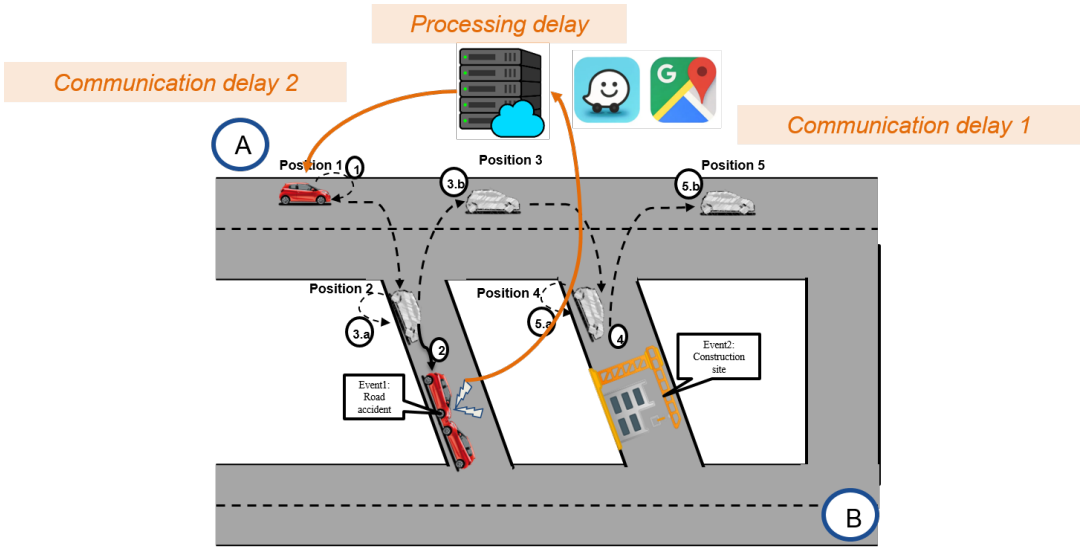


Figure 5.1: The communication flow for autonomous cars - fully-cloud architecture

To address these limitations, an alternative for autonomous cars is to get advantage from the cloud continuum and outsource part of their computation and data analytics to external but very close resources in the neighborhood as it is depicted in Figure 5.2. These resources could be implemented as fog/edge nodes and then host the services that will be able to gather (communication delay 1), parse (processing delay) and forward to the car any relevant information for its navigation (communication delay 2). The communication delays between the car and these remote services are then extremely short and do not affect the proper functioning of the car. This is quite challenging keeping in mind that, as discussed in Section 1.3, the topology of the fog/edge networks is highly dynamic. As a reminder, fog/edge nodes might arbitrarily (dis)appear and/or move and change location during runtime. This will inevitably increase the probability of QoS degradation at runtime. For instance, the car could communicate with a remote service to get additional relevant data for its navigation. The service is hosted and executed over a close edge device. A sudden workload increase (in the edge node), a system shutdown (of the edge node), or a car movement are among the reasons that would trigger a significant increase of the latency between the car and the edge node which might cause the car not to operate in a proper and safe way. The associated QoS management techniques must be efficient enough to detect and fix any prospective QoS degradation and continuously able to maintain the network latency within the required range for the autonoumous car proper functioning. To that end, it is obvious that a reactive QoS management will not be able to meet such requirement and that

proactive QoS management with the suitable techniques would allow to detect and fix the prospective QoS degradation that might happen at runtime is more appropriate.

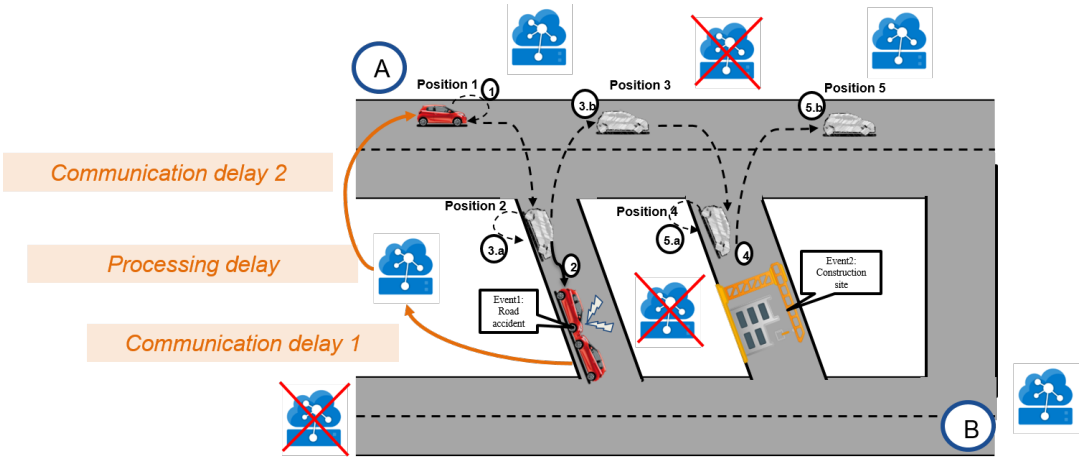


Figure 5.2: The communication flow for autonomous cars - hybrid cloud-edge architecture

5.2 Proactive QoS management

The conducted literature review highlighted a close relationship between QoS management in dynamic and mobile environments from one side and event-based service placement problems in these environments from the other side. Indeed, managing QoS metrics (e.g., latency, bandwidth, jitter, response time, packets loss) in dynamic and mobile environments often meant (re)placing and/or migrating the services over the available resources in such way to optimize the QoS metrics with respects to the applications/services requirements. Figure 5.3 shows the classification of the relevant literature [123] [124] [125] [126] [127] [128] [129] [130] [131] [132] [133] [134] [135] [136] [137] [138] [139]. The performed classification takes into consideration the work that only deal with network dynamicity (with no support of the edge nodes/end-users mobility), as well as, the work that deal with both characteristics. For each one of them, the classification tree involves the reactive and the proactive QoS management approaches.

To set the stage for proactive QoS management, a prelimenary research work is already done. It relies on the case study introduced in Section 5.1 and aims to support a statistical approach for QoS metrics prediction (see the highlighted branch in the classification tree). The considered service metric is the network latency. The latter refers to the communication delay from the vehicle to the computing edge node, the processing time of the service, and the communication delay from the computing node back to the vehicle. It is calculated through the following equation:

$$latency_{cn_i} = cd_{v,cn_i} + texec_{cn_i} + cd_{cn_i,v} \tag{5.1}$$

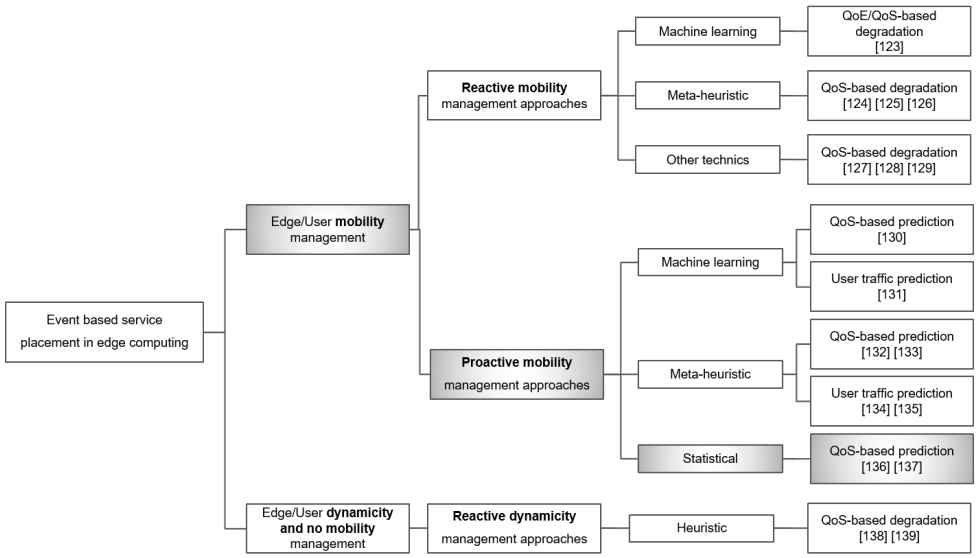


Figure 5.3: Classifiation of the literature for service placement

Where:

- $latency_{cn_i}$ is the latency of the i^{th} computing node (cn_i).
- cd_{v,cn_i} is the communication delay from the vehicle v to the cn_i.
- $texec_{cn_i}$ is the processing time of the decision module by cn_i.
- $cd_{cn_i,v}$ is the communication delay from cn_i back to v.

The selection of this approach was motivated by the target deployment environment (i.e., mobile and resources-limited edge nodes). Indeed, statistical models are less demanding than other machine learning models such as the neural networks. For the considered use case, predicting the latency degradation implies identifying the optimal next edge node to host and execute the remote service and proceeding with its re placement (i.e., stateful migration) to that node, in advance, before even than the QoS degradation occurs. This process should be completely independent and transparent to the moving autonomous car. Figure 5.4 depicts the high-level architecture introduced in [12] and enabling the next-generation autonomous car to ride and intercat with the smart city edge devices. For autonomous navigation and cruising, the car relies on several services, part of the *Autonomous car domain*:

- The *Perception service* relies on the car's equipment (e.g. sensors, cameras) to gather the navigation data like video stream and distances from other objects on the road.

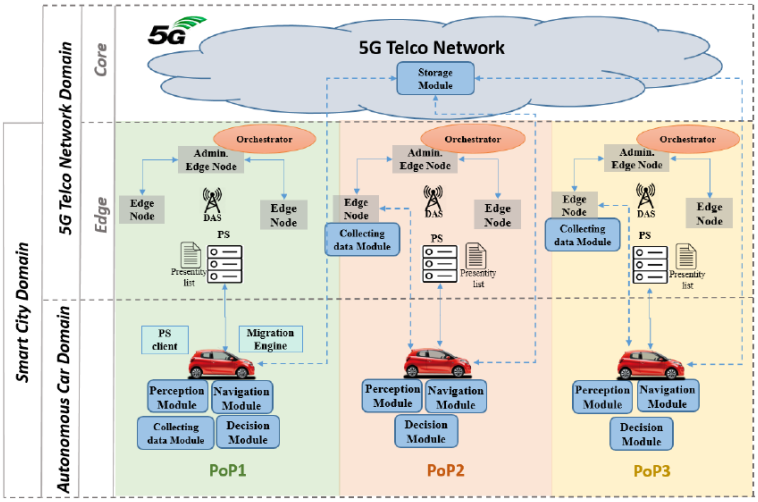


Figure 5.4: High-level architecture of the next-generation autonomous car case study

- The *Collecting-data service* is in charge of collaborating with the other devices in the car's neighborhood (e.g. crossing cars, edge nodes) to collect real-time information (e.g. traffic, accidents, temporary roads closure) that are relevant with the car itinerary.
- The *Navigation service* is responsible of determining the shortest itinerary of the car trip. Based on the real-time information gathered by the *Collecting-data service*, the navigation itinerary is likely to be updated, on-the-fly, so that the car gets to destination with the minimum of delays.
- The *Decision service* implements the driving policies and specifies the car behavior in traffic condition (e.g. rolling into a traffic lane, brake and stop when traffic light is red, yield crossing pedestrians). The inputs of this module is provided by the *Perception service*.
- The *Storage service* takes care of saving the navigation data (e.g. traffic video stream) provided by the *Perception service*, as well as, the taken decisions by the *Decision service* in appropriate datacenters for prospective offline processing (e.g. optimization and learning purposes).

