Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering (35^{th}cycle)

Machine Learning-Powered Management Architectures for Edge Services in 5G Networks

By

Corrado Puligheddu

******

Supervisor(s):
Prof. Carla Fabiana Chiasserini

Doctoral Examination Committee:
Dr. Raffaele Bruno, CNR
Prof. Paolo Giaccone, Politecnico di Torino
Prof. Isabelle Guérin Lassous, Université Claude Bernard Lyon 1
Prof. Tamer Khattab, Qatar University
Prof. Renato Lo Cigno, Università di Brescia

Politecnico di Torino
2022

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Corrado Puligheddu
2022

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

I would like to express my gratitude to all the people that shared with me even a small part of this 3 years journey. I want to explicitly extend my appreciation to the people without whom this work would not have been the same.

First of all, I have to thank my advisor Prof. Carla Fabiana Chiasserini for her invaluable continuous support in guiding me toward this result. I sincerely appreciate all the assistance, insights and advice she shared with me during these years. Her outstanding dedication and professionalism have truly been an inspiration. I would like to extend my gratitude also to Prof. Claudio Casetti for his guidance and support.

I want to thank all the colleagues and friends that I met at Politecnico. Among them, a big thank you to Giuseppe, who first showed me that work can be fun, to Francesco, who shared with me this journey not only in Italy but also in the USA, and Federico, for the coffees and headaches we had together. Thanks also to the whole research group, past and present members.

Next, I want to thank my closest friends. K and Andrea, who have been there since my first day at Politecnico, Matte and Cami, thanks to whom Turin feels more like home. Thanks also to Robee, Giuli, Jaci, Eli and all of my close friends in Cagliari.

Finally, I want to thank my parents and family for their long-distance encouragement and support. I can not express how much I appreciate you always being there for me.

PhD Activity

Journal publications

- Li, X.; Chiasserini, C. F.; Mangues-Bafalluy, J.; Baranda, J.; Landi, G.; Martini, B.; Costa-Perez, X.; **Puligheddu, C.**; Valcarenghi, L., Automated Service Provisioning and Hierarchical SLA Management in 5G Systems, in: *IEEE Transactions on Network and Service Management*, 2021

- Casetti, C.; Chiasserini, C. F.; Marcato, S.; **Puligheddu, C.**; Mangues-Bafalluy, J.; Baranda, J.; Brenes, J.; Bocchi, F.; Landi, G.; Bakhshi, B., ML-driven Provisioning and Management of Vertical Services in Automated Cellular Networks, in: *IEEE Transactions on Network and Service Management*, 2022

- Tripathi, S.; **Puligheddu, C.**; Chiasserini, C. F.; Mungari, F.; A Context-aware Radio Resource Management in Heterogeneous Virtual RANs, in: *IEEE Transactions on Cognitive Communications and Networking*, 2022

- Tripathi, S.; **Puligheddu, C.**; Pramanik, S.; Garcia-Saavedra, A.; Chiasserini, C. F.; Fair and Scalable Orchestration of Network and Compute Resources for Virtual Edge Services, submitted to: *IEEE Transactions on Mobile Computing* (major revision received), 2022

Book chapters

- Martin-Perez, J.; Magoula, L.; Antevski, K.; Guimaraes, C.; Baranda, J.; Chiasserini, C. F.; Sgambelluri, A.; Papagianni, C.; Garcia-Saavedra, A.; Martínez, R.; Paolucci, F.; Barmpounakis, S.; Valcarenghi, L.; Casetti, C.; Li, X.; Bernardos, C. J.; De Vleeschauwe, D.; De Schepper, K.; Kontopoulos, P.; Koursioumpas, N.; **Puligheddu, C.**; Mangues-Bafalluy, J.; Zeydan, E.; *Self Managed 5G Networks*

in *Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning*, 2021, John Wiley & Sons

Conference publications

- Baranda, J.; Mangues-Bafalluy, J.; Vettori, L.; Martinez, R.; Avino, G.; Chiasserini, C. F.; **Puligheddu, C.**; Casetti, C.; Brenes, J.; Landi, G.; Kondepu, K.; Paolucci, F.; Fichera, S.; Valcarenghi, L., Arbitrating Network Services in 5G Networks for Automotive Vertical Industry, in: *IEEE INFOCOM 2020 - Demo Session*, 2020

- Baranda, J.; Mangues-Bafalluy, J.; Zeydan, E.; Casetti, C.; Chiasserini, C. F.; Malinverno, M.; **Puligheddu, C.**; Groshev, M.; Guimaraes, C.; Tomakh, K.; Kucherenko, D.; Kolodiazhnyi, O., Demo: AIML-as-a-Service for SLA management of a Digital Twin Virtual Network Service, in: *IEEE INFOCOM 2021 - Demo Session*, 2021

- De Vleeschauwer, D.; Baranda, J.; Mangues-Bafalluy, J.; Chiasserini, C. F.; Malinverno, M.; **Puligheddu, C.**; Magoula, L.; Martin-Perez, J.; Barmpounakis, S.; Kondepu, K.; Valcarenghi, L.; Li, X.; Papagianni, C.; 5Growth Data-Driven AI-Based Scaling, in: *2021 EuCNC & 6G Summit*, 2021

- Tripathi, S.; **Puligheddu, C.**; Chiasserini, C. F.; An RL Approach to Radio Resource Management in Heterogeneous Virtual RANs, in: *IEEE/IFIP WONS 2021*, 2021

- Tripathi, S.; **Puligheddu, C.**; Pramanik, S.; Garcia-Saavedra, A.; Chiasserini, C. F.; VERA: Resource Orchestration for Virtualized Services at the Edge, in: *IEEE International Conference on Communications (IEEE ICC 2022)*, 2022

- Sgambelluri, A.; Baranda, J.; Groshev, M.; Tomakh, K.; Kucherenko, D.; Paolini, E.; Zanzi, L.; Xavier Salvat, J.; **Puligheddu, C.**; Malandrino, F.; Kolodiazhnyi, O.; Bernardos, C. J.; De la Oliva, A.; Mangues-Bafalluy, J.; Chiasserini, C. F.; Li, X.; Valcarenghi, L.; Exploiting Forecasting for Automatic Network Service Operations in Digital Twin Applications, in: *IEEE International Conference on Sensing, Communication, and Networking (IEEE SECON 2022)*, 2022

Attended classes and passed tests

Name	Hours
Advanced deep Learning	30
Big data processing and programming	20
Communication	5
Connected vehicles	20
Data mining concepts and algorithms	20
Data science for networks	20
Entrepreneurial finance	5
Parallel and distributed computing	25
Advanced scientific programming in matlab	28
Project management	5
Public speaking	5
Research integrity	5
Thinking out of the box	1
Time management	2
Writing scientific papers in english	15

External training activity

Name	Location	Hours
Computing@Polito - HPC/Big Data/Cloud for Research	Politecnico di Torino	4
Lipari School on Advanced Networking Systems	Lipari Island, Italy	30

Abstract

Next-generation mobile networks are designed to allow vertical industries to offer a broad range of virtualized services to their users. However, increasingly evolved mobile services force more and more demanding performance requirements, which are particularly hard to guarantee, without expensive overprovisioning, in case of time-varying service and traffic demands. This work addresses the issue by taking two complementary approaches.

First, we propose a centralized automation solution for the provisioning of edge services and management of edge resources. Using service scaling (i.e., allocating more resources to a service, only when needed, to manage more traffic) the proposed solution can ensure the satisfaction of performance requirements of an automotive vertical service. We then improve our solution by integrating the concept of ML-as-a-Service (MLaaS) through a MLaaS Platform able to train and serve ML models to the elements of a 5G system, thus giving them the possibility of making smarter decisions. We demonstrate the new capabilities of our proposal by developing two ML-driven algorithms for network slice subnet sharing and run-time service scaling. Results show that service performance can be always satisfied while saving 30% on OPEX.

The second approach investigates distributed RAN orchestration and edge resource management using reinforcement learning. In this case, the decision-making logic is colocated with the services and applications it controls, allowing local fine-grained and low-latency actions. We propose two reinforcement learning frameworks for edge resource management: CAREM and VERA. CAREM operates in heterogeneous vRANs; it is able to select the best available radio link and the transmission parameters, enabling efficient radio resource allocation in time-varying scenarios. CAREM exhibits excellent performance when compared both to the closest existing scheme based on neural networks, and to a contextual bandit approach. Instead, VERA addresses the concurrent execution of two kinds of services at the edge, namely user applications and network functions. We show that often the computing resources required by these services are entangled since the data processed by the former has to be transferred by the latter and vice versa.

Acknowledging this complex dynamic, we propose a scalable reinforcement learning-based framework to orchestrate resources at the edge. Considering as services an LTE vRAN and a video transcoder, we demonstrate that VERA is able to meet services KPIs over 96% of the observation periods.

Contents

Chapter 1

Introduction

In the last couple of decades, mobile networks have become a fundamental pillar of modern society and are going to acquire a progressively more pervasive and ubiquitous role moving into the future. Not only the number of active subscribers is going to increase, mostly as a consequence of the growing penetration rate in developing countries, but the evolution of demanding use cases leveraging mobile networks is going to cause a surge in worldwide mobile data traffic. Many of these use cases will have a profound positive impact on our society. However, video streaming, online gaming, augmented reality, industry 4.0, connected vehicles, and many more such novel applications, which are going to be requested by the most disparate vertical companies, are going to require performance that is just not attainable with the current generation of mobile network technology. According to the recent Ericsson Mobility Report, in 2021 more than 8 billion worldwide mobile subscriptions were exchanging 84 EB/month of data traffic. In 2027, 9 billion subscribers are forecast to exchange as much as 368 EB/month of data traffic[1]. To accommodate the exponential growth of data traffic and the more and more demanding application performance requirements, it's imperative that the 5G mobile network technology is set to offer better throughput, delays and reliability while being more energy and spectrum efficient.

To achieve this performance improvement, two key concepts have been selected to build the foundations of 5G and beyond networks: Network Function Virtualization (NFV) and Edge Computing.

NFV is a network architecture concept that envisions the split of network node functions in virtualized elementary blocks called Virtual Network Functions (VNFs) that can be interconnected to build advanced network services. This way, an unprecedented

level of flexibility is allowed in managing network services, which allows for advanced optimizations of the network operations. The interested reader can find more details in [2].

Edge Computing enables cloud computing capabilities at the edge of the mobile network, to run applications and process data closer to the end user so as to minimize the propagation delays. In fact, cloud data center locations are not chosen to minimize the latency to the user, rather they are built considering more pressing factors such as: high bandwidth internet connection availability, energy, land and building cost, and low risk of natural disasters. Instead, mobile operators have the possibility of using existing widespread RAN sites to host computing capabilities at the very edge of the network, thus avoiding the costs of new dedicated sites. A more complete discourse on the topic can be found in [3].

In addition to the aforementioned concepts, **Machine Learning (ML)** has also become more and more prominent in mobile networks. It has been proven to be a powerful technique for network automation, with particularly promising results in the management of Radio Access Networks (RAN). Indeed, the research community agrees on the pivotal role of ML-based techniques in the future of mobile networks.

As it will be better detailed in the next sections, this work exploits these three concepts to advance the state of the art in the management and orchestration of edge services in 5G networks. More specifically, this thesis will investigate which techniques based on the aforementioned concepts can be utilized in the management of 5G networks to guarantee Service Level Agreements (SLAs), and what performance can be obtained when these techniques are applied to mission-critical edge services with a time-varying load. Many mission-critical services, in many fields such as e-health and automotive, will require low-latency connectivity and extreme reliability for an immediate and deterministic response time, that, if not always obtained, could cost human lives.

We will take advantage of edge computing to design, develop and evaluate mission-critical automotive applications that require low latency and reliable connectivity to properly operate. The NFV paradigm will be of paramount importance to orchestrate edge services, especially when considering load fluctuations due to the occasional peaks in the number of mobile users requesting the service, to guarantee the steady performance of mission-critical services. Indeed, if necessary, low-priority services will be suspended to free resources and improve the responsiveness of higher-priority load-stressed services. Finally, we will investigate how Machine Learning can be integrated to provide the intelligence needed to treat unexpected events that conventional algorithms fail to

properly manage, and attest whether they can provide added value when integrated into 5G management platforms.