The *5G telco network domain* includes entities at the core and the edge of the network. On one hand, the core of the 5G telco network involves cloud data centers for storage purposes, as well as, 3GPP IP Multimedia Subsystem (IMS) servers that are required for the inner functioning of a telco network are deployed in the core. For instance, it hosts the Home Subscriber Server (HSS) responsible of managing the subscribed cars within the network. On the other hand, the edge carries the edge nodes that implement the MEC servers (e.g. smartphone, laptop) and representing potential

placement hosts for the latency-sensitive services of the autonomous car. Edge nodes could be mobile (e.g. smartphones) and might change their location during runtime.

The third domain represents the smart city infrastructure and facilities. According to mobile telecommunication networks specification, the city is split up into distinct geographic zones assimilated as logic Points of Presence (PoPs) of the 5G network. Each PoP has its own *Access Point*, *Distributed Antenna Systems* (DAS), *IMS Presence Server*, an *Edge Node Administrator* and an *Orchestrator Engine*. The *Presence Server* lists and tracks the edge nodes of a PoP. The *Edge Node Administrator* is responsible of managing the required runtime over them. The *Orchestration Engine* is responsible of deploying and executing the autonomous car services.

For this study, the outsourced service is the *Decision service* and th selected model is the Autoregressive Integrated Moving Average (ARIMA). The latter is often used for time series prediction. ARIMA models could represent different types of time series, e.g., exclusive AutoRegressive (AR), that predicts the future values of the time series based on its own p past behavior, i.e., lagged values; exclusive moving average (MA), that predicts the future values of the time series based on its own q past forecast errors; and mixed AR and MA (ARMA) series. The integrated part I of order d represents the number, i.e., d, of the differences needed to make the time series stationary. According to [140], an ARIMA model of orders p, d, and q, i.e., $ARIMA(p, d, q)$, can be written as:

$$y\prime_t = c + \phi_1 y\prime_{t-1} + \cdots + \phi_p y\prime_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t \tag{5.2}$$

where $y\prime_t$ is the differenced series, c is the intercept, $\phi_1 \cdots \phi_p$ represent the coefficients of the lagged values, $y\prime_{t-1} y \cdots \prime_{t-p}$ represent the lagged values of $y\prime_t$, $\theta_1 \cdots \theta_q$ represent the coefficients of the past forecast errors, $\varepsilon_{t-1} \cdots \varepsilon_{t-q}$ represent the past forecast errors and ε_t represents the white noise.

5.2.1 ARIMA model design and features configuration

Considering the three previously introduced parameters of ARIMA 'i.e., p, d and q), Algorithm 10 was designed to to determine d. Its associated complexity is $\mathcal{O}(n)$. Different d values are tested in a given range, determining the number of differences needed to make the time series stationary. A stationarity check is done for each selected value using the Augmented Dickey-Fuller (ADF) unit-root test on the differenced time series. The first value that makes the time series stationary is chosen.

The identification of p and q is made using a grid search method that takes into consideration all combinations of parameters in a given range and selects the combination with the lowest Akaike's Information Criterion (AIC) value as described by Algorithm 11. The AIC, introduced in [141], imposes a penalty on the adjustment quality of models with multiple parameters and is expressed by Equation 5.3, introduced in [142].

$$AIC = -2 \times LL + (log(n) + 1) \times NP \tag{5.3}$$

Algorithm for the detection of the parameter d

```
1  Compute-D(series, maxd)
       ; // series represents a training set
       ; // maxd represents the maximum number of differences
2      bestd ← −1
3      d ← 1
4      checkstat ← False
5      while ¬checkstat AND d ≤ maxd do
6          series ← difference(series)
7          checkstat ← adftest(series)
8          d ← d + 1
9      if checkstat then
10         bestd ← d − 1
11     return bestd
```

where LL is the likelihood function logarithm, n refers to the observations number of the TS, and NP comprises the parameters' number [142]. Algorithm 11 implements the designed procedure to detect the best combination of the ARIMA's orders p and q that minimizes the AIC. Its associated time complexity is $\mathcal{O}(n^2)$.

Grid search algorithm to determine the best p and q parameters

```
1  GridSearch(series, maxp, maxq, d, aicthreshold)
       ; // series represents a training set
       ; // maxp represents the maximum lag length of AR
       ; // maxq represents the maximum lag length of MA
2      bestaic ← aicthreshold
3      bestp ← 0
4      bestq ← 0
5      for p ← 1 to maxp do
6          for q ← 1 to maxq do
7              ar ← ARIMA(endog = series, order(p, d, q))
8              model ← ar.fit()
9              if bestaic < model.aic then
10                 bestaic ← model.aic
11                 bestp ← p
12                 bestq ← q
13     return bestp, bestq
```

The resulting ARIMA model along with the configured three parameters is applied

to the time series to obtain the predicted latency values.

5.2.2 Early validation and model evaluation

To validate and evaluate the prediction model, a realife dataset was generated from the running prototype introduced in [12]. The dataset contains 10000 entries listing several QoS metrics (i.e., latency, CPU workload, memory workload) that are collected, every single second, by an autonomous car during its movement. As for the latency, each line lists the real and concrete network latency values between the car (i.e., Soundfounder PiCar X[3]) and 3 different edge devices (i.e., Raspberry Pi3). The first edge device is mobile and changes location during runtime while the second and third devices are not mobile.

Prior to the model execution, several outliers detection techniques were tried on the real dataset. Similarly, the outliers detection scope, as well as, the outliers processing operations were varied each time. Table 5.1 details the most relevant combinations of the performed configurations for the outliers processing.

Table 5.1: List of defined configurations for outliers detection and processing

Configuration	Detection technique	Size	Detection scope	Operation
C1	Box Plot	Sliding window	Time series	Replacement
C2	Moving Mean	Sliding window	Time series	Replacement
C3	Moving Mean	Cumulative	Time series	Replacement
C4	Moving Mean	Cumulative	Time series	Removal
C5	Moving Mean	Cumulative	Residual	Removal

As for the evaluation, 3 metrics were considered, i.e., Mean Absolute Error (MAE), Mean Square Error (MSE) and Root MSE (RMSE). The obtained values are showed in Table 5.2. The reader should note that these values are truncated to 4 decimal digits for the RMSE and the MAE and 3 decimal digits for the MSE.

Table 5.2: The obtained prediction results with ARIMA

Configuration	RMSE	MSE	MAE
C1	$3.4206 * 10^{-}2$	$1.170 * 10^{-}3$	$2.8121 * 10^{-}2$
C2	$3.4127 * 10^{-}2$	$1.164 * 10^{-}3$	$2.8170 * 10^{-}2$
C3	$2.9305 * 10^{-}2$	$0.858 * 10^{-}3$	$2.5230 * 10^{-}2$
C4	$2.9379 * 10^{-}2$	$0.863 * 10^{-}3$	$2.5162 * 10^{-}2$
C5	$3.0321 * 10^{-}2$	$0.919 * 10^{-}3$	$2.5340 * 10^{-}2$

Based on the obtained results, the best configuration, for this specific case study and considering the given dataset, is (**C3**). Figure 5.5 depicts the obtained results

[3] `docs.sunfounder.com/projects/picar-x`

with a focus on the computed element size. It shows the obtained predictions for cumulative (**C3**) versus sliding window (**C2**). Figure 5.6 depicts the obtained results with a focus on the considered outliers processing operation. It shows the obtained predictions for the replacement operation (**C3**) versus the removal of the outliers operation (**C4**).

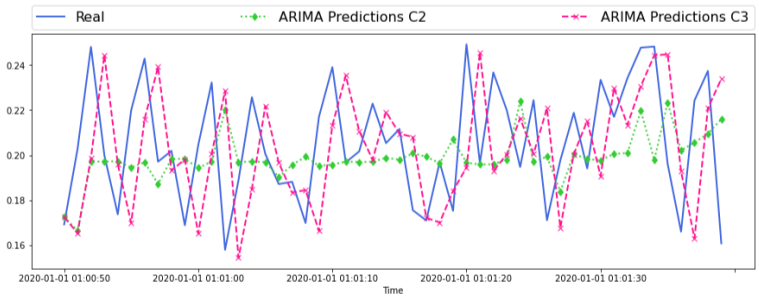


Figure 5.5: The obtained latency predictions - computed element size: Cumulative (C3) vs. Sliding window (C2)

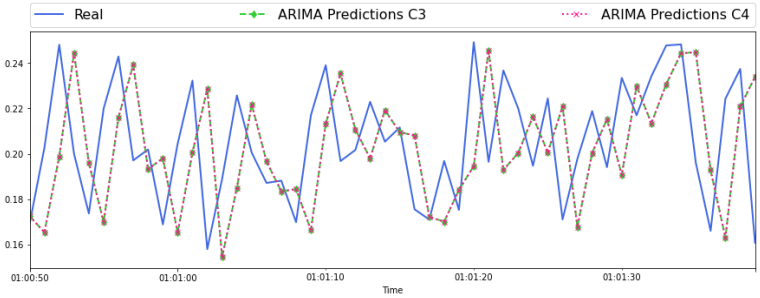


Figure 5.6: The obtained latency predictions - operation selection: Replacement (C3) vs. Removal (C4)

In the near future, it is intented to generalize this work and design generic proactive QoS management techniques to predict any prospective QoS degradation. The objective involves supporting any use case (e.g., autonomous cars, virtual/augmented reality, adaptive streaming) that would be provisioned in the cloud continuum. The target QoS techniques should be flexible and tunable enough to support the wide variety of these applications QoS requirements from one side and to take into consideration the hosting environments (e.g., hybrid cloud/edge, IoT, NFV). Furthermore, the associated data analytics models should be pluggable to optimize the accuracy and the proactive capability. Better yet, it is envisaged to design a framework that would be capable to dynamically switch from one data analytics model to another to optimize even more the predictions. Indeed, considering the constant changes in terms of network dynamicity and the strong mobility in the target environments, it is difficult to envision that one unique model could be the optimal one during the whole

runtime. Ideally, the changes in the network would also trigger a dynamic and automatic switch from one model to another at runtime. To that end, a comprehensive study of the data analytics models that support the time series will be conducted and the lessons learned will be used to build a knowledge database and a set of guidelines, useful to select the most relevant model considering the environement status and the applications requirements in terms of QoS.

5.3 Haptic communications and tactile Internet

A long term objective would be the achievement of the haptic communications in public service providers' infrastructures which would imply enabling the so-called "tactile Internet". Generally speaking, haptic communication is nonverbal and refers to the way humans communicate and interact via the sense of touch [143]. In computer networking, haptic communications refers to the emerging field of research that aims to improve human-human and human-machine interactions. Basically, it aims to augment traditional audiovisual communications by the haptic modality and henceforth offer to end-users the ability to physically get in touch with remote humans and objects. For instance, in [144], the authors propose a prototype that helps poeple suffering from phopbia. The prototype provides end-users suffering from phobia (e.g., touching spiders) to have access to a virtual and safe reality environment where they can have a therapy session under the guidance of a remote expert therapist.

The Tactile Internet has been defined by the International Telecommunication Union in August 2014 [145] as an Internet network featuring low latency, an extremely short transit, a high availability, high reliability and a high level of security. This technology rests on cloud computing proximity (i.e., multi-access edge computing) and the augmented/virtual reality for sensory and haptic controls [144]. Figure 5.7 depicts the reference architecture of the tactile Internet as it was introduced in [146]. This architecture has to meet to strict QoS requirements in terms of security, jitter, ultra-responsiveness and ultra-reliability. Basically, this could be translated to 1ms round trip latency between the master entity (e.g., glove equipped with sensors) and the slave entity (e.g., robot arm) going through the Radio Area Network (RAN) and the control servers at the edge of the network. Remote surgery and remote maintenance are among the most common use cases that the tactile Internet is expected to deliver.

Adhering to such strict QoS requirements impose strong management operation on the service providers' infrastructure. It is clear that efficient and reliable proactive QoS management is required to enable haptic communication between the master and the slave and, consequently, enable and maintain the tactile Internet during the runtime. To that end, service providers need (i) to anticipate any prospective breakdown in the network infrastructure or any eventual QoS degradation before it happens and (ii) to execute the necessary management actions (e.g., instantiate a new network resource, scale up the network resource, reconfigure the network resource, update the traffic rules of a network controller) in advance to prevent the QoS degradation. By

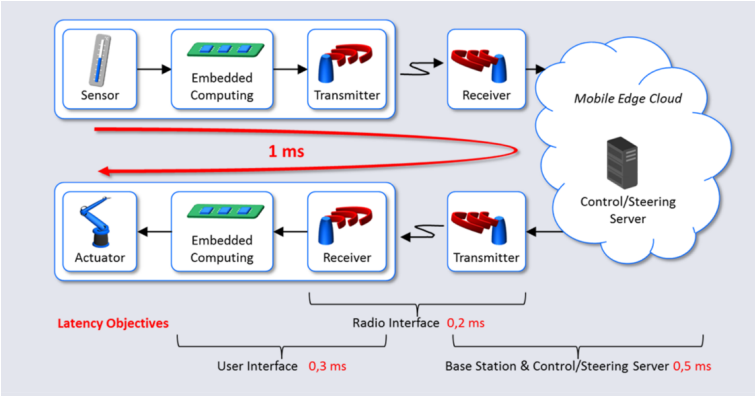


Figure 5.7: High-level architecture of the tactile Internet

doing so, the service providers could be able to maintain the required 1ms round trip latency. A possible starting alternative is to couple the classical QoS management techniques with appropriate and efficient machine learning models that should be able to provide the service providers with accurate predictions on the potential QoS variation over the time. This is challenging taking into consideration that, from one side, the traffic/data in the network are often stateless and, from the other side, the most accurate machine learning algorithms rely on stateful models. One can imagine introducing an assessment intermediate models that, considering a network traffic QoS dataset, it could inject the appropriate context, or simply adjust it, to the target use case before proceeding with the model training.

Chapter 6

Conclusion

The adoption of Service-Oriented Architecture (SOA) in cloud computing has pioneered the way how applications and services are built. One further noteworthy fact, cloud resources (e.g., datacenters, servers, storage volumes and even networks) has now come to be provisioned as SOA-basd software and according to the "Everything-as-a-Service" (XaaS) utility model. This means that the lifecycle of these resources is the very same lifecycle as the one introduced by the service computing paradigm in the 2000s.

The rise of the so-called next-generation networks, in conjunction with the evolution of the cloud computing to the cloud continuum, as well as, the advancement of the harwdare paving the way for machine learning support, have enabled novel emerging applications that range from unmanned systems (e.g., autonomous cars, car platooning, augmented/virtual reality and adaptive streaming are among the examples of emerging online applications. The fifth-generation/sixth-generation (5G/6G) wireless telco network, Content Delivery Networks (CDN) and Information-Centric Networks (ICN) are considered as next-generation networks and among the prospective hosting environments for these emerging applications.

Such applications and services are distributed, modular, reusable through composition techniques (e.g., service orchestration, service choreography) and might adopt novel design patterns that are even more distributed (e.g., micro services, Function-as-a-Service). Moreover, they are provisioned according to the service computing lifecycle management process. Broadly speaking, service developers design and develop the applications' services that will be deployed, executed and, then, managed, at runtime, in the target environment.

From one side, these emerging applications' services are getting more and more heterogeneous, distributed and mobile. They impose strong conditions and require strict Quality of Service (QoS) management on the hosting service providers. On the other side, the target runtime environments for these applications are getting more and more distributed, hetergenous, dynamic and mobile. Obviously, these bring with them their own pressures and opportunities from service lifecycle pointof view.

This research work focused on re-considering the service lifecycle management process to support the provisioning of novel and emerging applications in the cloud continuum and its inherent next-generation networks and infrastructures. It proposes several research contributions that cover and support every single step of the process.

Each chapter of this manuscript is dedicated to the work done for one or several related lifecycle steps. For practical reasons and for the sake of efficiency, the target contributions mostly focus on NFV domain and consider the Virtualized Network Functions (VNF) as the service building block of the cloud continuum.

For the service *design*, *publication* and *discovery*, a domain-independent VIrtualized networK functIoN ontoloGy (VIKING for short) that enables a comprehensive and generic description of the VNF capabilities from functional and non-functional perspectives. In addition, a semantic-based matchmaker that relies on VIKING to ensure the best matching between requested VNFs and published ones were designed and contributes to enabling accurate and automatic VNF discovery. As for validation, a prototype called "Mastermyr Chest tool box", including VIKING's instantiation along with the matchmaker in Content Delivery Networks (CDN) domain was implemented. This prototype illustrates a new way to contribute to the redesign of the CDN's traditional architecture by enabling value-added CDN service provisioning in an agile and dynamic manner. The performed experiments highlight the value-added and the reasonable overhead with regad to classical semantic-based approaches such as Web Ontology Language for Web Services (OWL-S).

For the service *instantiation* and *deployment*, a dynamic and agile VNF wiring (service composition) and instantiation (service deployment) approach is designed and implemented. This contribution introduces a model, as well as, an exhaustive procedure to: (i) draw the composition of a set of elementary VNF, (ii) instantiate and deploy the resulting Network Service (NS) in a target environment. This work advocated for "routing VNFs" with re-configurable wiring capabilities as a novel concept that would allow agile and dynamic VNFs wiring from design-time to run-time. As for validation, a proof-of-concept was developed as a plugin on top of the ETSI Open Source MANO orchestrator. The performed experiments show that the proposed approach: (i) performs better than the SDN-based solution in terms of latency and Mean Time-To-Operation (MTTO), and (ii) shows that the associated cost, with regard to SDN-based solution, remains acceptable.

For service *management*, a coordination-free orchestration for consistent VNF reconfiguration at runtime was introduced. The proposed model supports a prospective NS deployment under multi-domain federations while taking into consideration the non-functional dependencies between the involved VNF in the orchestration. Unlike the current state of the art, the proposed approach skips the coordination phase, necessary in ETSI NFV to avoid inconsistencies, and offers strong eventual consistency while supporting the non-functional dependencies during the reconfiguration. Two variants of the reconfiguration algorithm were designed: (i) A preventive variant, where transient inconsistent states are prevented, and (ii) a corrective variant , where intermediary inconsistent states are tolerated during the updating process. The performed evaluation in reallife cloud-based network federation showed that the preventive variant is stable and its performance is similar with different parameters while the corrective variant is sensitive to parameters. With high delays, the correc-

tive variant behaves like the preventive one. With low delays, it offers performance similar to the ETSI standard reconfiguration procedure but without the coordination phase and with the support of non-functional dependencies during the process.

Potential research directions for this work would be designing novel and appropriate models and procedures to cover additional aspects of service *management* lifecycle phase such as the predictive QoS management. An early investigation work proved the relevance and the feasbility of this research direction. The considered case study implements a next-generation autonomous car that interact with edge nodes in its neighborhood to offload part of its processing and rely on predictive network latency model to place part of the cars' services over the remote edge nodes. Yet another research perspective would be the adaptation and the inegration of innovative mechanisms that would allow such applications, provisioned in the cloud continuum, to dynamically select the most appropriate machine learning models for the data analytics (e.g., the QoS prediction) and, even more, switch from one model to another at runtime considering a pre-defined list of criteria such as the temporal data stationarity, the selected QoS metrics and the requirements of the running applications.

Appendix 1: List of Publications

- For journal publications, the Scimago Journal & Country Rank (SJR) ranking (2023) and the journal impact factor (2023) are indicated for each entry.
- For conference, demonstration and poster publications, Era Core 2023 ranking is indicated for each entry.
- The post PhD publications (2015-2023) are in red.