The rest of the chapter will first present the research questions that have driven the research activity and the main contributions of the present work, then it will detail the outline of the thesis.

1.1 Research questions and main contributions

The main contribution of this thesis is the design, development and evaluation of automation architectures and management solutions for virtualized 5G mobile networks with a focus on ML-based decision-making. This work runs along two parallel tracks: the **centralized management of edge services** and the **distributed orchestration of network and compute resources for edge services**. These share the need for careful management of edge services in dynamic conditions mindful of the Service Level Agreements (SLAs) between network operators and the services developers. In fact, a static configuration of the network can't be adequate in every possible network condition and for every edge service, unless network resources are expensively overprovisioned. Instead, a swift adaptation of the network configuration to the data traffic, radio conditions and resource availability can better tailor the services' needs without requiring overprovisioning of network resources. The difference between these two parallel tracks stands in the placement of the orchestration intelligence, which also affects the time scales over which decisions are provided. In the first case, network management is performed from the centralized standpoint of the network operator, thus having complete knowledge and control of the network architecture. This possibly leads to high-level global optimal decisions, although inevitably coarse and non-real-time. Conversely, in the second case, network automation can be performed directly at the edge in a distributed manner, with the advantage of real-time and fine-grained local decisions. Notice that these two approaches are not in contrast with each other, rather they can be deployed concurrently to achieve the best results.

In the rest of the section, a more in-depth description of these two topics will be provided, specifying the research questions that have driven the research activity and our contribution to addressing them. We also provide details of our dissemination activity.

1.1.1 Centralized management of edge services in 5G networks

Network management is considered from the centralized standpoint of their core network, where, through a multi-layer and multi-domain platform, Network Operators are able to deploy vertical network services starting from high-level descriptors provided by the verticals. The descriptors are ultimately mapped to configurations of VNFs, PNFs and links that implement the edge services according to the vertical needs. However, a static network configuration may not be well suited for every possible scenario, so it is the responsibility of the management platform to dynamically adapt the service configuration to avoid SLA violations. In this context, we have addressed the following main research questions (RQ).

RQ1: How to provide automated service provisioning and hierarchical SLA management in 5G systems?

Specifically, how can network and compute resources, spanning multiple geographical sites, be automatically provisioned to a mission-critical automotive edge service? Which techniques can be utilized to orchestrate edge services of different priorities to ensure the respect of vertical SLAs when resources are limited and the services are subject to time-varying traffic load? What performance can be expected from such a system?

To answer RQ1, we design, develop and evaluate architectures and procedures for the SLA-aware management of network edge services. Collaborating within the 5G-TRANSFORMER EU projects, first, we develop an SLA management solution based on the service scaling mechanism and then we evaluate it in an automotive safety application, with tests on the field using real cars.

This part is covered in detail in Chapter 2. To foster further research activities, the code base has been released as open source[1]. Our contributions to international conferences and journals can be found in:

1. Baranda, J.; Mangues-Bafalluy, J.; Vettori, L.; Martinez, R.; Avino, G.; Chiasserini, C. F.; **Puligheddu, C.**; Casetti, C.; Brenes, J.; Landi, G.; Kondepu, K.; Paolucci, F.; Fichera, S.; Valcarenghi, L., Arbitrating Network Services in 5G Networks for Automotive Vertical Industry, in: *IEEE INFOCOM 2020 - Demo Session*, 2020

[1] https://github.com/5g-transformer

2. Baranda, J.; Mangues-Bafalluy, J.; Zeydan, E.; Casetti, C.; Chiasserini, C. F.; Malinverno, M.; **Puligheddu, C.**; Groshev, M.; Guimaraes, C.; Tomakh, K.; Kucherenko, D.; Kolodiazhnyi, O., Demo: AIML-as-a-Service for SLA management of a Digital Twin Virtual Network Service, in: *IEEE INFOCOM 2021 - Demo Session*, 2021

3. Li, X.; Chiasserini, C. F.; Mangues-Bafalluy, J.; Baranda, J.; Landi, G.; Martini, B.; Costa-Perez, X.; **Puligheddu, C.**; Valcarenghi, L., Automated Service Provisioning and Hierarchical SLA Management in 5G Systems, in: *IEEE Transactions on Network and Service Management*, 2021

RQ2: How to provide ML-driven provisioning and management in automated cellular networks?

In particular, acknowledging the promising capabilities of Machine Learning applied to network management, how can ML capabilities be integrated into a state-of-the-art 5G platform to improve the decision-making? What advantages can be expected by leveraging ML algorithms to control mission-critical edge services?

To answer RQ2, we leverage the 5G-TRANSFORMER architecture used to address R1 and, within the 5Growth EU project, evolve it in the 5Growth platform. Aware of the promising capabilities of Machine Learning (ML) applied to network management, we develop a ML-as-a-Service (MLaaS) Platform and integrate it into the 5Growth system to assist the different entities of the Management and Orchestration (MANO) architecture with ML-powered decision-making. The MLaaS Platform takes care of the centralized training and serving of the ML models required by the entities of MANO platform. Leveraging the MLaaS Platform, two ML-driven algorithms for network slice subnet sharing and run-time service scaling have been developed. We show that network reconfigurations commanded by these two algorithms take place in just a matter of seconds. Results show considerable savings in operational costs while still respecting SLAs. The code base is entirely open source[2] and can be used to reproduce the obtained results, as well as a basis for a commercial product. This part is covered in detail in Chapter 3.

Our contributions to international conferences and journals addressing this topic can be found in:

[2] https://github.com/5growth

1. De Vleeschauwer, D., Baranda, J., Mangues-Bafalluy, J., Chiasserini, C. F., Malinverno, M., **Puligheddu, C.**, Magoula, L., Martin-Perez, J., Barmpounakis, S., Kondepu, K., Valcarenghi, L., Li, X., Papagianni, C., 5Growth Data-Driven AI-Based Scaling, in: *2021 EuCNC & 6G Summit*, 2021

2. Casetti, C.; Chiasserini, C. F.; Marcato, S.; **Puligheddu, C.**; Mangues-Bafalluy, J.; Baranda, J.; Brenes, J.; Bocchi, F.; Landi, G.; Bakhshi, B., ML-driven Provisioning and Management of Vertical Services in Automated Cellular Networks, in: *IEEE Transactions on Network and Service Management*, 2022

1.1.2 Distributed orchestration of network and compute resources for edge services

The second part of the work sees the design and application of resources management policies specifically for vRANs and edge user applications. In this case, instead of managing the entire network from a centralized standpoint, the focus of the contribution is the real-time management of edge resources and services to meet the selected KPIs for network and user services running at the edge, therefore closer to the user. The developed frameworks, named CAREM and VERA, are both based on reinforcement learning techniques. In this context, we have addressed the following main research questions.

RQ3: How to provide a context-aware radio resource management in heterogeneous virtual RANs?

Specifically, can ML techniques be used to operate multiple radio technologies at the same time? Which learning framework would be able to orchestrate heterogeneous radio links, and what kind of performance gains can be expected from it?

Our answer to this question is CAREM, a reinforcement learning framework designed to operate on heterogeneous virtual RANs. CAREM leverages the low latency of edge computing to control in real-time which radio link to use and adjust the transmission parameters over this link, according to the radio link quality and buffer state. We show that CAREM outperforms both the reference software-defined LTE implementation provided by srsRAN and a close competitor based on contextual bandit. This is going to be presented in detail in Chapter 4.

Our contributions can be found in:

1. Tripathi, S., **Puligheddu, C.**, Chiasserini, C. F., An RL Approach to Radio Resource Management in Heterogeneous Virtual RANs, in: *IEEE/IFIP WONS 2021*, 2021

2. Tripathi, S., **Puligheddu, C.**, Chiasserini, C. F., Mungari, F., A Context-aware Radio Resource Management in Heterogeneous Virtual RANs, in: *IEEE Transactions on Cognitive Communications and Networking*, 2022

RQ4: How to provide fair and scalable orchestration of network and compute Resources for edge services?

In particular, how can edge resources be allocated to edge user applications and network services co-located in the same edge node, considering that they may have entangled resource needs? Can ML techniques be used to learn the relationship and the consequent behavior of service operating parameters? Can further optimization be obtained when considering jointly the resource management of multiple services?

In a similar reinforcement learning approach as that of CAREM, we design VERA to operate over an LTE vRAN and a user application whose traffic is transmitted through the vRAN to the mobile user, and to provide scalable and fair resource allocation decisions. Indeed, in this case, it is clear that the resource requests of network functions (i.e., vRAN) and user applications are often entangled, thus a joint allocation decision is required to obtain the optimal performance. VERA is validated on a proof-of-concept testbed implementation, then its performance is evaluated numerically. Besides, we also compare its scaling cost to those of a centralized framework based on deep-Q networks. A detailed analysis and evaluation of the VERA framework can be found in Chapter 5.

Our contribution on this topic has been disseminated in:

1. Tripathi, S., **Puligheddu, C.**, Pramanik, S., Garcia-Saavedra, A., Chiasserini, C. F., VERA: Resource Orchestration for Virtualized Services at the Edge, in: *IEEE International Conference on Communications (IEEE ICC 2022)*, 2022

1.2 Outline

This thesis is organized as follows.

Chapter 2 presents a practical demonstration of an innovative management system for 5G networks, able to automatize vertical services provisioning on constrained network and compute resources while being aware of vertical Service Level Agreements (SLA). First, it presents an open and flexible 5G transport and computing platform developed within the European project 5G-TRANSFORMER, then it shows how, through service scaling at different layers, namely application-level, service-level, and resource-level, the platform is able to reconfigure the virtualized resource to guarantee SLAs. The 5G-TRANSFORMER platform capabilities are demonstrated with field tests, where two automotive vertical services are deployed on three geographically-distributed sites.

Chapter 3 focuses on the concept of Machine Learning as a service (MLaaS). Softwarized 5G network architectures really benefit from the adoption of Machine Learning techniques to drive decision-making in autonomous orchestration. To this end, first the 5Growth MANO stack, which evolved from the 5G-TRANFORMER platform and now integrates the MLaaS Platform (5Gr-MLaaSP), is presented; then it is demonstrated how the MLaaS can drive two ML-driven algorithms to perform network management tasks, namely network slice subnet sharing and run-time service scaling.

Chapter 4 proposes CAREM, a reinforcement learning framework designed to work in heterogeneous 5G Radio Access Networks. CAREM selects the best available radio link and transmission parameters to meet specified Key Performance Indicator (KPI) requirements according to the dynamic radio context. CAREM performance is evaluated through a testbed implementation that leverages LTE and IEEE 802.11p radio technologies, showing encouraging results, even compared to the closest existing scheme.

Chapter 5 introduces VERA, a reinforcement learning framework for resource orchestration of user applications and network functions at the edge. First, it is shown how the concurrent request of resources by user applications and network functions is often entangled, then the advanced features of VERA are described, highlighting the key role of the Pareto analysis in providing fair decisions for different applications. Finally, VERA's performance is assessed through numerical analysis, proving VERA's capabilities in meeting the target KPIs.

Chapter 6 summarizes the presented work, highlights the obtained results and offers conclusive remarks and considerations.

Chapter 2

Automated Service Provisioning and Hierarchical SLA Management in 5G Systems

Empowered by *network softwarization*, 5G systems have become the key enabler to foster the digital transformation of the vertical industries by expanding the scope of traditional mobile networks and enriching the network service offerings. To make this a reality, we propose an *automation* solution for vertical services provisioning and hierarchical Service Level Agreement (SLA) management. *Service scaling* is one of the most essential operations to adapt the service deployments and resource allocations to ensure SLA fulfillment. Three different scaling levels are addressed in this work: application-, service- and resource-level. We have implemented our solution in a proof-of-concept of a virtualized mobile network platform, spanning over three geographically-distributed sites. To evaluate our solution, we leverage field tests, focusing on *automotive vertical services* comprising a mission-critical application (collision-avoidance) and an entertainment one (video streaming). The results demonstrate the excellent performance of our solution, and its ability to automatically deploy vertical services and ensure their SLAs through different levels of service scaling.