Peer-Reviewed Journal Articles

1. X. Li, Z. Zhou, Q. He, Z. Shi, W. Gaaloul, S. Yangui. Re-Scheduling IoT Services in Edge Networks. IEEE Transactions on Network and Service Management, 2023, in press. **(SJR Q1, Impact Factor = 5.3)**
2. D. Zhao, Z. Zhou, Z. Cai, S. Yangui, X. Xue. ASTL: Accumulative STL with a Novel Robustness Metric for IoT Service Monitoring. IEEE Transactions on Mobile Computing, Vol.22, Issue.10, pp. 5751-5768, 2023, IEEE. **(SJR Q1, Impact Factor = 7.9)**
3. H. Sliman, I. Megdiche, L. Alajramy, A. Taweel, S. Yangui, A. Drira, E. Lamine: MedWGAN based synthetic dataset generation for Uveitis pathology. Intelligent Systems with Applications. Vol.18. p. 200223, 2023, ELSEVIER. **(SJR Q1, Impact Factor = 9.4)**
4. S. Ghrab, I. Lahyani, S. Yangui, M. Jmaiel: A Core IoT Ontology for Automation Support in Edge Computing. Service Oriented Computing and Applications. Vol.17, Issue.1, pp. 25-37, 2023, Springer. **(SJR Q2, Impact Factor = 1.3)**
5. J. C Cisneros, S. E. Pomares Hernandez, J. C. Perez Sansalvador, L. M. Rodriguez-Henriquez, S. Yangui, K. Drira. Coordination-free Multi-domain NFV Orchestration for Consistent VNF Forwarding Graph Reconfiguration. IEEE Transactions on Network and Service Management, Vol.19, Issue.4, pp.5133-5151, 2022, IEEE. **(SJR Q1, Impact Factor = 5.3)**
6. J. C Cisneros, S. Yangui, S. E. Pomares Hernandez, K. Drira. A survey on distributed NFV multi-domain orchestration from an algorithmic functional perspective. IEEE Communications Magazine, Vol.60, Issue. 8, pp. 60-65, 2022, IEEE. **(SJR Q1, Impact Factor = 11.2)**
7. J. C Cisneros, S. E. Pomares Hernandez, S. Yangui, J. C. Perez Sansalvador, L. M. Rodriguez-Henriquez, K. Drira. VNF-based network service consistent reconfiguration in multi-domain federations: A distributed approach. Journal of

Network and Computer Applications, Vol. 195, pp. 103226, 2021, ELSEVIER. **(SJR Q1, Impact Factor = 8.7)**

8. N. E. Nouar, S. Yangui, N. Faci, K. Drira, S. Tazi. A Semantic virtualized network functions description and discovery model. Journal of Computer Networks, Vol.195, pp. 108152, 2021, ELSEVIER. **(SJR Q1, Impact Factor = 5.6)**

9. A. Yahyaoui, T. Abdellatif, S. Yangui, R. Attia. READ-IoT: Reliable Event and Anomaly Detection Framework for the Internet of Things. IEEE Access, Vol.9, pp. 24168-24186, 2021, IEEE. **(SJR Q1, Impact Factor = 3.9)**

10. S. Yangui. A Panorama of Cloud Platforms for IoT Applications Across Industries. Sensors, Vol.20, Issue 9, 2701, 2020, MDPI. **(SJR Q1, Impact Factor = 3.9)**

11. A. Soltanian, F. Belqasmi, S. Yangui, M. A. Salahuddin, R. H. Glitho, H. Elbiaze. A Cloud-Based Architecture for Multimedia Conferencing Service Provisioning. IEEE Access, Vol.6, pp. 9792-9806, 2018, IEEE. **(SJR Q1, Impact Factor = 3.9)**

12. C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, P. A. Polakos. A Comprehensive Survey on Fog Computing: State-of-the-art and Research Challenges. IEEE Surveys and Tutorials, Vol.20, Issue 1, pp.416-464, 2018, IEEE. **(SJR Q1, Impact Factor = 35.6)**

13. S. Yangui, S. Tata. An OCCI Compliant Model for PaaS Resources Description and Provisioning. The Computer Journal: Science and Mathematics, Vol.59, Issue 3, pp.308-324, 2016, Oxford Journals. **(SJR Q2, Impact Factor = 1.4)**

14. S. Yangui, R. H. Glitho, C. Wette. Approaches to End-user Applications Portability in the Cloud: A Survey. IEEE Communications Magazine, Vol.54, Issue 7, pp.138-145, 2016, IEEE. **(SJR Q1, Impact Factor = 11.2)**

15. S. Yangui, S. Tata. The SPD Approach to Deploy Service-based Applications in the Cloud*****. Concurrency and Computation: Practice and Experience, Vol.17, Issue 15, pp.3943-3960, 2015, John Wiley & Sons Ltd. **(SJR Q2, Impact Factor = 2)**
***Best Research Publication Award - Université Val d'Essonne**.

16. S. Yangui, I.J. Marshall, J.P. Laisne, Samir Tata. CompatibleOne: The Open Source Cloud Broker. Journal of Grid Computing, Vol.12, Issue 1, pp.93-109, 2014, Springer Netherlands. **(SJR Q1, Impact Factor = 4.1)**

17. S. Moalla, S. Yangui. Discovery of Capacity-driven Web services. International Journal of Web Applications, Vol.2, Issue 4, pp.261-279, 2010, DLine Editor. **(Impact Factor = 0.7)**

Peer-Reviewed Conference Papers

18. A. Abusafia, A. Bouguettaya, A. lakhdari, S. Yangui: Context-Aware Trustworthy IoT Energy Services Provisioning, Internationl Conference on Service-Oriented Computing, ICSOC'23, in press, Rome, Italy, November 28-December 1, 2023. **(Era core A)**

19. D. Zhao, Z. Zhou, X. Xue, J. Diao, S. Yangui, B Liu, W Gaaloul: A Novel Logic-Based Adaptive Monitoring for Composite Edge Services. The IEEE International Conference on Web Services, IEEE ICWS'23, pp. 310-317, Chicago, USA, July 2-8, 2023. **(Era core A)**

20. H. Sliman, I. Megdiche, S. Yangui, A. Drira, I. Drira, E. Lamine: A Synthetic Dataset Generation for the Uveitis Pathology Based on MedWGAN Model. The ACM/SIGAPP Symposium On Applied Computing, ACM/SIGAPP SAC'23, pp. 559-566, Tallinn, Estonia, March 27-32, 2023. **(Era core B)**

21. D. Zhao, Z. Zhou, Z. Cai, T. Long, S. Yangui, X. Xue. ASTL: Accumulative Signal Temporal Logic for IoT Service Monitoring. The IEEE International Conference on Web Services, IEEE ICWS'22, pp. 256-265, Barcelona, Spain, July 11-25, 2022. **(Era core A)**

22. N. E. Nouar, S. Yangui, N. Faci, K. Drira, S. Tazi. Agile and Dynamic Virtualized Network Functions Wiring in Network Services. The IEEE Conference on Cloud Computing, IEEE CLOUD'21, pp. 322-332, virtual conference, October 18-24, 2021. **(Era core B)**

23. J.C. Cisneros, S. Yangui, S.E. Pomares Hernandez, J.C Perez Sansalvador, L. M. Rodriguez-Henriquez, K. Drira. Towards Consistent VNF Forwarding Graph Reconfiguration in Multi-domain Environments. The IEEE Conference on Cloud Computing, IEEE CLOUD'21, pp. 355-366, virtual conference, October 18-24, 2021. **(Era core B)**

24. X. Li, Z. Zhou, Z. Zhao, S. Yangui, W. Zhang. Data and Computation-Intensive Service Re-Scheduling In Edge Networks. International Conference on Web Services, IEEE ICWS'21, pp.389-396, virtual conference, October 18-24, 2021. **(Era core A)**

25. J. Tang, X. Xue, S. Yangui, Z. Zhou. Efficient Search for Moving Object Devices in Internet of Things Networks. International Conference on Web Services, IEEE ICWS'20, pp.454-462, virtual conference, October 19-23, 2020. **(Era core A)**

26. J.C. Cisneros, S. Yangui, S.E. Pomares Hernandez, J.C Perez Sansalvador, K. Drira. Coordination Algorithm for Migration of Shared VNFs in Federated Environments. IEEE Conference on Network Softwarization, NetSoft'20, pp.252-256, virtual conference, June 29- July 3, 2020. **(Era core B)**

27. F. Raissi, S. Yangui, F. Camps. Autonomous Cars, 5G Mobile Networks and Smart Cities: Beyond the Hype*. IEEE International Conference on Collaboration Technologies and Infrastructures, Wetice'19, pp.180-185, Capri, Italy, June 12-14, 2019. **(Era core C)**
Best Research Paper Award.

28. M. Aloui, H. Elbiaze, R.H. Glitho, S. Yangui. Analytics as-a-Service Architecture for Cloud-based CDN: Case of Video Popularity Prediction. IEEE Consumer Communications & Networking Conference, CCNC'18, pp. 1-4 Las Vegas, NV, USA, January 12-15, 2018. **(Era core B)**

29. C. Mouradian, S. Yangui, R.H. Glitho. Robots as-a-Service in Cloud computing: Search and Rescue in Large-scale Disasters Case Study. IEEE Consumer Communications & Networking Conference, CCNC'18, pp.1-7, Las Vegas, NV, USA, January 12-15, 2018. **(Era core B)**

30. M.A Naceur, S. Yangui, S. Tata, R.H. Glitho. Provisioning of Component-based Applications Across Multiple Clouds. International Conference on Cloud Computing and Services Science, CLOSER'17, pp104-114, Porto, Portugal, April 24 -26, 2017.

31. N.T Jahromi, S. Yangui, A. Larabi, D. Smith, M.A. Salahuddin, R.H. Glitho, R. Brunner, H. Elbiaze. NFV and SDN-based Cost-efficient and Agile Value-added Video Services Provisioning in Content Delivery Networks. IEEE Consumer Communications & Networking Conference, CCNC'17, pp.671-677, Las Vegas, NV, USA, January 8-11, 2017. **(Era core B)**

32. S. S Chauhan, S. Yangui, R.H. Glitho, C. Wette. A Case Study for a Presence Service in the Cloud. IEEE International Conference on the Network of the Future, NoF'16, pp. 1-7, Buzios, Brazil, November 16-18, 2016.

33. M. Abu-Lebdeh, S. Yangui, D. Naboulsi, R.H. Glitho, C. Wette. A Virtual Network PaaS for 3GPP 4G and Beyond Core Network Services. IEEE International Conference on Cloud Networking, Cloudnet'16, pp.7-13, Pisa, Italy, October 3-5, 2016.