Part of the work described in this chapter has been already published in X. Li et al., "Automated Service Provisioning and Hierarchical SLA Management in 5G Systems," in *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4669-4684, Dec. 2021, doi: 10.1109/TNSM.2021.3102890, ©2021 IEEE.

2.1 Introduction and motivation

5G systems are envisioned to expand the scope of traditional mobile networks to support various vertical services, such as eHealth, automotive, media, and cloud robotics, hence greatly enriching the telecom network ecosystem. In this new scenario, the imperative for telco service providers is to promptly support vertical industries to deploy their services over the 5G systems, fulfill their diverse requirements, and adjust the service deployments to the dynamic users and traffic demands. To this aim, network softwarization becomes a revolutionary technological shift, thanks to Software-Defined Networking (SDN) and Network Function Virtualization (NFV) enabling the dynamic creation of *Network Slices*, i.e., logically independent network partitions over a shared infrastructure. Network slices are provisioned as end-to-end network services composed of a set of interconnected Virtualized Network Functions (VNFs). Importantly, they are created by properly configuring virtual resources (network, compute, and storage), and tailoring them to address the specific requirements of the vertical services (e.g., bandwidth and end-to-end latency). Such a 5G system is beyond providing mobile radio access network (RAN) and core network functions as defined by 3GPP, but more about providing end-to-end network slicing solutions able to support heterogeneous network and RAN technologies for providing communications suitable for individual vertical services.

Despite the advanced development of network slicing solutions leveraging SDN/NFV, how to automatically deploy a vertical service on a reliable network slice in telco's operational networks is still a real challenge in practice. In particular, what is labored is for network slices to fulfill at any point in time the required service quality, as per the Service Level Agreement (SLA) established between the vertical and the telco provider. Indeed, SLAs specify a set of service business aspects and quality parameters that telco providers have to guarantee to verticals not to incur in penalties, e.g., required bandwidth, end-to-end latency between service endpoints, mean time to service recovery. Since vertical services are deployed and operated over network slices sharing a common infrastructure, some degradation or violation of the slice service parameters may occur that could impact the performance of the vertical service offered to final users, and, hence, could affect the reputation or business leadership of the vertical itself. For this reason, legal aspects are also regulated in SLAs between telco providers and verticals, identifying which party is responsible for reporting service failures or paying fees.

To meet the vertical SLAs, telco service providers need to map and translate high-level SLA business requirements into network slice- and infrastructure-related requirements, which can be actually handled and addressed at the network level. It follows that

other agreements at the network operational level have to be generated in cascade between the telco provider and other parties (e.g., network engineering departments, or cloud infrastructure providers). In an open NFV ecosystem, the SLA management is therefore a multi-dimensional provisioning and management problem, where multiple and interdependent aspects need to be addressed. This calls for a coordinated SLA framework accounting for different levels of performance inter-dependency and obligations, namely, at the application, service, and resource level.

Towards these challenges, the EU H2020 5G-PPP 5G-TRANSFORMER (5GT) project [4] has developed an open and flexible 5G transport and computing platform, able to automatically onboard and deploy vertical services. Importantly, this platform can also manage the service life-cycle and the SLAs, so as to fulfill diverse service requirements. It includes:

- a vertical portal to translate vertical service requirements into network slice-related requirements. This portal also maps vertical services onto network slices, realizing the latter through Network Services, as defined in NFV (NFV-NS);
- a service orchestration layer to manage the NFV-NSs and construct their logical networks. This is achieved by placing and connecting the service components in the virtual infrastructure, and by allocating the required virtual resources;
- an infrastructure layer that not only manages the underlying infrastructure resources but also handles the actual mapping of a logical network onto the shared physical network. It thus realizes the deployment of the vertical services into slices.

The above three layers may be owned and managed by different providers and entities in the real network scenarios, thus a hierarchical SLA management is essential to provide an automated and coordinated vertical service management throughout the whole stack of the system. As part of the SLA management, service scaling is one of the important operations to automatically adapt the service deployments according to (i) mutable needs of the vertical service and application components deployed on a slice (e.g., varying demand of service instances or of total resources required by the vertical services), (ii) the priorities of different vertical services running into the network slice instances, or (iii) real-time availability of (virtual) resources in the infrastructure underpinning the deployed network slices. Along this line, different levels of service scaling are provided at the different layers of the 5GT platform for such hierarchical SLA management.

In this chapter, we present the hierarchical SLA management framework that we have designed and developed on top of the 5GT platform (Sec. 2.2), and we introduce

the service scaling mechanisms that we have defined at the different levels, namely, application, service, and resource level (Sec. 2.3). We then describe the proof-of-concept test-bed where we implemented the scaling mechanisms (Sec. 2.4), which, importantly, have been released as open-source software[1]. Finally, we provide a thorough experimental evaluation in the relevant, practical case of automotive vertical services, as an example to demonstrate the ability of our framework to enable automatic service provisioning and ensure a successful SLA management (Sec. 2.5), and as well as a summary of related work (Sec. 2.6).

2.2 The 5G-TRANSFORMER Platform

The 5GT platform consists of three main building blocks [5], as shown in Fig. 2.1 and described in the following.

The **Vertical Slicer (5GT-VS)** is the aforementioned vertical portal and acts as one-stop shop entry point for the verticals to request a custom network slice, tailored to their needs. A vertical service is a composition of vertical applications as well as network functions, defined by its functional and behavioural specification, as detailed in the Vertical Service Blueprint (VSB). In particular, the vertical requests a service by selecting a VSB from the catalogue offered by the 5GT-VS and customizes it with additional details at the service-level (e.g., expected number of users, coverage area, required SLAs, etc.) thereby defining a Vertical Service Descriptor (VSD). In turn, the 5GT-VS, through its *Translator* module, maps these service requirements into a potential set including Network Service Descriptor (NSD), Deployment Flavour (DF), and Instantiation Level (IL). This triple defines the characteristics of the target network slice (deployed through an NFV-NS) in terms of:

- the functional elements and the structure of an NFV-NS underpinning the network slice able to host the requested vertical service. This is defined through the NSD and the related VNF descriptors;
- the number and capacity of the VNFs, and the virtual links needed to meet the performance requirements of the vertical service. Specifically, the DFs define the different options to instantiate the service, including a min-max range for the number of VNFs to be instantiated, while each IL indicates the specific number of VNF instances and their required computing resources.

[1] https://github.com/5g-transformer

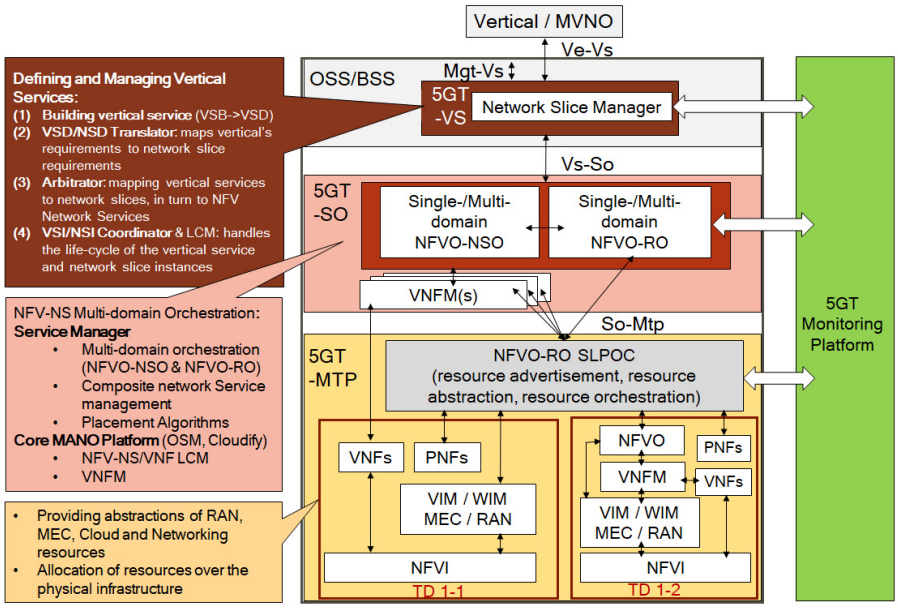


Fig. 2.1 The 5G-TRANSFORMER (5GT) system architecture

In the 5GT system, the definition of network slices is aligned with the latest networking slicing model from 3GPP [6] and has been extended to consider not only the mobile communication segments of the end-to-end service (as in the standard), but also the involved vertical applications. Furthermore, the 5GT network slice model is rather generic and can support any RAN and wireless technologies (although the latter aspects are beyond the scope of 3GPP). Network slices are deployed through NFV-NSs, which are instantiated according to a specific [NSD, DF, IL] set, selected on the basis of the service characteristics. The network slice components can be instantiated through a dedicated NFV-NS, to guarantee the maximum level of isolation, or exploit VNFs already instantiated for other services to optimize resource allocation. Some works in the literature analyze the challenges of RAN and core network slicing and resource sharing (e.g., [7] [8] [9]). In 5GT, the decisions on the sharing strategy applied to each network slice depend on the particular service and slice profile (such as coverage area, resource sharing/isolation policy, and performance requirements), and are determined through arbitration at the level of vertical services using those slices.

Specifically, according to the network slice model defined by 3GPP, a network slice can include multiple slice subnets, where each subnet can be shared among multiple end-to-end network slices, thus improving the infrastructure utilization efficiency. The number of network slices sharing a slice subnet and, consequently, the number of vertical services running over a slice subnet has an impact on its resource requirements (e.g., in

terms of traffic load to be supported). Also, any decision about re-using a slice subnet instance should be compliant with the isolation requirements of the services using it. As depicted in Fig. 2.1, the 5GT-VS includes an *Arbitrator* module that, starting from the candidate [NSD, DF, IL] set generated by the *Translator* and taking into account the network slices currently instantiated in the system, decides how to deploy the network slice for the requested vertical service. In particular, the 5GT-VS *Arbitrator* determines: *(i)* the [NSD, DF, IL] of the new network slice to be instantiated and the existing slice subnets that can be re-used to build the new slice, and *(ii)* for each slice subnet to be re-used, if and how it needs to be scaled to meet the requirements of the additional vertical service. The actions defined as output of the *Arbitrator* involve the instantiation and scaling of NFV-NSs, which are requested to the *Service Orchestrator*.

It is worth to point out that the 5GT-VS actions cover the entire lifecycle of a network slice, as defined by the 3GPP TR28.530 specification [10], from the preparation to the decommissioning phase. In particular, the definition and the on-boarding of VSBs and related NSDs correspond to the design and the on-boarding steps of the network slice preparation, respectively. The definition of the VSD and the instantiation of the service with the related network slice correspond to the creation step of the slice commissioning phase and its activation in the operation phase. Any action related to the scaling of network slice subnets and associated NFV-NSs can be mapped to the modification step during the slice operation, while the service termination actions include the de-activation and termination of the network slice, in the decommissioning phase.

The **Service Orchestrator (5GT-SO)** [11] provides both network service and resource orchestration capabilities in order to instantiate and manage network slices (deployed as NFV-NS instances) over shared resources, across single or multiple administrative domains [12]. This requires an interaction with (i) the 5GT-VS to receive NFV-NS service requests, (ii) the Monitoring Platform to configure metrics and respond to alerts, (iii) the Mobile Transport and Computing Platform to allocate resources, and (iv) other 5GT-SOs in case of multi-administrative domain service orchestration. As such, it is a central point for the coordination of all the architectural entities required to fulfill the SLA requirements of the requested service.

Therefore, the 5GT-SO implements the workflows for the (i) NFV-NS service lifecycle management (including on-boarding, instantiation, scaling, query, termination), (ii) intra-domain and multi-administrative domain orchestration, (iii) selection of VNF placement and inter-VNF links, and (iv) allocation of virtual networking, computing and storage resources through the 5GT-MTP based on service requirements and availability of the

resources offered by each administrative domain. The 5GT-SO also integrates core MAN and Orchestration (MANO) platforms, such as Open Source MAMO (OSM) or Cloudify, through wrappers, hence enabling the interworking between different core MANO platforms used by different administrative domains.