34. A.F. Bin Alam, A. Soltanian, S. Yangui, M.A Salahuddin, R.H. Glitho, H. Elbiaze. A Cloud Platform as-a-Service for Multimedia Conferencing Service Provisioning. IEEE Symposium on Computers and Communication, ISCC'16, pp.289-294, Messina, Italy, June 27-30, 2016. **(Era core B)**

35. S. Yangui, P. Ravidran, O. Bibani, R.H. Glitho, N.B. Hadj Alouane, M. Morrow, P. Polakos. A Platform as-a-Service for Hybrid Cloud/Fog Environments. IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN'16, pp.1-7, Rome, Italy, June 13-15, 2016. **(Era core C)**

36. S. Yangui, K. Klai, S. Tata. Deployment of Service-based Processes in the Cloud using Petri Net Decomposition. International Conference on COOPERATIVE INFORMATION SYSTEMS, CoopIS'14, pp.57-74, Amantea, Italy, October, 2014. **(Era core B)**

37. S. Yangui, M. Nasrallah, S. Tata. PaaS-independent Approach to Provision Appropriate Cloud Resources for SCA-based Applications Deployment. IEEE International Conference on Semantics, Knowledge & Grids, Beijing, SKG'13, pp. 14-21, Beijing, China, October, 2013. **(Era core - National China)**

38. M. Sellami, S. Yangui, M. Mohamed, S. Tata. PaaS-independent Provisioning and Management of Applications in the Cloud. IEEE International Conference on Cloud Computing, CLOUD'13, pp. 693-700, Santa Clara Marriott, CA, USA, June-July, 2013. **(Era core B)**

39. S. Yangui, S. Tata. CloudServ: PaaS Resources Provisioning for Service-based Applications. IEEE International Conference on Advanced Information Networking and Applications, AINA'13, pp.522-529, Barcelona, Spain, March 25-28, 2013. **(Era core B)**

40. A. Omezzine, S. Yangui, N. Bellamine, S. Tata. Mobile Service Micro-containers for Cloud Environments. IEEE International Conference on Collaboration Technologies and Infrastructures, Wetice'12, pp.154-160, Toulouse, France, June 2012. **(Era core C)**

41. S. Yangui, M. Mohamed, S. Tata, S. Moalla. Scalable Service Containers*. IEEE International Conference on Cloud Computing Technology and Science, CloudCom'11, pp.348-356, Athens, Greece, November 29 - December 1, 2011. **(Era core C)**
Best Research Paper Award.

42. M. Mohamed, S. Yangui, S. Moalla, S. Tata. Service Micro-container for Service-based Applications in Cloud Environments. IEEE International Conference on Collaboration Technologies and Infrastructures, Wetice'11. pp.61-66, Paris, France, June 27-29, 2011. **(Era core C)**

Peer-Reviewed Demos, Posters & Workshops Papers

43. F. Raissi, C.A. Ouedraogo, S. Yangui, F. Camps, N.B. Hadj-Alouane. Paving the Way for Autonomous Cars in the City of Tomorrow: A Prototype for Mobile Devices Support at the Edges of 5G Network*. Internationl Conference on Service-Oriented Computing, ICSOC'18, pp.481-485, Hangzhou, Zhejiang, China, November 12-15, 2018. **(Era core A)**
Best Demo Award.

44. C.A. Ouedraogo, E.F. Bonfoh, S. Medjiah, C. Chassot, S. Yangui. A Prototype for Dynamic Provisioning of QoS-oriented Virtualized Network Functions in the Internet of Things. IEEE Conference on Network Softwarization, NetSoft'18, pp.323-325, Montreal, QC, Canada, June 25- 29, 2018. **(Era core B)**

45. N.T Jahromi, S. Yangui, S. Shanmugasundaram, A. Rangy, R.H. Glitho, A. Larabi, D. Smith, R. Brunner. A Prototype for Value-added Video Services Provisioning in Content Delivery Networks*. IEEE Consumer Communications & Networking Conference, CCNC'17, Las Vegas, NV, USA, January 8-11, 2017. **(Era core B)**
Runner-up Demo Award.

46. O.Bibani, S. Yangui, C. Mouradian, R.H. Glitho, W. Gaaloul, N.B. Hadj-Alouane, M. Morrow, P. Polakos. A demo of IoT Healthcare Application Provisioning in Hybrid Cloud/Fog Environment. IEEE International Conference on Cloud Computing Technology and Science, CloudCom'16, Luxembourg City, Luxembourg, December 12-15, 2016. **(Era core C)**

47. O.Bibani, S. Yangui, R.H. Glitho, W. Gaaloul, N.B. Hadj Alouane, M.Morrow, P. Polakos. A Demo of a PaaS for IoT Applications Provisioning in Hybrid Cloud/Fog Environment*. IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN'16, pp.1-2, Rome, Italy, June 13-15, 2016. **(Era core B)**
Runner-up Demo Award.

48. S. Yangui, R.H. Glitho, F. Belqasmi, M. Morrow, P. Polakos. IoT End-user Applications Provisioning in the Cloud: State of the Art. IEEE International Conference on Cloud Engineering, IC2E'16, pp.232-233, Berlin, Germany, April 4-8, 2016.

49. I. Khan, F.Z Errounda, S. Yangui, R.H Glitho, N. Crespi. Getting Virtualized Wireless Sensor Networks' IaaS Ready for PaaS. DCOSS'15, International workshop on Internet of Things and Ideas and Perspectives, Fortaleza, Brazil, June, 2015. **(Era core B)**

50. K. Gaaloul, S. Yangui, S. Tata, H.A. Proper. Architecting Access Control for Business Processes in the Cloud. IEEE International Workshop on Advanced Information Systems for Enterprises, IWAISE'14, Hammamet, Tunisia, November, 2014.

51. S. Yangui, S. Tata. Paas Elements for Hosting Service-based Applications. International Conference on Cloud Computing and Services Science, CLOSER'12, pp.476-479, Porto, Portugal, April 2012.

Appendix 2: Teaching Activities and Perspectives

LIST OF TAUGHT COURSES

INSA Toulouse (September 2017 - Present)

- Cloud computing and virtualization technologies,
- (Cloud and edge computing) (previously Cloud computing and adaptability),
- Containerization and DevOps,
- Big data architectures and frameworks,
- Software-defined communication infrastructure,
- Software design (UML),
- Object-oriented programming (advanced JAVA),
- ADA programming.

Concordia University (2015 - 2017)

- Cloud networking and service provisioning,
- Higher layer telecommunications protocols.

Télécom SudParis & Télécom Ecole de Management (2011 - 2014)

- Relational databases,
- Application design (UML),
- Object-oriented programming (Java),
- Multi-tier applications.

Faculty of Science of Tunis (2009 - 2011)

- Web programming,
- Graphic user interfaces,
- Algorithmics and data structures,

- Applied-mathematics programming (Matlab).

It should be noted that additional Master courses were tought as visiting scholar or guest lecturer in several foreign universities. The list below enumerates some examples (non-comprehensive list):

- Distributed systems, ISIMM, University of Monastir, Tunisia (2020, 2021)
- Service and Business Process Management, China University of Geosciences in Beijing, China (2018)
- Internet of Things, ENIT, University of Tunis EL-Manar, Tunisia (2017, 2018)

TEACHING MOCKUP & TIMETABLE

The following tables detail the comprehensive list of the taught courses. The first subsection describes the teaching activities from 2017 (INSA appointment) to present while the second subsection describes the teaching activities during the PhD and the postdoc appointments.

Teaching courses and classes (2017 - Present)

INSA Toulouse (1192 Hours)			
Year	**Course**	**Class**	**Hours**
2023-24	Virtualization & cloud computing	Engineering (Y5)/Master	26 hours
	Containerization and DevOps	Engineering (Y5)	12 hours
	Big data architectures and frameworks	Engineering (Y5)	31 hours
	Advanced software design	Engineering (Y4)	14 hours
	Advanced Java	Engineering (Y4)	70 hours
	Integrator project	Engineering (Y5)	24 hours
	Cloud and edge computing	Engineering (Y5)	25 hours
	Collaborative tools and methodologies	Engineering (Y4)	10 hours
2022-23	Virtualization & cloud computing	Engineering (Y5)/Master	26 hours
	Big data architectures and frameworks	Engineering (Y5)	28 hours
	Advanced software design	Engineering (Y4)	14 hours
	Advanced Java	Engineering (Y4)	70 hours
	Integrator project	Engineering (Y5)	24 hours
	Cloud computing and adaptability	Engineering (Y5)	18 hours
	Software-defined communication infrastructure	Engineering (Y5)	12 hours
2021-22	Virtualization & cloud computing	Engineering (Y5)/Master	26 hours
	Big data architectures and frameworks	Engineering (Y5)	31 hours
	Advanced software design	Engineering (Y4)	14 hours
	Advanced Java	Engineering (Y4)	70 hours
	Integrator project	Engineering (Y5)	24 hours
	Cloud computing and adaptability	Engineering (Y5)	18 hours
2020-21	Virtualization & cloud computing	Engineering (Y5)/Master	26 hours
	Big data architectures and frameworks	Engineering (Y5)	31 hours
	Advanced software design	Engineering (Y4)	14 hours
	Advanced Java	Engineering (Y4)	20 hours
	Integrator project	Engineering (Y5)	24 hours
	Cloud computing and adaptability	Engineering (Y5)	18 hours

INSA Toulouse (1192 Hours)			
Year	**Course**	**Class**	**Hours**
2019-20	Virtualization & cloud computing	Engineering (Y5)/Master	23 hours
	Big data architectures and frameworks	Engineering (Y5)	28 hours
	UML & object-oriented programming	Engineering (Y4)	61 hours
	XML language	Engineering (Y5)	37 hours
	Integrator project	Engineering (Y5)	15 hours
	Cloud computing and adaptability	Engineering (Y5)	18 hours
2018-19	Virtualization & cloud computing	Engineering (Y5)/Master	23 hours
	Big data architectures and frameworks	Engineering (Y5)	28 hours
	UML & object-oriented programming	Engineering (Y4)	61 hours
	ADA programming	Common Curriculum (Y1)	21 hours
	Integrator project	Engineering (Y5)	11 hours
	Cloud computing and adaptability	Engineering (Y5)	18 hours
2017-18	Virtualization & cloud computing	Engineering (Y5)/Master	23 hours
	Big data architectures and frameworks	Engineering (Y5)	28 hours
	Object-oriented programming	Engineering (Y4)	33 hours
	ADA programming	Common Curriculum (Y1)	21 hours
	Integrator project	Engineering (Y5)	19 hours
	Cloud computing and adaptability	Engineering (Y5)	4 hours
Total : **1192 hours**			

Teaching courses and classes (2010 - 2017)

Concordia University (120 Hours)			
Year	**Course**	**Class**	**Hours**
2016–2017	Cloud networking and service provisioning	Graduate students	20 hours
	Higher layer telecommunications protocols	Graduate students	20 hours
2015–2016	Cloud networking and service provisioning	Graduate students	20 hours
	Higher layer telecommunications protocols	Graduate students	20 hours
2014–2015	Cloud networking and service provisioning	Graduate students	20 hours
	Higher layer telecommunications protocols	Graduate students	20 hours
Institut Mines-Télécom, Télécom SudParis (99 Hours)			
Year	**Course**	**Class**	**Hours**
2013–2014	Relational databases	Bachelor (B2)	15 hours
	Relational databases	Master (M1)	9 hours
	Databases	Bachelor (B2)	15 hours
	UML design & object-oriented programming	Engineering (E2)	27 hours
	Multi-tier applications	Engineering (E2)	21 hours
	Algorithmics & Java programming	Bachelor (B1)	12 hours
Faculty of Science of Tunis, University Tunis El Manar (150 Hours)			
Year	**Course**	**Class**	**Hours**
2009–2010	Web programming	Bachelor (B3)	15 hours
	Algorithmics & data structures	Core Curriculum (L1)	30 hours
2010–2011	Graphic User Interfaces	Bachelor (B3)	30 hours
	Algorithmics & C programming	Bachelor (B2)	30 hours
	Oriented-object programming	Bachelor (B3)	15 hours
	Maple Programming	Core Curriculum (L2)	30 hours

Perspectives and future projects

The various teaching activities could be classified into three categories: (i) software engineering (e.g., software design, programming), virtualization and distributed systems (e.g., cloud computing, edge computing, service containers), frameworks and solutions for big data (e.g., NoSQL, Hadoop and MapReduce, Spark).