The **Mobile Transport and Computing Platform (5GT-MTP)** is responsible for managing the compute, storage, and networking resources (both physical and virtual) in the infrastructure where network slices and services from the above layers are eventually executed. The resources are generally spread in different technological domains (e.g., computing Point of Presence (PoP), Wide Area Network (WAN), RAN) and, hence, the 5GT-MTP provides a coordinated management and orchestration of all these resources toward the fulfilment of 5GT-SO requests. On the one hand, the 5GT-MTP aggregates the underlying resource pool in the infrastructure to be abstracted and exposed as a single coherent whole to the 5GT-SO at Single Logical Point of Contact. On the other hand, the 5GT-MTP translates the 5GT-SO requests from abstract to low-level resource requests to be allocated in each domain. For the 5GT-MTP to interwork with underlying resources, each technology domain exposes the API of its controller (e.g., Openstack, SDN controller, RAN controller) to the 5GT-MTP. The 5GT-MTP commands each controller through a corresponding plug-in acting as client of such API. This includes the transport WAN Infrastructure Manager (WIM), the Virtual Infrastructure Manager (VIM), the Multi-access Edge Computing (MEC), and the RAN plug-ins.

Transversal to the three aforementioned building blocks, the 5GT architecture includes a cross-layer **Monitoring Platform (5GT-MON)** that collects monitoring data from the 5GT-VS, 5GT-SO, and 5GT-MTP, and generates notifications (alerts) as input for SLA management decisions at the different layers. It is based on the Prometheus and the Grafana software, for the collection/storage/elaboration and the visualization of monitoring data, respectively. The 5GT-MON aggregates metrics and KPIs generated at the different layers, e.g., physical infrastructure and virtual resources load, the performance of network services, and metrics associated with vertical applications. Starting from the elaboration of these data, the 5GT-MON recognizes any performance degradation or anomalous conditions on the basis of thresholds defined in the descriptors (e.g., in the NSD) and notifies the 5GT components through asynchronous alerts. These notifications trigger the reaction of the 5GT platform (e.g., scaling or recovery actions) to guarantee the continuous fulfillment of the SLAs established at the different layers.

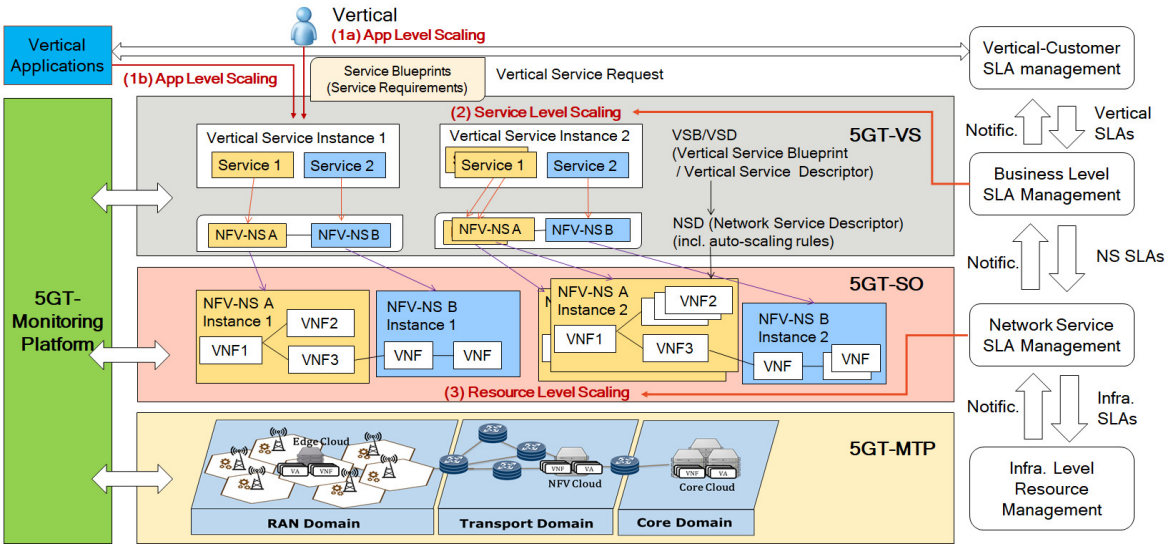


Fig. 2.2 Hierarchical SLA Management framework

2.3 Hierarchical SLA Management and Service Scaling: Concept & Implementation

To automatically manage vertical services through the 5GT system and fulfill the service requirements specified by the vertical, we propose a hierarchical SLA management framework, as illustrated in Fig. 2.2. From the top down, we define per-layer SLAs, along with the associated management mechanisms, as detailed below.

Vertical SLAs. They are business-level SLAs, which are negotiated between the 5GT telco provider's OSS/BSS (Operations Support System and Business Support System) and the vertical, and are managed by the 5GT-VS. On the one hand, the vertical provides the vertical service requirements in the service request, specifying business-level and service-level parameters (i.e., required service KPIs like maximum service provisioning time, required device density, and maximum service latency). On the other hand, the telco provider's OSS/BSS can offer different business service level classes. The matching between the vertical service requirements and the service level classes offered by the 5GT telco provider defines a Vertical SLA. In the 5GT-VS, the Vertical SLA management functions include: (i) mapping the Vertical SLAs to Network Service SLAs (NS SLAs) that will be requested to the 5GT-SO, including network service related policies such as rules for the automatic scaling of a NFV-NS instance; (ii) mapping a vertical service on network slice(s), either instantiating a new network slice or re-using existing slice subnet(s); and (iii) handling service arbitration and service scaling actions to deal with

the dynamic changes on the service itself and according to the service Vertical SLAs, the total available resource budget, and the services priority.

NS SLAs. They are defined at the NFV-NS level and are managed by the 5GT-SO. NS SLAs define distinct guarantees on resource availability and KPIs, such as guaranteed data rate, geographical availability, and end-to-end latency. Their management is provided by end-to-end service and resource orchestration, including (i) deciding the optimum placement of the VNFs in certain PoPs/servers and the inter-PoP/inter-server connectivity, and (ii) handling the NFV-NS auto-scaling operation to adapt to the dynamic network conditions following the aforementioned NFV-NS auto-scaling rules defined in the NSD.

Infrastructure SLAs. They are managed by the 5GT-MTP, which is in charge of the actual resource allocation for a specific service, as requested by the 5GT-SO. The Infrastructure SLAs are specified based on the NS SLAs and define different guarantees on infrastructure-level QoS (such as CPU load, network delay, packet losses, link throughput). At the infrastructure level, the resource management function is in charge of the placement of virtual machines (VM) (or containers) in physical servers, and managing their required networking connectivity such as routing and path provisioning.

In summary, each layer is fully responsible for: (i) translating its SLAs to lower-level SLAs and requesting them to the lower layer, and (ii) guaranteeing the corresponding SLAs through its internal SLA management functions. To this end, each layer interacts also with the monitoring platform, which provides monitoring data about SLA-related metrics and triggers alerts whenever a degradation or violation of the SLAs is detected. Finally, each layer may notify the higher layer about the results of the SLAs it is managing.

As highlighted above, one of the crucial SLA management functions is ***scaling***. Depending on the layer at which it is performed, we can define:

1. *Application-level scaling*, which is triggered by the vertical and implies the scaling of a vertical service deployment based on the operational context information [13]. This typically results in a renovated slice service demand to the telco service provider (i.e., the to 5GT-VS), with different service parameters;

2. *Service-level scaling*, which is triggered by the 5GT-VS as a result of an arbitration procedure among service instances and yields the scaling of one or more of them;

3. *Resource-level scaling*, which is triggered by the 5GT-SO after detecting lack of sufficient resources to meet certain NS SLAs. It leads to the scaling of the virtual resources underpinning the network slice deployment.

2.3.1 Application-level scaling

Application-level scaling consists in adjusting vertical service deployments into the network slices during their runtime, according to evolving vertical's business targets (e.g., enlarging the geographical area that is served) or following the dynamics of the application operational context (e.g., average and peak number of user requests). In both cases, the scaling decision results in a renovated slice service request to the 5GT-VS but with different service parameters. Examples of such service parameters could be the number of mobile users or content items for a Content Delivery Network in the case of a multimedia service. For an automotive safety service, a relevant parameter could be the maximum number of cars to be served in a given area. To support the application-level scaling, the 5GT-VS provides a vertical north bound interface (NBI) that allows the verticals to issue slice service requests with revised service parameters specified in the VSD.

The verticals and their applications have thus a fundamental role in the application-based scaling model. They are not only responsible for detecting the need to scale a service and deciding its target size (expressed in terms of service-level parameters within the target VSD), but also for triggering the entire scaling procedure by interacting with the 5GT-VS. In particular, the need for a new VSD with updated service parameters can be stated by the vertical service administrator through a manual configuration (1a in Fig. 2.2), or can be detected automatically by a service control logic internal to the application (1b in Fig. 2.2). In the former case, the system administrator uses the 5GT-VS web GUI to manually request the modification. In the latter case, the application itself interacts directly with the NBI of the 5GT-VS, through the REST APIs. The mechanisms for making decisions about application-based scaling are service-dependent and they rely on business considerations or application-level performance metrics. The applications implement their own monitoring procedures to gather and elaborate the required application-level metrics. Their internal logic makes decisions about the required scaling actions, e.g., based on the thresholds defined in compliance with the SLAs established between the verticals and their customers.

At the 5GT-VS level, the enforcement of the vertical service scaling is managed in two phases. In the first phase, the 5GT-VS uses the *Translator* module to map the new VSD into the definition of a network slice able to meet its requirements. This mapping follows the same procedure performed during the instantiation phase and it identifies the characteristics of the target network slice. In particular, the output identifies the kind, size, and capacity of NFV-NS(s) corresponding to the end-to-end network slice, including its slice subnets. If the target network slice differs from the current one (e.g., in terms of DF and/or IL of the correspondent NFV-NS(s)), the current network slice(s) must be updated (i.e., scaled in or out); thus, the 5GT-VS starts a second phase that involves the *Arbitrator* module. Therein the system verifies the compatibility between the new network slice with its new target size/capacity and the SLAs established with the vertical, taking into account the whole set of network slices already active for the given vertical. In this phase, the arbitration algorithm (see Sec. 2.3.2) computes the modifications required for all of the different services owned by the vertical, based on their relative priorities. As output, the *Arbitrator* decides the feasibility of the scaling action and its impact on the existing network slice and slice subnet instances. It thus identifies the NFV-NSs of those services with lower priority and belonging to the same vertical that may need to be scaled down to make some resources available to services with higher priority. The overall resulting set of changes are then applied to the NFV-NSs (realizing the network slices) scaling actions and then requested in an ordered manner to the 5GT-SO, which will modify the NFV-NSs as requested.

2.3.2 Service-level scaling

Service-level scaling consists in adjusting the size (i.e., number of VNF instances) and/or the capacity (i.e., total resource demand) of network slice instances hosting the vertical services, as a result of a decision made by the 5GT-VS Arbitrator. As mentioned before, the *Arbitrator* is the entity responsible for making any decision about network slice sharing and scaling. In particular, upon receiving a vertical's request to deploy a new service instance, the *Arbitrator* looks up the corresponding Vertical SLA, namely: (1) the priority level of the new, as well as the existing, service instances requested by that vertical, (2) the set of VNFs composing the service and how they are inter-connected, (3) the relative virtual CPU (vCPU) and memory/storage requirements of the involved VNFs as well as the networking requirements for their inter-connectivity, and (4) the vertical's KPI requirements (e.g., end-to-end latency, service availability, or reliability

level). Note that such information is described in the NSD created by the *Translator* for the received VSD.

Given such an input information, the *Arbitrator* makes the following decisions:

1. it determines whether the newly requested service must be created from scratch, thus instantiating all its elements ex-novo, or, instead, one or more already existing slice subnets can be reused;

2. if the service has to be deployed entirely (or partially) ex-novo, it determines (a) the amount of resources that may be needed for the service (or part of it) to be instantiated and to handle the expected traffic load, and (b) whether or not such an amount is compatible with the resource budget available to the vertical, as per the Vertical SLA;

3. when existing slice subnets can be re-used, it decides which (if any) scaling action for any of them is necessary to fit the service requirements, and also feasible as per the Vertical SLA.

Thus, for every new or to-be-reused slice subnet, the *Arbitrator* provides as output the associated pair [DF, IL], so as to ensure that the vertical KPI requirements are met, while accounting for the services priority level and the remaining resource budget available to the vertical, as per the Vertical SLA.

Let us first consider that no existing sub-slices can be reused for the deployment of a newly requested service, and let us denote with C, B, and S the total amount of, respectively, vCPU, bandwidth, and storage that can be allocated for the services of that vertical as per the Vertical SLA. As the first step, the *Arbitrator* orders the list of all service instances $\mathcal{S}$, both the one to be deployed and the existing ones, from the highest priority level to the lowest. It then considers the highest-priority service instance, say, s, and allocates storage resources based on the needs exhibited by the VNFs composing s.

A more complicated procedure, however, is required for the vCPU and bandwidth allocation. In particular, let us focus on the service latency as the main performance metric, and denote with D_s the target latency for service s. Two factors contribute to the service latency: (i) the processing time, due to the execution of the VNFs composing the service, and (ii) the network time, i.e., the time it takes to transfer data from a VNF to the next one[14]. While the former depends on the vCPU allocated to the VM or containers running the VNFs, the latter depends on the deployment decisions made by

the 5GT-SO, and on the bandwidth associated with the virtual links connecting the servers hosting the VNFs.

In the *best* case, the whole set of VNFs composing service s, denoted by $\mathcal{V}$, can be deployed within the same server. In this scenario, the network time is negligible [15], hence the bandwidth required for data transfers over virtual links for s, β^b, can be set to zero. Additionally, the latency budget, D_s, can be entirely used as processing time, thus reducing the required amount of vCPU, μ^b. To determine such value, we follow a widely adopted approach (see, e.g., [16, 17]) and model each VNF instance as an M/M/1-PS queue. Note that the choice of the processor sharing (PS) policy for the queue model closely emulates the behavior of a multi-threaded application running on a VM. Then μ^b can be computed so as to satisfy the below inequalities:

$$\sum_{v \in \mathcal{V}} \frac{1}{f_v \mu^b - \lambda_v} \leq D_s; \qquad \mu^b \leq C\,. \tag{2.1}$$

The first inequality imposes that the total latency due to the processing at the service VNFs does not exceed the maximum target value. In particular, the left hand side term represents the total latency due to the processing at every VNF $v \in \mathcal{V}$ [18], with $f_v \mu^b$ being the output rate of the VNF queue v and λ_v being the service request rate input to v. Also, f_v is the relative computational requirement of VNF v, with $\sum_{v \in \mathcal{V}} f_v = 1$. The second inequality, instead, imposes that the vCPU allocation does not exceed the vertical vCPU budget, C.

Assuming that all VNFs run within the same server, however, might be overly restrictive. Thus, the *Arbitrator* also considers a *worst-case* scenario, accounting for the network latency component as well. As a smaller portion of the latency budget would be available for processing, the amount of processing resources required in this case increases. Specifically, in the worst case, each VNF in $\mathcal{V}$ is deployed in a different server, hence the allocated vCPU, μ^w, and bandwidth, β^w, have to satisfy the following constraints:

$$\sum_{v \in \mathcal{V}} \frac{1}{f_v \mu^w - \lambda_v} + \sum_{(u,v) \in \mathcal{E}} \frac{d_{u,v}}{f_{u,v} \beta^w} \leq D_s \tag{2.2}$$

$$\mu^w \leq C \quad \text{and} \quad \beta^w \leq B \tag{2.3}$$

where $f_{u,v}$ is the relative bandwidth requirement for the virtual link connecting the servers where VNFs u and v are deployed, and $d_{v,v}$ is the amount of data that needs to be transferred from VNF u to VNF v. In (2.2), the two left hand side terms represent the latency due to, respectively, the VNF execution and the travel time over the virtual

links connecting any two adjacent functions in the VNF set ($\mathcal{E}$ denotes the set of edges interconnecting the VNFs composing the service). The constraints in (2.3), instead, impose that the total vCPU and bandwidth allocations do not exceed the corresponding budgets available to the vertical, as per the Vertical SLA.

Next, given the pairs $(\mu^b, 0)$ and (μ^w, β^w) for service s, the *Arbitrator* can compute the corresponding per-VNF and per-virtual link values, by leveraging the f_v and $f_{u,v}$ values expressing the relative computation and bandwidth requirements of each VNF and virtual link (resp.). The *Arbitrator* then selects an ordered list of [DF, IL] pairs, as encoded in the NSD, with the first pair corresponding to the best-case allocation and the last one to the worst-case allocation; a practical example is provided in Sec. 2.4.

Once the 5GT-SO receives from the 5GT-VS the instantiation request, it deploys the service selecting the most efficient [DF, IL] pair among the viable ones suggested by the 5GT-VS. The quota of resources used by the vertical is updated at the 5GT-VS, based on the 5GT-SO's choice. Given such a value, the *Arbitrator* proceeds with the second service in the list, following the same steps as above but replacing C and B (in (2.1)–(2.3)) with the amounts of vCPU and bandwidth still available to the vertical. The procedure is repeated for all (newly requested or already deployed) service instances; it is clear that, in case of resource shortage, some lower-priority service instances may not be accommodated, or may be terminated due to the need to reallocate resources to higher-priority services.

As a relevant case in terms of scaling, let us consider now that a service instance requested by the vertical can be deployed by reusing one or more of the existing slice subnets. As mentioned, sharing a slice subnet across multiple end-to-end slices may impact its performance and, consequently, the performance of the vertical services using it. Thus, to guarantee the required performance, the size and/or the capacity of a slice subnet instance may need to be adjusted according to the number and characteristics of the end-to-end slice instances that are sharing it. In this case, the *Arbitrator* adds the traffic load due to the newly requested vertical service to the load of the existing VNFs (λ_v) and virtual links ($d_{u,v}$), and re-computes the necessary vCPU and bandwidth allocation, as described above. Again, the *Arbitrator* uses the vCPU and bandwidth values obtained in the best and worst cases, to update the [DF, IL] pairs associated with the involved VNFs and virtual links.

Finally, we remark that the same procedure is performed when a service instance is terminated: the 5GT-VS updates the amount of resources available to the vertical

and recomputes the [DF, IL] pairs for the remaining services, upgrading some of them if needed.

2.3.3 Resource-level scaling

Resource-level scaling involves monitoring and reconfiguration of virtual resources orchestrated by the 5GT-SO (in coordination with the 5GT-MTP), to prevent the performance degradation of VNFs/NFV-NSs. More specifically, it regards the auto-scaling of NFV-NSs according to the scaling rules given in the NSD, by configuring related monitoring jobs and alerts in the monitoring platform, and by properly reacting to such alerts. Scaling rules are defined by the vertical, based on business-related considerations or the application-related operational context, and are part of the VSB definition when on-boarded in the system. These scaling rules are encoded in the NSD and then forwarded from the 5GT-VS to the 5GT-SO.

Each auto-scaling rule contains (a) the conditions to be met by certain metrics, to trigger alerts based on the service monitored data, and (b) a corresponding reaction (i.e., a scaling out/in action). During the NFV-NS instantiation phase, the 5GT-SO configures the 5GT-MON monitoring platform according to the conditions encoded in the NSD auto-scaling rules, in order to receive the required alerts at the NFV-NS runtime. Whenever the 5GT-SO is notified by the 5GT-MON that one of the conditions is met (e.g., exceeded vCPU usage), it triggers the NFV-NS scaling according to the corresponding reaction specified in the auto-scaling rule. To this end, it also coordinates the operation of the core MANO and 5GT-MTP. In particular, in case of scaling out, the 5GT-SO issues a new resource allocation request to the 5GT-MTP for scaling the VNF instances and, hence, to reconfigure the virtual resources towards the new instantiation level specified in the auto-scaling rules. The 5GT-MTP applies all needed settings for the required resource re-allocations, while the 5GT-SO notifies the 5GT-VS about the scaling operation outcome. In case of scaling-out failure, due to, e.g., resource shortage, the 5GT-SO undoes or rolls-back the scaling operation and also informs the 5GT-VS about the failure.

In terms of implementation, the 5GT-SO provides resource-based scaling thanks to two internal submodules, namely the *Monitoring Manager* and the *SLA Manager* [11]. The former configures the monitoring jobs required to measure the resource metrics involved in scaling decisions. The latter requests the configuration of the alerts associated with these metrics in the 5GT-MON and also processes the received alerts to trigger scaling actions

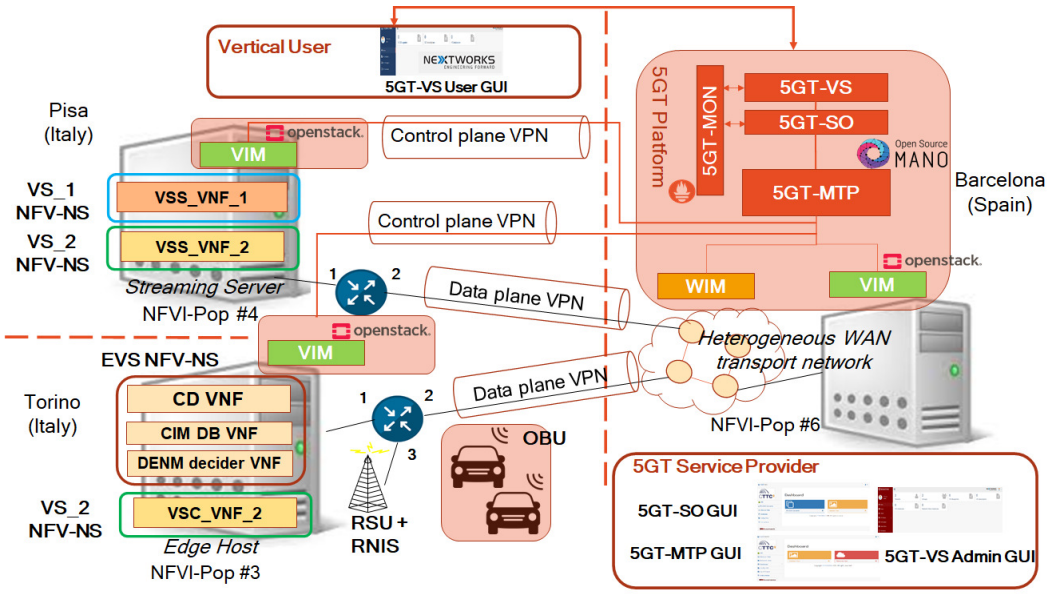


Fig. 2.3 Testbed architecture and a possible deployment of the considered NFV-NS across the available PoPs

according to the auto-scaling rules. The 5GT-MON configuration is then coordinated through a *Configuration Manager* component, which offers REST APIs and wraps the logic of the configuration for the different Prometheus components involved in the monitoring task. Specifically, 5G-MON leverages the following Prometheus components: (i) Monitoring jobs, used to retrieve monitoring data from different sources through the mediation of Prometheus exporters specialized for infrastructure or application metrics, (ii) Thresholds, used to trigger alerts towards the SLA manager, and (iii) Dashboards, used to visualize the monitoring data through Grafana.

2.4 Proof-of-concept: Testbed and Scenarios

In this section, we present the proof-of-concept testbed we deployed to evaluate our framework. The testbed implements the whole 5GT platform introduced in Sec. 2.2, and the applications required by an automotive vertical to be provided to mobile users and vehicles. It is worth noting that here we focus on the scaling of NFV-based vertical services rather than on the network functions related to the mobile infrastructure. More importantly, our solution is generic to support any network functions (including RAN or core network functions) and vertical applications, and hence, as also underlined below, our solution can work well with any radio access technology.

Below, we start by introducing the testbed architecture and the vertical targeted services (Sec. 2.4.1), then we describe the scenario and the experiments that we performed in our field tests (Sec. 2.4.2). The performed field tests aim to demonstrate the effectiveness of the developed 5GT platform and our proposed solutions, able to: (i) automatically deploy vertical services upon receiving the service requests, and (ii) perform automated management of the vertical services across different layers of the architecture. In particular, SLA assurance is achieved through service-level scaling to handle the arbitration among different services of the same vertical according to their priorities, and by resource-level scaling to handle the scaling of resources (i.e., in terms of the number of VNF instances), according to resource dynamics and load variations.