For the software engineering category, a logical and natural evolution of the teaching courses would be the inclusion of the emerging engineering methodologies such as the model-driven engineering (i.e., from examples to knowledge) and the design thinking. Yet another perspective would be the integration of formal and agile methodologies (e.g., SCRUM, SAFE) that are now set to become a culture and a mindset in the IT compagnies.

As for the second category, it is envisaged to evolve the current courses material with a focus on service automation. For instance, the adoption of the DevOps methodologies and tools could even better leverage and bring to light the potential

of the cloud continuum in service engineering (e.g., collaborative coding, automatic testing, continuous integration, continuous development). Obviously, this ties in with the discussed evolution and the perspectives of the software engineering category.

When it comes to the frameworks and solutions for big data, the current courses material is oriented "data-engineering". A short-term objective is upskilling, not only the materials, but also the course scope to involve and cover "data science" as well. This includes data analytics and machine learning models in particular.

Last but not least, a promising and challenging teaching perspective would be setting up and supervising cross-cutting and multidisciplinary study projects where students might need additional skills, in addition to computer engineering, to understand, model, design, implement and evaluate the required solutions. Software business, Information and Communications Technology (ICT) entrepreneurship are among the required additional skills for carrying out such projects.

Bibliography

[1] The service definition in the cambridge dictionary. https://dictionary.cambridge.org/dictionary/english/service, May 2023.

[2] The principle of services and services computing. In *Services Computing*, pages 3–19. Springer Berlin Heidelberg.

[3] B. Medjahed, B. Benatallah, A. Bouguettaya, A.H.H. Ngu, and A.K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB Journal The International Journal on Very Large Data Bases*, 12(1):59–85, may 2003.

[4] Sami Yangui. *Service-based applications provisioning in the cloud. (Déploiement des applications à base de services dans le cloud)*. PhD thesis, Telecom & Management SudParis, Évry, Essonne, France, 2014.

[5] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C. Narendra, and Bo Hu. Everything as a service (xaas) on the cloud: Origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628, 2015.

[6] Christof Ebert and Lorin Hochstein. Devops in practice. *IEEE Software*, 40(1):29–36, 2023.

[7] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site reliability engineering: How Google runs production systems.* " O'Reilly Media, Inc.", 2016.

[8] Vodafone taas: A telecommunications paradigm shift. https://vodafone.bblabs.co.uk/vodafone-taas-a-telecommunications-paradigm-shift/, May 2023.

[9] Hanif Ullah, Nithya Gopalakrishnan Nair, Adrian Moore, Chris Nugent, Paul Muschamp, and Maria Cuevas. 5g communication: An overview of vehicle-to-everything, drones, and healthcare use-cases. *IEEE Access*, 7:37251–37268, 2019.

[10] Amine Abouaomar, Soumaya Cherkaoui, Zoubeir Mlika, and Abdellatif Kobbane. Resource provisioning in edge computing for latency-sensitive applications. *IEEE Internet of Things Journal*, 8(14):11088–11099, 2021.

[11] Fatma Raissi, Clovis Anicet Ouedraogo, Sami Yangui, Frédéric Camps, and Nejib Ben Hadj-Alouane. Paving the way for autonomous cars in the city of tomorrow: A prototype for mobile devices support at the edges of 5g network. In Xiao

Liu, Michael Mrissa, Liang Zhang, Djamal Benslimane, Aditya Ghose, Zhongjie Wang, Antonio Bucchiarone, Wei Zhang, Ying Zou, and Qi Yu, editors, *Service-Oriented Computing - ICSOC 2018 Workshops - ADMS, ASOCA, ISYyCC, CloTS, DDBS, and NLS4IoT, Hangzhou, China, November 12-15, 2018, Revised Selected Papers*, volume 11434 of *Lecture Notes in Computer Science*, pages 481–485. Springer, 2018.

[12] Fatma Raissi, Sami Yangui, and Frederic Camps. Autonomous cars, 5g mobile networks and smart cities: Beyond the hype. In *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 180–185, 2019.

[13] Lingling Lv, Yanjun Shi, and Weiming Shen. Mobility-as-a-service research trends of 5g-based vehicle platooning. *Service Oriented Computing and Applications*, 15:1 – 3, 2020.

[14] Heekwang Kim and Kwangsue Chung. Multipath-based http adaptive streaming scheme for the 5g network. *IEEE Access*, 8:208809–208825, 2020.

[15] Daniel S. Berger. Towards lightweight and robust machine learning for CDN caching. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. ACM, nov 2018.

[16] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2018.

[17] Hani Attar, Haitham Issa, Jafar Ababneh, Mahdi Abbasi, Ahmed AA Solyman, Mohammad Khosravi, Ramy Said Agieb, et al. 5g system overview for ongoing smart applications: Structure, requirements, and specifications. *Computational Intelligence and Neuroscience*, 2022, 2022.

[18] Sami Yangui. A panorama of cloud platforms for iot applications across industries. *Sensors*, 20(9), 2020.

[19] Suryaveer Singh Chauhan, Sami Yangui, Roch H. Glitho, and Constant Wette. A case study for a presence service in the cloud. In *2016 7th International Conference on the Network of the Future (NOF)*, pages 1–7, 2016.

[20] Mohammad Abu-Lebdeh, Sami Yangui, Diala Naboulsi, Roch Glitho, and Constant Wette Tchouati. A virtual network paas for 3gpp 4g and beyond core network services. In *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, pages 7–13, 2016.

[21] Jose Ordonez-Lucena, Pablo Ameigeiras, Diego Lopez, Juan J Ramos-Munoz, Javier Lorca, and Jesus Folgueira. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, 55(5):80–87, 2017.

[22] *The Principle of Services and Services Computing*, pages 3–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[23] Sami Yangui, Roch H. Glitho, and Constant Wette. Approaches to end-user applications portability in the cloud: A survey. *IEEE Communications Magazine*, 54(7):138–145, 2016.

[24] Nour el houda Nouar, Sami Yangui, Noura Faci, Khalil Drira, and Saïd Tazi. A semantic virtualized network functions description and discovery model. *Comput. Networks*, 195:108152, 2021.

[25] Online, February 2020.

[26] Ulrich Küster, Birgitta König-Ries, Mirco Stern, and Michael Klein. Diane: an integrated approach to automated service discovery, matchmaking and composition. In *Proceedings of the 16th international conference on World Wide Web*, pages 1033–1042, 2007.

[27] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, et al. Owl-s: Semantic markup for web services. *W3C member submission*, 22(4), 2004.

[28] John Domingue, Dumitru Roman, and Michael Stollberg. Web service modeling ontology (wsmo)-an ontology for semantic web services, 2005.

[29] John F. Sowa. Semantic networks, 1987.

[30] Stuart C Shapiro and William J Rapaport. Sneps considered as a fully intensional propositional semantic network. In *The knowledge frontier*, pages 262–315. Springer, 1987.

[31] Shaun Voigt, Catherine Howard, Dean Philp, and Christopher Penny. Representing and reasoning about logical network topologies. In Madalina Croitoru, Pierre Marquis, Sebastian Rudolph, and Gem Stapleton, editors, *Graph Structures for Knowledge Representation and Reasoning*, pages 73–83, Cham, 2018. Springer International Publishing.

[32] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - (SIGCOMM)*. ACM Press, 2003.

[33] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[34] Committee Specification. Tosca simple profile for networkfunctions virtualization (nfv) version 1.0. techreport, OASIS, May 2017.

[35] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. Tosca: portable automated deployment and management of cloud applications. In *Advanced Web Services*, pages 527–549. Springer, 2014.

[36] Marouen Mechtri, Chaima Ghribi, Oussama Soualah, and Djamal Zeghlache. Nfv orchestration framework addressing sfc challenges. *IEEE Communications Magazine*, 55(6):16–23, 2017.

[37] Georgios Xilouris, Eleni Trouva, Felicia Lobillo, João M Soares, Jorge Carapinha, Michael J McGrath, George Gardikis, Pietro Paglierani, Evangelos Pallis, Letterio Zuccaro, et al. T-nova: A marketplace for virtualized network functions. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE, 2014.

[38] Tm forum information framework (sid) gb922. Technical report, November 2013.

[39] Joao Soares, Miguel Dias, Jorge Carapinha, Bruno Parreira, and Susana Sargento. Cloud4nfv: A platform for virtual network functions. In *2014 IEEE 3Rd international conference on cloud networking (cloudnet)*, pages 288–293. IEEE, 2014.

[40] L. C. Hoyos and C. E. Rothenberg. Non: Network function virtualization ontology towards semantic service implementation. In *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6, Nov 2016.

[41] I. Oliver, S. Panda, K. Wang, and A. Kalliola. Modelling nfv concepts with ontologies. In *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 1–7, Feb 2018.

[42] N. Bouten, M. Claeys, R. Mijumbi, J. Famaey, S. Latré, and J. Serrat. Semantic validation of affinity constrained service function chain requests. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 202–210, June 2016.

[43] 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.

[44] Erl Thomas. Soa principles of service design. *Boston: Prentice Hall*, 37:71–75, 2007.