2.4.1 Testbed and supported services

The testbed, depicted in Fig. 2.3, spans over three geographical sites, which are connected through VPN tunnels encapsulating both control traffic and data traffic. The Barcelona site, in Spain, hosts all layers of the 5GT platform, including the 5GT-VS, 5GT-SO, 5GT-MTP and 5GT-MON, an instance of an Openstack-based VIM, and an instance of a WIM controlling the core transport network. The 5GT platform uses OSM Release 6 as MANO platform. The WIM, as described in [19], follows an IETF Application-Based Network Operation (ABNO) architecture using the Control Orchestration Protocol (COP) to communicate with the 5GT-MTP and interacts with the forwarding elements of the underlying transport networks by means of open source SDN controllers like Ryu and OpenDaylight. Then, two instances of an Openstack-based VIM are deployed at, respectively, the Pisa and the Torino site, in Italy, with the latter acting as MEC host. The Torino site also hosts an IEEE 802.11p Roadside Unit (RSU) physical network function to provide radio access to vehicles equipped with On-Board Units (OBU), and a Radio Network Information Service (RNIS) providing channel state information to the applications requiring it. An equivalent testbed was deployed in Torino [20], using the open-source Open Air Interface (OAI) [2] implementation of the LTE E-UTRAN and EPC, which is compliant with 3GPP LTE Releases 8/10. Laboratory tests [21] showed that the proposed system architecture and scaling solution work effectively also under these settings. However, during field tests an IEEE 802.11p-based radio access was preferred, so as to deal with vehicular communications and ensure a sufficiently large outdoor coverage.

[2] https://openairinterface.org

The testbed supports two services requested by an automotive vertical, namely, (i) vehicle collision avoidance at intersections, and (ii) video content delivery, which are relevant examples of, respectively, safety and entertainment services for vehicular users.

The vehicle collision avoidance service will be referred to as Extended Virtual Sensing (EVS), since it leverages vehicular communications as a virtual sensor collecting data related to vehicle mobility [22, 23, 20]. Specifically, it exploits the Cooperative Awareness Messages (CAMs), defined by ETSI, which are periodically transmitted by vehicles and carry the position, speed, acceleration, and heading of the sender. By processing such data, the EVS service can detect dangerous situations and generate warnings accordingly. These warnings are encoded in the ETSI Decentralized Environmental Notification Messages (DENMs) and delivered to human drivers, or to an emergency braking system aboard vehicles. The VNFs composing the EVS service are as follows:

- the CIM (Cooperative Infrastructure Manager), which receives, decodes, and stores CAMs sent by the vehicles within the area covered by the EVS service;
- the Collision Detector (CD), which queries the CIMs for new CAMs and runs a trajectory-based algorithm (e.g., the one presented in [22, 23]), to detect pairs of vehicles on collision course;
- the DENM Decider, which timely encodes the warning messages and sends them to the vehicles deemed to be on collision course.

The video content delivery service refers to a Video Streaming (VS) service that may be provided in *full-fledged* or *reduced* configuration. The *full-fledged* version consists of the following two VNFs:

- the Video Streaming controller (VSC), which exploits a Radio Network Information Service (RNIS) and a radio link manager, recording information on the quality of the user radio channel. This information is given as input to an optimization algorithm [24] that selects the most suitable bit rate for streaming the video to the user;
- the Video Streaming Server (VSS), featuring a Python-based front-end, which applies the selected video bit rate to the video segments to be transmitted. It is based on HTTP streaming and contains a video catalogue, a front-end, the Media Presentation Description (MPD) files, and the media chunks. The front-end receives the selected video bit rate and edits the MPD files with such a rate.

The *reduced* VS service differs from the *full-fledged* version in the fact that it includes the VSS only, i.e., it is unable to adapt video encoding to the user channel conditions. Finally, we remark that both the EVS and the VS services require a mobile transport function that realizes the communication between the network infrastructure and the vehicular users.

2.4.2 Evaluation setup and experiments

The evaluation setup consists of (i) a vertical service deployment in the form of NFV-NSs, and (ii) a vertical service operation once the NFV-NSs are deployed.

As for the vertical service deployment, upon receiving the automotive vertical request for services, the corresponding VSD is compiled at the 5GT-VS. Importantly, at this stage, the vertical specifies (i) the services' priority (with EVS having higher priority than VS), (ii) the services' configuration, (iii) the storage requirements for the VS and EVS VNFs, (iv) the geographical area that has to be covered by each service, and (v) the estimated number of users to serve. Also, since the EVS should be combined with other collision avoidance mechanisms based on physical sensors aboard the vehicles, the maximum target latency specified by the vertical in the VSD is set to 20 ms. The VSD is then translated into the NFV NSD, and the [DF, IL] pairs are set for each VNF instance according to the output of the arbitration algorithm running at the 5GT-VS. In particular, for all VNFs composing the VS and the EVS service, except for the CD, only one instance using up to 1 vCPU is foreseen as both minimum and maximum allocation, since the processing latency of such VNFs is limited and does not significantly increase with the traffic load. For the EVS CD, instead, the [DF, IL] indicates the possibility to have from 1 up to 2 instances, each using 1 vCPU. The 5GT-SO, which is aware of the computing and network resources available as exposed by the 5GT-MTP (which, in turn, interacts with underlying VIMs and WIMs), computes the most appropriate placement for the VNFs, and instantiates them as Openstack VMs, following the NSD requirements. Service monitoring is performed by the 5GT-MON, which pulls and stores metrics from the VMs hosting the aforementioned VNFs.

With regard to the vertical service operation, the test field used for our experimental evaluation consists (unless otherwise specified) of a urban intersection with two vehicles, one of which is an automated car equipped with an Automatic Emergency Braking (AEB) system. Both vehicles are equipped with an IEEE 802.11p OBU, thus they transmit CAMs every 100 ms and can receive DENMs. The vehicle equipped with the AEB can

Fig. 2.4 Map of the geographical area served by the EVS service, featuring two intersections and including an IEEE802.11p RSU: one EVS instance and corresponding covered intersections highlighted in yellow (left), two EVS instances and corresponding covered intersections highlighted in pink (right)

also process DENMs and has the necessary on-board logic to translate the DENM content into a command for the AEB. The vehicles travel on perpendicular roads and approach the intersection at full urban speed (namely, 50 km/h).

The field trial includes three phases, as detailed below.

- **Phase 1:** The vertical asks for the deployment of two VS instances, one *full-fledged* and the other in *reduced* configuration. This first part of the trial shows how an automotive vertical can use the 5GT platform to instantiate two different VS services, just by providing high-level service parameters and without any detailed knowledge of the underlying infrastructure.
- **Phase 2:** The vertical asks for the deployment of an EVS instance, which, being a safety service, has higher priority than VS. Due to limited resource budget available as per the vertical SLA, the instantiation of the EVS requires that service priority is properly handled by the *Arbitrator* at the 5G-VS (service-level scaling).
- **Phase 3:** The vehicle density on the area served by the EVS increases, which impacts significantly on the computing load and in turn the application latency. Thus, whenever load changes, scaling at the resource level is needed to keep up with the SLA latency requirements (resource-level scaling).

To emulate a high vehicle density, in Phase 3 we consider a urban section of the city of Torino, depicted in Fig. 2.4(left), including two urban intersections, and we leverage a mobility trace obtained with the SUMO simulator [25]. This trace is processed so as to generate the CAMs corresponding to the simulated vehicles; such CAMs are then injected through the same data plane connections used for real cars in the field trial.

This allows us to handle such CAMs in exactly the same way as those generated by real vehicles, i.e., they are transmitted on air and, upon being received, the information they carry is stored in the CIM.

2.5 Field Trial Performance Results

In this section, we report the actions taken by the 5GT platform during the Phase 1 to 3 of the field trial, as well as the performance of the services that are deployed. We first report the decisions made by the 5GT-VS upon receiving the VS and EVS set-up requests, and the decisions made by the 5GT-SO as the vehicle density increases (Sec. 2.5.1). Then we present some results obtained by profiling the service creation time components, for both VS and EVS (Sec. 2.5.2). Finally, we show results related to service-level and resource-level scaling, which take place during, respectively, Phase 2 (Sec. 2.5.3) and Phase 3 (Sec. 2.5.4) of the trial.

2.5.1 Services performance – All phases

In Phase 1, upon receiving the request of the two VS services, the 5GT-VS goes through all the steps described in the previous sections and ultimately generates two NFV-NSs. One NFV-NS is for the *full-fledged* VS service with an IL corresponding to a large amount of allocated resources and enforcing that the VSC is placed in the MEC. The other NFV-NS is for the *reduced* VS version, with an IL for low resource footprint. We recall that the VSC must be deployed in the MEC, since it includes a radio manager requiring the RNIS (MEC) service for the tracking of the user channel quality. The 5GT-SO then deploys the VSS instances of the two VS NFV-NSs as VMs in the Pisa site, and one VSC instance in the Torino (MEC) site, and sets up the links to interconnect the VSC with its associated VSS in the Pisa site through the Barcelona transport infrastructure by interacting with the 5GT-MTP.

Fig. 2.5 (Phase 1: from 0 to 273 s) shows the bit rate of the two VSs (i.e., with and without VSC). The *full-fledged* VS (purple line) increases the video segment bit rate when the quality of the radio channel (reflected by the Channel Quality Indicator (CQI), blue line) is high, while the *reduced* VS (green line) maintains the same video segment bit rate.

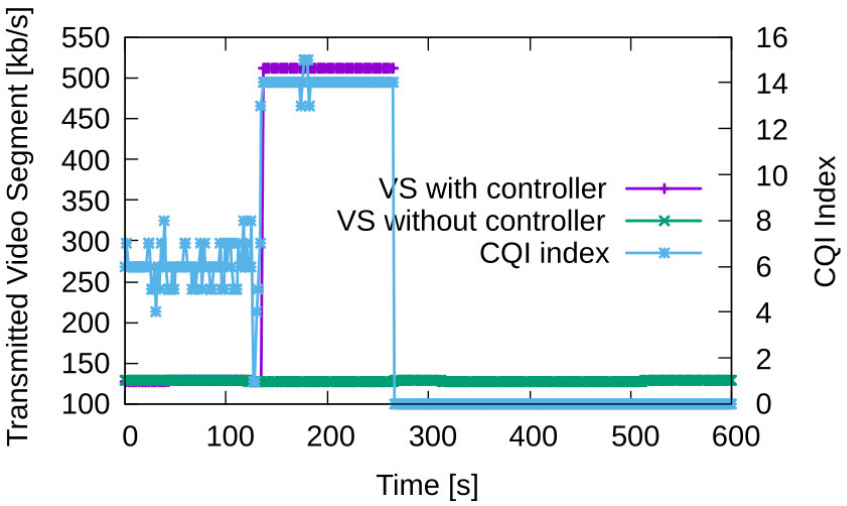


Fig. 2.5 Video service segment bit rate and received CQI before (0–273 s) and after arbitration (273–585 s)

In Phase 2, the 5GT-VS receives the request for an instance of EVS, characterized by a target maximum end-to-end latency of 20 ms, which implies that the corresponding NFV-NS constraints the service deployment in the MEC (Torino site). However, following the *Arbitrator* algorithm in Sec. 2.3.2, the 5GT-VS detects that the amount of resources necessary to deploy the EVS exceeds the total resource budget specified in the vertical SLA, and that the resources previously allocated for the VS services must be revised. In particular, the algorithm indicates that the *full-fledged* VS service must be terminated so that the resources allocated to the VM implementing the VSC VNF in the MEC host are made available to the EVS service.

Fig. 2.5 (Phase 2: from 273 to 585 s) reports the bit rate of the video segment after arbitration, when only the *reduced* video service (the one without VSC) remains in place. Thanks to the availability of real cars, this part of the trial not only demonstrates the correct behavior of the *Arbitrator*, but also that the EVS is successfully deployed and meets the automotive safety requirements and the maximum target latency of 20 ms. In particular, the value of end-to-end latency, from the transmission of the CAM to the reception of the corresponding DENM, averaged over 10 different tests, is equal to 8.870 ms, with a standard deviation of 1.447 ms, and a maximum and a minimum value equal to 11.637 ms and 5.050 ms, respectively.

In Phase 3, the *reduced* VS service and the EVS service run simultaneously, one in the Pisa site and the other in the Torino (MEC) site. With regard to the EVS, as the vehicle density in the area highlighted in yellow in Fig. 2.4(left) increases, the CAM

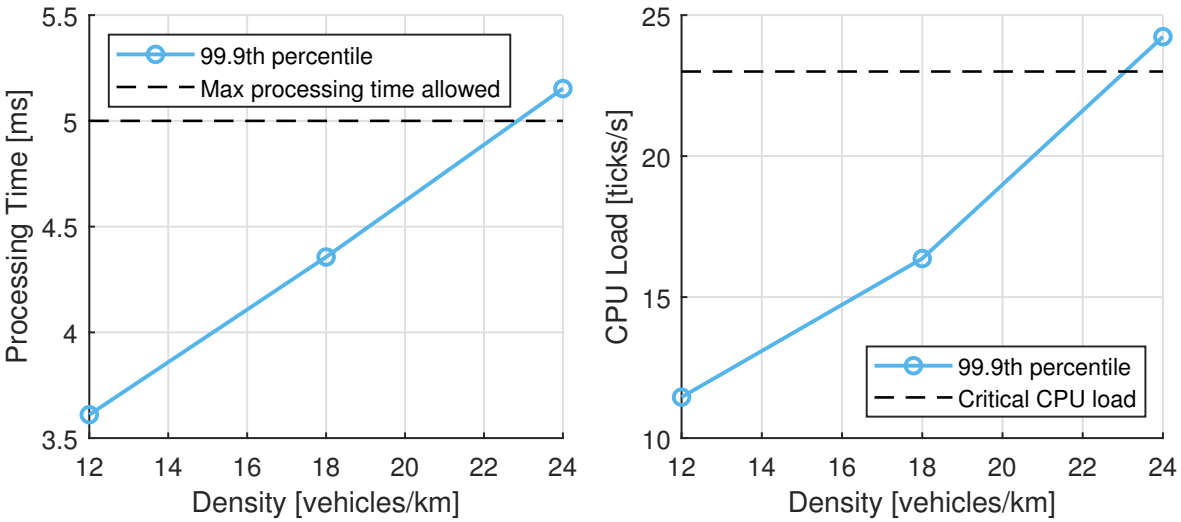


Fig. 2.6 VNF implementing the CD algorithm: processing time (left) and CPU load (right), as functions of the vehicular density

and processing load grows as well. Such an increased load of the VMs implementing the VNFs yields an increased processing time, hence an increased end-to-end latency for the EVS. In particular, we observed that the major contribution to the processing time is due to the VM implementing the CD algorithm, while the contribution of the other VNFs is negligible (e.g., 10 times smaller) and does not significantly scale up as the vehicle density grows. As a consequence, we focused on the CD VNF and measured its processing time (Fig. 2.6(left)) and the corresponding CPU load (Fig. 2.6(right)), for different values of vehicular density.

Given the latency contributions of the other VNFs and the latency of the data radio transfer, we computed a threshold for the processing time of the CD algorithm (namely, 5 ms) that cannot be exceeded. Then, leveraging the results in Fig. 2.6(left), we identified the critical vehicle density (e.g., 22.5 vehicles/km in the plot) below which a CD processing time of less than 5 ms is recorded for the 99.9% of the time. Finally, we used this density value in Fig. 2.6(right) and derived the critical CPU load (namely, 23%) as the threshold for resource scaling.

Once we determined the critical CPU load threshold, we configured the monitoring jobs and alerts in the 5GT-MON platform to generate an alert whenever the CPU consumption of the CD VM reaches such a value. The alert is handled by the *SLA Manager* module of the 5GT-SO, which generates a scale-out request, i.e., the deployment of a second CD instance, and makes the EVS service self-reconfigure to split the load between the two CD VNFs. In this way, half of the cars in the area covered by the service

Table 2.1 Description of main service creation time components

Service configuration	Composition	Observation
reduced VS	1 VSS VNF	Low resource footprint IL
full-fledged VS	1 VSS VNF, 1 VSC VNF	High resource footprint IL
EVS (IL with 1 CD)	1 CIM VNF, 1 DENM VNF, 1 CD VNF	Initial IL
EVS (IL with 2 CD)	1 CIM VNF, 1 DENM VNF, 2 CD VNFs	IL for high density scenarios

can be handled by the initial CD VNF and the rest by the new CD VNF. Specifically, with reference to Fig. 2.4(right), each CD instance processes only the CAMs generated by the vehicles crossing one of the intersections highlighted in pink. As a consequence, the CPU load is halved, and the target latency required by the EVS service can be fulfilled. An initial functional prototype was demonstrated in [21]. Importantly, the above discussion applies to scale-in operations as well. Hence, when the density decreases and some resources can be freed, a scale-in operation is performed exploiting the same mechanism as described above.

2.5.2 Service creation – Phase 1

We analyzed Phase 1 from the viewpoint of the 5GT platform. The focus is on profiling of the service creation and instantiation process for the requested VS and the EVS services, considering the different ILs available for each considered NFV-NS (see Table 2.1). We remark that, in all plots presented here and in the following sections, boxplots represent the experienced maximum, minimum, average, median, 20th- and 80th-percentile of the ten repetitions performed for each experiment.

Fig. 2.7 shows the various components of the service creation time, when deploying the EVS service consisting of 2 CD VMs The phase taking longer is the Allocate VNFs one, whose duration is roughly 6 times higher than the longest of the remaining components. This phase accounts for the time it takes to the OSM wrapper to interact with OSM, which, in turn, deploys the VMs that implement the EVS service by interacting with the 5GT-MTP and Openstack. In this case, there are four VMs to deploy, i.e., 1 CIM VM, 1 DENM Decider VM, and 2 CD VMs. The following components in order of importance are the creation of the intra-PoP networks (6.986 s on average) and the 5GT-VS processing (6.385 s on average). The former accounts for the time it takes to the OSM wrapper to interact with the MTP, which, in turn, interacts with Openstack to create all the required intra-PoP networks of the 2 CD-EVS service. The latter one has a much larger dispersion due to the polling that the 5GT-VS does to the 5GT-SO to know

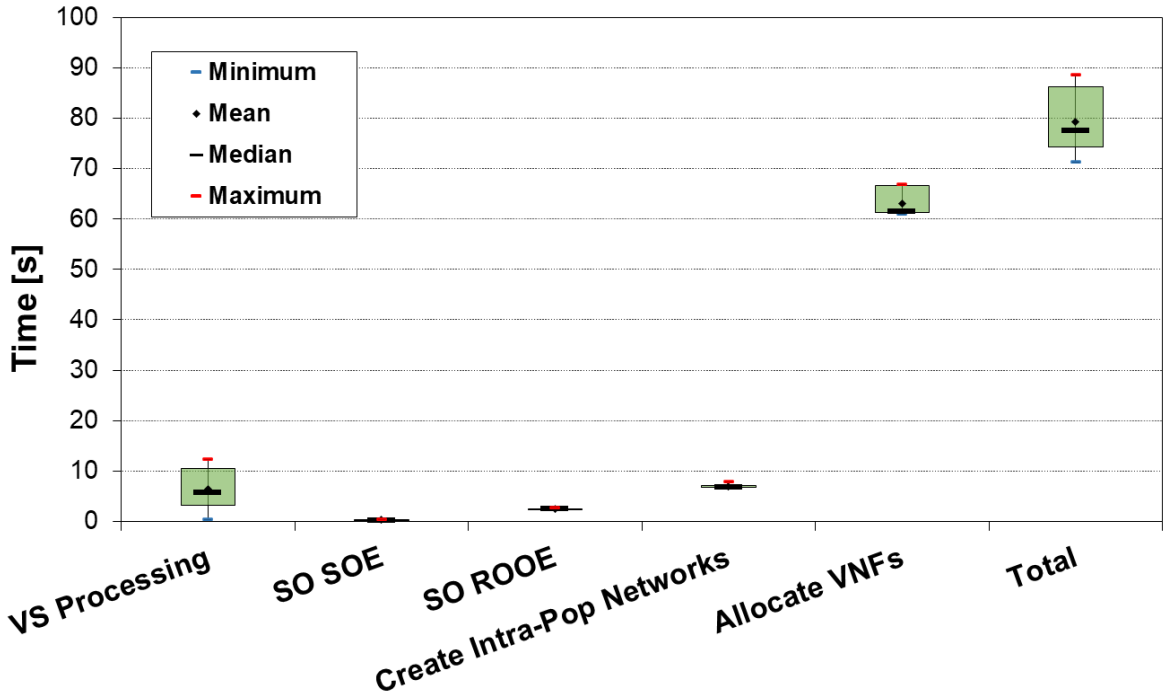


Fig. 2.7 Service creation time for the scaled-out EVS service (2 CD VMs)

if the instantiation process has finished (in addition to the internal processing, which is much lower). The configured polling period is of 20 s. Next, the 5GT-SO Resource Orchestrator Engine (ROE) processing accounts for the interaction with the 5GT-MTP to retrieve the topology and available resources (with the largest component equal to 1.964 s on average), the interaction and placement calculation in the Placement Algorithms (PA) server (522.3 ms), and other much smaller components. Finally, the processing in the 5GT-SO Service Orchestrator Engine (SOE) lasts 292.6 ms on average. This operation accounts for the time it takes to the SOE to interact and coordinate the operations at the different entities in the 5GT-SO module.

Fig. 2.8 presents the service creation time for the *full-fledged* VS service featuring same pattern of EVS service creation time in terms of relative importance of the components. The only remarkable differences compared to the above EVS service are the larger time for intra-PoP network creation (12.862 s on average) and the smaller time for VNF allocation (29.732 s on average). As for the former, this happens even if the service to deploy is simpler (i.e., 2 VNFs instead of 4), because in the previous case all 4 VNFs were deployed within the same host (the MEC in Torino) and the same intra-PoP networks were used by all of them. Conversely, in this case each VNF is deployed in a different PoP. Therefore, two sets of intra-PoP networks must be created and the 5GT-SO must interact

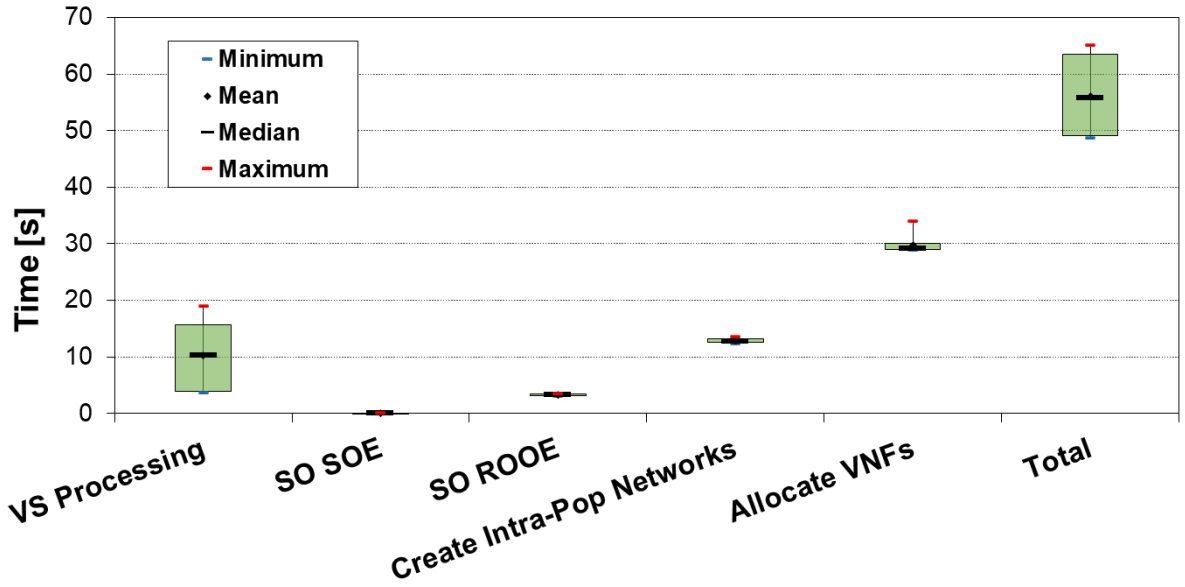


Fig. 2.8 Service creation time for the full-fledged VS service

twice with the 5GT-MTP, which, in turn, interacts with different Openstack instances to create them. Furthermore, since these two intra-PoP networks must be stitched to allow both VNFs to interact as part of the vertical service logic, the deployment time of inter-PoP logical links is not zero (293.2 ms on average), which also makes the processing time at the ROE larger (3.271 s vs. 2.501 s on average), despite being a simpler service. This time also includes the interaction with the 5GT-MTP, which, in turn, interacts with the WAN controller to configure the required transport network connection [19]. Finally, the shorter VNF allocation time in this case is due to the fact that only two VNFs, instead of 4, have to be deployed by the respective Openstack intances at each PoP.