[45] Andrea Ordanini and Paolo Pasini. Service co-production and value co-creation: The case for a service-oriented architecture (soa). *European Management Journal*, 26(5):289–297, 2008.

[46] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, 6(2):86–93, 2002.

[47] Aaron E Walsh. *Uddi, Soap, and WSDL: the web services specification reference book*. Prentice Hall Professional Technical Reference, 2002.

[48] Xiaofeng Tao, Yan Han, Xiaodong Xu, Ping Zhang, and Victor C. M. Leung. Recent advances and future challenges for mobile network virtualization. *Science China Information Sciences*, 60(4):040301, Mar 2017.

[49] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, apr 2010.

[50] Luis Cuellar Hoyos and Christian Esteve Rothenberg. Non: Network function virtualization ontology towards semantic service implementation. In *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE, 2016.

[51] Yasmine M Afify, Nagwa L Badr, Ibrahim F Moawad, and Mohamed F Tolba. A comprehensive business domain ontology for cloud services. In *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 134–143. IEEE, 2017.

[52] Artan Mazrekaj, Isak Shabani, and Besmir Sejdiu. Pricing schemes in cloud computing: An overview. 2016.

[53] Network Functions Virtualisation NFV. Etsi gs nfv-sec 009 v1. 1.1 (2015-12). 2015.

[54] Shankar Lal, Tarik Taleb, and Ashutosh Dutta. Nfv: Security threats and best practices. *IEEE Communications Magazine*, 55(8):211–217, 2017.

[55] Jagruti Sahoo, Mohammad Ali Salahuddin, Roch H. Glitho, Halima Elbiaze, and Wessam Ajib. A survey on replica server placement algorithms for content delivery networks. *IEEE Communications Surveys & Tutorials*, 19:1002–1026, 2016.

[56] Meisong Wang, Prem Prakash Jayaraman, Rajiv Ranjan, Karan Mitra, Miranda Zhang, Eddie Li, Samee Khan, Mukkaddim Pathan, and Dimitrios Georgeakopoulos. *An Overview of Cloud Based Content Delivery Networks: Research Dimensions and State-of-the-Art*, pages 131–158. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[57] The streaming media magazine. Online, 2020.

[58] 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*, 42(4):13–24, 2012.

[59] N. Bouten, J. Famaey, R. Mijumbi, B. Naudts, J. Serrat, S. Latré, and F. De Turck. Towards nfv-based multimedia delivery. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 738–741, May 2015.

[60] Narjes T. Jahromi, Sami Yangui, Adel Larabi, Daniel Smith, Mohammad A. Salahuddin, Roch H. Glitho, Richard Brunner, and Halima Elbiaze. NFV and sdn-based cost-efficient and agile value-added video services provisioning in content delivery networks. In *14th IEEE Annual Consumer Communications & Networking Conference, CCNC 2017, Las Vegas, NV, USA, January 8-11, 2017*, pages 671–677, 2017.

[61] R. Song, Z. Luo, J-R. Wen, Y. Yu, and H-W. Hon. Identifying ambiguous queries in web search. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 1169–1170, 2007.

[62] Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with owls-mx. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, page 915–922, New York, NY, USA, 2006. Association for Computing Machinery.

[63] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[64] Nour el houda Nouar, Sami Yangui, Noura Faci, Khalil Drira, and Saïd Tazi. Agile and dynamic virtualized network functions wiring in network services. In Claudio Agostino Ardagna, Carl K. Chang, Ernesto Daminai, Rajiv Ranjan, Zhongjie Wang, Robert Ward, Jia Zhang, and Wensheng Zhang, editors, *14th IEEE International Conference on Cloud Computing, CLOUD 2021, Chicago, IL, USA, September 5-10, 2021*, pages 322–332. IEEE, 2021.

[65] Network functions virtualisation (nfv);management and orchestration. Technical report, ETSI GS NFV-MAN 001 V1.1.1, 2014.

[66] Network functions virtualisation (nfv);use cases. Technical report, ETSI GR NFV 001 V1.2.1, 2017.

[67] M. Boban, A. Kousaridas, K. Manolakis, J. Eichinger, and W. Xu. Connected roads of the future: Use cases, requirements, and design considerations for vehicle-to-everything communications. *IEEE Vehicular Technology Magazine*, 13(3):110–123, 2018.

[68] Hanwen Cao, Sandip Gangakhedkar, Ali Ramadan Ali, Mohamed Gharba, and Josef Eichinger. A 5g v2x testbed for cooperative automated driving. In *2016 IEEE Vehicular Networking Conference (VNC)*, pages 1–4. IEEE, 2016.

[69] Nathan F Saraiva De Sousa, Danny A Lachos Perez, Raphael V Rosa, Mateus AS Santos, and Christian Esteve Rothenberg. Network service orchestration: A survey. *Computer Communications*, 2019.

[70] Ahmed M Medhat, Tarik Taleb, Asma Elmangoush, Giuseppe A Carella, Stefan Covaci, and Thomas Magedanz. Service function chaining in next generation networks: State of the art and research challenges. *IEEE Communications Magazine*, 55(2):216–223, 2016.

[71] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[72] Molka Gharbaoui, S Fichera, Piero Castoldi, and Barbara Martini. Network orchestrator for qos-enabled service function chaining in reliable nfv/sdn infrastructure. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5. IEEE, 2017.

[73] Eder J Scheid, Cristian C Machado, Ricardo L dos Santos, Alberto E Schaeffer-Filho, and Lisandro Z Granville. Policy-based dynamic service chaining in network functions virtualization. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 340–345. IEEE, 2016.

[74] Franco Callegati, Walter Cerroni, Chiara Contoli, and Giuliano Santandrea. Dynamic chaining of virtual network functions in cloud-based edge networks. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5. IEEE, 2015.

[75] Junjie Liu, Wei Lu, Fen Zhou, Ping Lu, and Zuqing Zhu. On dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management*, 14(3):543–553, 2017.

[76] Zoltan Zsoka and Khalil Mebarkia. Layered solutions for dynamic service chaining. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 292–296. IEEE, 2019.

[77] AA Mohammed, Molka Gharbaoui, Barbara Martini, Federica Paganelli, and Piero Castoldi. Sdn controller for network-aware adaptive orchestration in dynamic service chaining. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 126–130. IEEE, 2016.

[78] Barbara Martini and Federica Paganelli. A service-oriented approach for dynamic chaining of virtual network functions over multi-provider software-defined networks. *Future Internet*, 8(2):24, 2016.

[79] Attila Csoma, Balázs Sonkoly, Levente Csikor, Felicián Németh, Andràs Gulyas, Wouter Tavernier, and Sahel Sahhaf. Escape: Extensible service chain prototyping environment using mininet, click, netconf and pox. *ACM SIGCOMM Computer Communication Review*, 44(4):125–126, 2014.

[80] A. Campanella. Intent based network operations. In *2019 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, 2019.

[81] Y. Han, J. Li, D. Hoang, J. Yoo, and J. W. Hong. An intent-based network virtualization platform for sdn. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 353–358, 2016.

[82] Mariam Kiran, Eric Pouyoul, Anu Mercian, Brian Tierney, Chin Guok, and Inder Monga. Enabling intent to configure scientific networks for high performance demands. *Future Generation Computer Systems*, 79:205–214, 2018.

[83] F. Paganelli, F. Paradiso, M. Gherardelli, and G. Galletti. Network service description model for vnf orchestration leveraging intent-based sdn interfaces. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2017.

[84] Open vswitch,.

[85] Paul Quinn, Uri Elzur, and Carlos Pignataro. Network service header (nsh). In *RFC 8300*. RFC Editor, 2018.

[86] Michele Segata, Renato Lo Cigno, and Falko Dressler. Towards communication strategies for platooning, 2013.

[87] Irshad Ahmed Sumra, Halabi Bin Hasbullah, et al. Effects of attackers and attacks on availability requirement in vehicular network: a survey. In *2014 International Conference on Computer and Information Sciences (ICCOINS)*, pages 1–6. IEEE, 2014.

[88] Giuseppe Araniti, Massimo Condoluci, Pasquale Scopelliti, Antonella Molinaro, and Antonio Iera. Multicasting over emerging 5g networks: Challenges and perspectives. *Ieee network*, 31(2):80–89, 2017.

[89] Konstantinos Antonakoglou, Xiao Xu, Eckehard Steinbach, Toktam Mahmoodi, and Mischa Dohler. Toward haptic communications over the 5g tactile internet. *IEEE Communications Surveys & Tutorials*, 20(4):3034–3059, 2018.

[90] Patricia T Endo, Moisés Rodrigues, Glauco E Gonçalves, Judith Kelner, Djamel H Sadok, and Calin Curescu. High availability in clouds: systematic review and research challenges. *Journal of Cloud Computing*, 5(1):1–15, 2016.

[91] Kostas Katsalis, Vasilis Sourlas, Thanasis Korakis, and Leandros Tassiulas. A cloud-based content replication framework over multi-domain environments. In *2014 IEEE International Conference on Communications (ICC)*, pages 2926–2931, 2014.

[92] Ricard Vilalta, Arturo Mayoral, Ramon Casellas, Ricardo Martínez, and Raul Muñoz. Sdn/nfv orchestration of multi-technology and multi-domain networks in cloud/fog architectures for 5g services. In *2016 21st OptoElectronics and Communications Conference (OECC) held jointly with 2016 International Conference on Photonics in Switching (PS)*, pages 1–3, 2016.

[93] Raphael Vicente Rosa, Mateus Augusto Silva Santos, and Christian Esteve Rothenberg. Md2-nfv: The case for multi-domain distributed network functions virtualization. In *2015 International Conference and Workshops on Networked Systems (NetSys)*, pages 1–5, 2015.

[94] Nathan F. Saraiva de Sousa, Danny A. Lachos Perez, Raphael V. Rosa, Mateus A.S. Santos, and Christian Esteve Rothenberg. Network Service Orchestration: A survey. *Computer Communications*, 142-143(May):69–94, jun 2019.

[95] ETSI, NFVISG. GS NFV-SOL 004 V2. 3.1 Network Functions Virtualisation (NFV) release 2; Protocols and Data Models; NFV descriptors based on YANG Specification, 2019.

[96] Omar Houidi, Oussama Soualah, Wajdi Louati, and Djamal Zeghlache. Dynamic VNF Forwarding Graph Extension Algorithms. *IEEE Transactions on Network and Service Management*, 17(3):1389–1402, sep 2020.

[97] Josué Castañeda Cisneros, Sami Yangui, Saúl E. Pomares Hernández, Julio César Pérez Sansalvador, Lil María Rodríguez-Henríquez, and Khalil Drira. Towards consistent VNF forwarding graph reconfiguration in multi-domain environments. In Claudio Agostino Ardagna, Carl K. Chang, Ernesto Daminai, Rajiv Ranjan, Zhongjie Wang, Robert Ward, Jia Zhang, and Wensheng Zhang, editors, *14th IEEE International Conference on Cloud Computing, CLOUD 2021, Chicago, IL, USA, September 5-10, 2021*, pages 355–366. IEEE, 2021.

[98] ETSI, NFVISG. ETSI GS NFV 002 V1.1.1 Network Functions Virtualisation (NFV); Architectural Framework, 2013.

[99] Josué Castañeda Cisneros, Saúl E. Pomares Hernández, Julio César Pérez Sansalvador, Lil María Rodríguez-Henríquez, Sami Yangui, and Khalil Drira. Coordination-free multi-domain NFV orchestration for consistent VNF forwarding graph reconfiguration. *IEEE Trans. Netw. Serv. Manag.*, 19(4):5133–5151, 2022.

[100] Josué Castañeda Cisneros, Sami Yangui, Saúl E. Pomares Hernández, and Khalil Drira. A survey on distributed NFV multi-domain orchestration from an algorithmic functional perspective. *IEEE Commun. Mag.*, 60(8):60–65, 2022.

[101] Luis M. Vaquero, Felix Cuadrado, Yehia Elkhatib, Jorge Bernal-Bernabe, Satish N. Srirama, and Mohamed Faten Zhani. Research challenges in nextgen service orchestration. *Future Generation Computer Systems*, 90:20–38, 2019.

[102] Wei Ren, R W Beard, and E M Atkins. A survey of consensus problems in multiagent coordination. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 1859–1864 vol. 3, jun 2005.

[103] Heidi Howard and Richard Mortier. Paxos vs Raft: Have we reached consensus on distributed consensus? *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data, PaPoC 2020*, pages 8–10, 2020.

[104] Y Xiao, N Zhang, W Lou, and Y T Hou. A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Communications Surveys Tutorials*, 22(2):1432–1465, 2020.

[105] Peter Bailis and Ali Ghodsi. Eventual Consistency Today: Limitations, Extensions, and Beyond. *Queue*, 11(3):20–32, mar 2013.

[106] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-Free Replicated Data Types. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[107] Werner Vogels. Eventually Consistent: Building Reliable Distributed Systems at a Worldwide Scale Demands Trade-Offs?Between Consistency and Availability. *Queue*, 6(6):14–19, oct 2008.

[108] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. *Europe*, 2011.

[109] ETSI. Dgr/nfv-ifa028. network functions virtualisation (nfv) release 3; management and orchestration; report on architecture options to support multiple administrative domains. techreport, ETSI, January 2018.

[110] Gao Zheng, Anthony Tsiopoulos, and Vasilis Friderikos. Dynamic VNF Chains Placement for Mobile IoT Applications. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, dec 2019.

[111] Pham Tran Anh Quang, Abbas Bradai, Kamal Deep Singh, Gauthier Picard, and Roberto Riggio. Single and Multi-Domain Adaptive Allocation Algorithms for VNF Forwarding Graph Embedding. *IEEE Transactions on Network and Service Management*, 16(1):98–112, 2019.

[112] Changwoo Kim, Yujeong Oh, and Jaiyong Lee. Latency-based graph selection manager for end-to-end network service on heterogeneous infrastructures. In *2018 International Conference on Information Networking (ICOIN)*, pages 534–539. IEEE, jan 2018.

[113] M Zeng, W Fang, and Z Zhu. Orchestrating Tree-Type VNF Forwarding Graphs in Inter-DC Elastic Optical Networks. *Journal of Lightwave Technology*, 34(14):3330–3341, jul 2016.

[114] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. A Green VNF-FG Embedding Algorithm. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 141–149. IEEE, jun 2018.

[115] S Khebbache, M Hadji, and D Zeghlache. Dynamic Placement of Extended Service Function Chains: Steiner-based Approximation Algorithms. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pages 307–310, oct 2018.

[116] Bart Spinnewyn, Steven Latré, and Juan Felipe Botero. Delay-constrained NFV orchestration for heterogeneous cloud networks. *Computer Networks*, 180:107420, 2020.

[117] Josué Castañeda Cisneros, Sami Yangui, Saul E Pomares Hernández, Julio César Pérez Sansalvador, Lil M Rodríguez Henríquez, and Khalil Drira. Towards Consistent VNF Forwarding Graph Reconfiguration in Multi-domain Environments. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 355–366, 2021.

[118] P T Anh Quang, Y Hadjadj-Aoul, and A Outtagarts. Evolutionary Actor-Multi-Critic Model for VNF-FG Embedding. In *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–6, jan 2020.

[119] Pham Tran Anh Quang, Abbas Bradai, Kamal Deep Singh, and Yassine Hadjadj-Aoul. Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach. In *IEEE INFOCOM 2019 - IEEE Conference on*

Computer Communications Workshops (INFOCOM WKSHPS), pages 886–891. IEEE, apr 2019.

[120] Yue Hao, Yi Li, Xinghua Dong, Li Fang, and Ping Chen. Performance Analysis of Consensus Algorithm in Private Blockchain. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 280–285. IEEE, jun 2018.

[121] Iqbal Alam, Kashif Sharif, Fan Li, Zohaib Latif, M M Karim, Sujit Biswas, Boubakr Nour, and Yu Wang. A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV. *ACM Computing Surveys*, 53(2):1–40, mar 2021.

[122] G Gardikis, S Costicoglou, H Koumaras, Ch. Sakkas, A Kourtis, F Arnal, L M Contreras, P Aranda Gutierrez, and M Guta. NFV applicability and use cases in satellite networks. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 47–51. IEEE, jun 2016.

[123] Min Chen, Wei Li, Giancarlo Fortino, Yixue Hao, Long Hu, and Iztok Humar. A dynamic service migration mechanism in edge cognitive computing. *ACM Transactions on Internet Technology (TOIT)*, 19:1 – 15, 2018.

[124] Tao Ouyang, Rui Li, Xu Chen, Zhi Zhou, and Xin Tang. Adaptive user-managed service placement for mobile edge computing: An online learning approach. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1468–1476, 2019.

[125] Yuxuan Sun, Xueying Guo, Sheng Zhou, Zhiyuan Jiang, Xin Liu, and Zhisheng Niu. Learning-based task offloading for vehicular cloud computing systems. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, 2018.

[126] Yuxuan Sun, Sheng Zhou, and Jie Xu. Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2637–2646, 2017.

[127] Zehong Lin, Suzhi Bi, and Ying-Jun Angela Zhang. Optimizing ai service placement and resource allocation in mobile edge intelligence systems. *IEEE Transactions on Wireless Communications*, 20(11):7257–7271, 2021.

[128] Nathaniel Hudson, Hana Khamfroush, and Daniel E. Lucani. Qos-aware placement of deep learning services on the edge with multiple service implementations. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–8, 2021.

[129] Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, 36(10):2333–2345, 2018.

[130] Chao-Lun Wu, Te-Chuan Chiu, Chih-Yu Wang, and Ai-Chun Pang. Mobility-aware deep reinforcement learning with glimpse mobility prediction in edge computing. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–7, 2020.

[131] Amin Azari, Panagiotis Papapetrou, Stojan Denic, and Gunnar Peters. User traffic prediction for proactive resource management: Learning-powered approaches. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[132] Bin Gao, Zhi Zhou, Fangming Liu, and Fei Xu. Winning at the starting line: Joint network selection and service placement for mobile edge computing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1459–1467, 2019.

[133] Vajiheh Farhadi, Fidan Mehmeti, Ting He, Tom La Porta, Hana Khamfroush, Shiqiang Wang, and Kevin S Chan. Service placement and request scheduling for data-intensive applications in edge clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1279–1287, 2019.

[134] Jan Plachy, Zdenek Becvar, and Emilio Calvanese Strinati. Dynamic resource allocation exploiting mobility prediction in mobile edge computing. In *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6, 2016.

[135] Huirong Ma, Zhi Zhou, and Xu Chen. Predictive service placement in mobile edge computing. In *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 792–797, 2019.

[136] Lucas Pacheco, Denis Rosário, Eduardo Cerqueira, and Leandro Villas. Service migration in edge computing environments for connected autonomous vehicles. In *Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 533–546, Porto Alegre, RS, Brasil, 2020. SBC.

[137] Lucas Pacheco, Denis Rosário, Eduardo Cerqueira, Leandro Villas, Torsten Braun, and Antonio A. F. Loureiro. Distributed user-centric service migration for edge-enabled networks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 618–622, 2021.

[138] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Latency-aware application module management for fog computing environments. *ACM Transactions on Internet Technology (TOIT)*, 19:1 – 21, 2018.

[139] Faizan Murtaza, Adnan Akhunzada, Saif ul Islam, Jalil Boudjadar, and Rajkumar Buyya. Qos-aware service provisioning in fog computing. *Journal of Network and Computer Applications*, 165:102674, 2020.

[140] Paul Newbold. Arima model building and the time series analysis approach to forecasting. *Journal of forecasting*, 2(1):23–35, 1983.

[141] Hirotogu Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer New York, New York, NY, 1998.

[142] Antonio Rafael Sabino Parmezan, Vinicius M.A. Souza, and Gustavo E.A.P.A. Batista. Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model. *Information Sciences*, 484:302–337, 2019.

[143] Kory Floyd Judee K Burgoon, Laura K Guerrero. *Nonverbal Communication*. Pearson, 1 edition, 2009.

[144] Yassine Jebbar, Fatna Belqasmi, Roch H. Glitho, and Omar Alfandi. A fog-based architecture for remote phobia treatment. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Sydney, Australia, December 11-13, 2019*, pages 271–278. IEEE, 2019.

[145] Thomas Wiegand Erich Zielinsk Hans Schotten Peter Merz Sandra Hirche Andreas Festag Walter Häffner Michael Meyer Eckehard Steinbach Rolf Kraemer Ralf Steinmetz Frank Hofmann Peter Eisert Reinhard Scholl Frank Ellinger Erik Weiss Ines Riedel Gerhard Fettweis, Holger Boche. The tactile internet. Technical report, ITU-T Technology Watch Report, 2014.

[146] Adnan Aijaz, Mischa Dohler, A. Hamid Aghvami, Vasilis Friderikos, and Magnus Frodigh. Realizing the tactile internet: Haptic communications over next generation 5g cellular networks. *IEEE Wireless Communications*, 24(2):82–89, 2017.