Fig. 2.9 presents the experienced service creation time for the different NFV-NSs and ILs presented in Table 2.1, ordered by the total amount of VNFs in the NFV-NS. We can observe that similar considerations can be made for the other services involved (i.e., 1-CD EVS and *reduced* VS). The main components are the time for allocating VNFs and the time for creating intra-PoP networks, with the latter being of the same order as that obtained for the EVS service with two CD VNF instances, since there is only a single PoP involved.

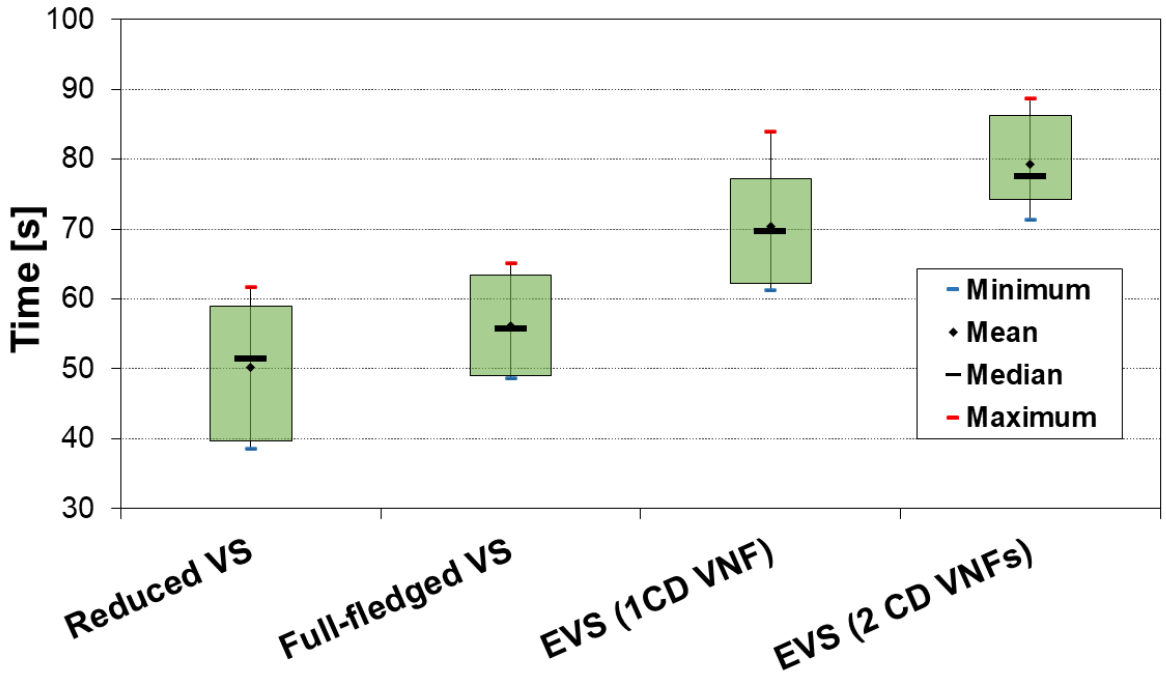


Fig. 2.9 Service creation time for the different considered NFV-NSs and ILs

2.5.3 Service-level scaling – Phase 2

As mentioned, in Phase 2 an EVS service creation request arrives at the 5GT-VS, but this vertical is already running two VSs (i.e., *full-fledged* and *reduced* VS) that consume most of the available resource budget. As a consequence, the *Arbitrator* decides to terminate the *full-fledged* VS service to make resources available for the (higher priority) EVS service.

Fig. 2.10 shows the three components of the service-level scaling time. First, the Rx-to-decision time accounts for the time it takes to the *Arbitrator* to realize that the new service request does not fit in the vertical's budget and to decide to terminate the *full-fledged* VS service. This is the smallest component, since it only involves internal processing inside the 5GT-VS (165 ms on average). Out of this time, 45% (on average) is spent inside the *Arbitrator* module. Second, VS service termination accounts for all the operations (including interactions) carried out into the 5GT platform stack to remove the inter-PoP logical links, to terminate the VMs, and to terminate the intra-PoP networks created in the PoPs. These interactions are triggered by a message from the 5GT-VS to the 5GT-SO once the decision has been made at the *Arbitrator*, and it involves the SOE, the wrapper, the core MANO platform, the ROE inside the 5GT-SO and its interaction with the 5GT-MTP. The latter, in turn, interacts with the underlying network

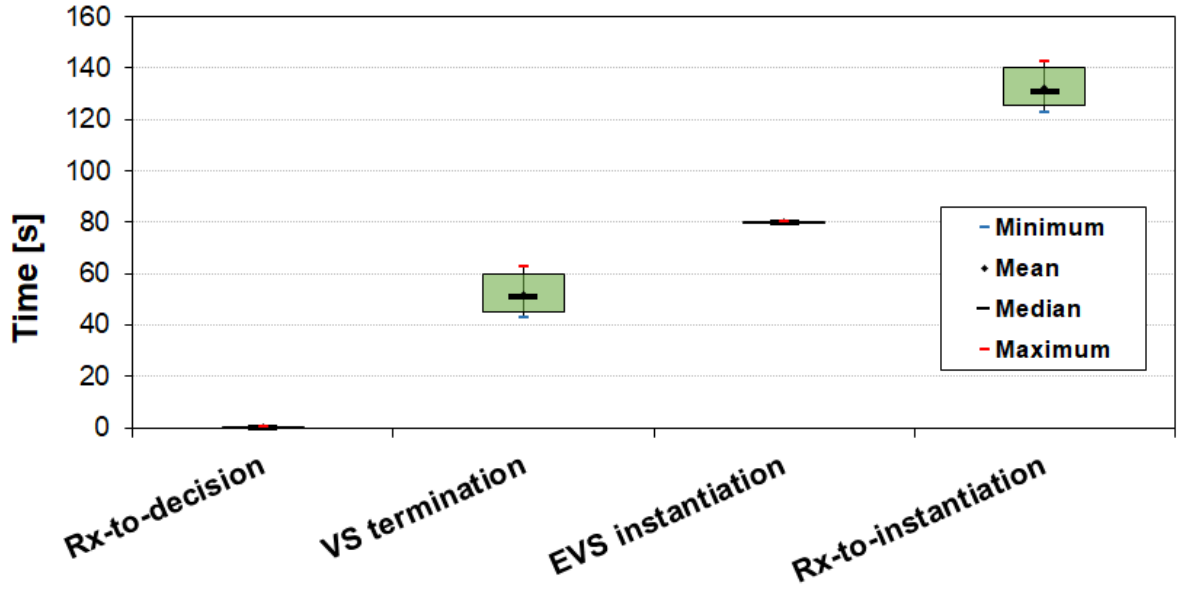


Fig. 2.10 Service-level scaling time

infrastructure (WAN controllers) and Openstack instances of the involved PoPs (acting as VIMs). Finally, the databases at all layers are also updated. The whole process takes on average 52.020 s. Third, once enough resources are freed for the new service, the EVS instantiation phase accounts for the deployment of the EVS service (80.009 s on average). In this case, the deployed EVS NFV-NS counts a single instance of the CD VNF. As shown in Fig. 2.10, the addition of these three components (Rx-to-instantiation) results in a total average of 132.194 s, from the reception of the high priority service request to its instantiation after having terminated other low priority service.

2.5.4 Resource-level scaling – Phase 3

We also evaluated the resource-level scaling process by the 5GT-SO, according to the auto-scaling rules that are generated as a consequence of the evaluation presented in Sec. 2.5.1.

Fig. 2.11 shows the components of the scale-out operation for the EVS service. Out of the total scale-out time (30.862 s on average), the largest time component corresponds to the deployment of a new instance of the CD VNF in a VM (see *Wrapper apply* component of Fig. 2.11). In this case, this step (30.481 s on average) accounts for the time from requesting scaling to the wrapper (sent by other building blocks of the 5GT-SO), the interaction with OSM and the 5GT-MTP, and the interaction with Openstack to deploy

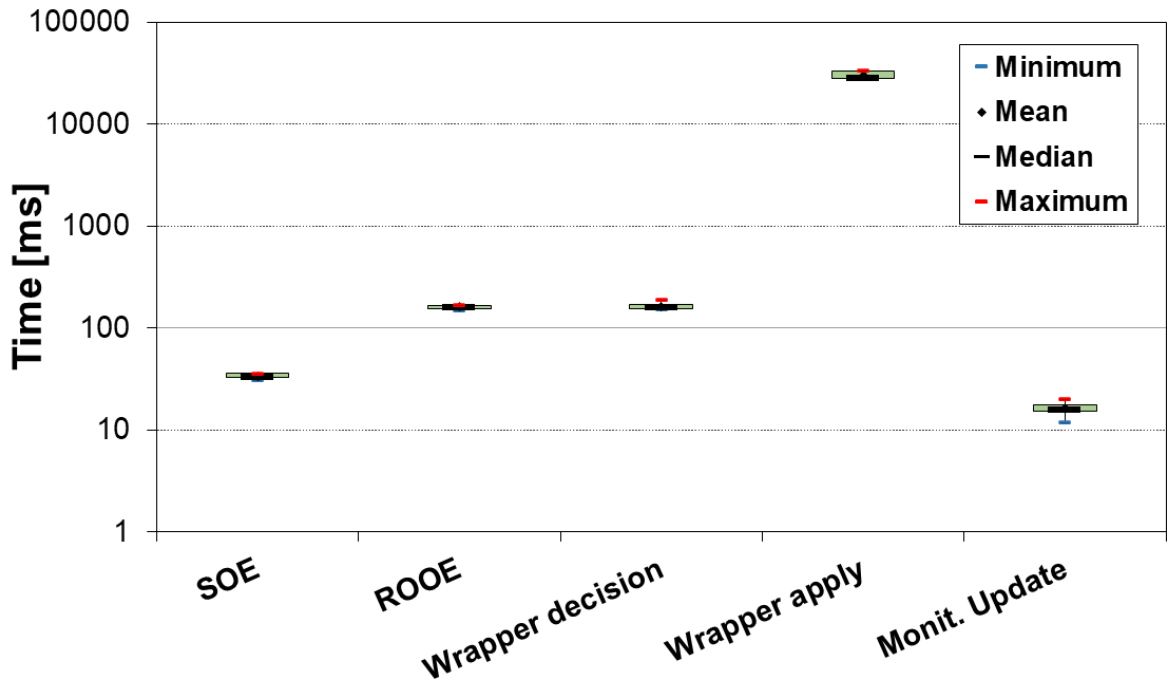


Fig. 2.11 Scale-out breakdown time analysis for the EVS service

the new VM and to attach it to the corresponding intra-PoP network. The remaining components are much smaller (tens or hundreds of ms), as depicted in Fig. 2.11 (notice the applied logaritmic scale). First, the SOE processing (33.8 ms on average) measures the time the SOE takes to handle the scaling request coming from the SLA *Manager* as a consequence of an alert being triggered at the 5GT-MON. This alert (and the associated monitoring job) was previously configured at instantiation time based on the scaling rules (and monitoring jobs) in the NSD of the EVS service. Database update operations are also included in the SOE processing time. Second, the ROE processing (161.8 ms on average) accounts for the time it takes to prepare all the information to be sent to the wrapper to switch from the initial EVS (the running service) to the scaled-out EVS (with an additional CD VNF instance to balance the vertical service load). Furthermore, the ROE is also in charge of creating the new logical links between the new CD VNF and the rest of VNFs of the service (in the same way it was done at the instantiation time for the original one) to maintain the service logic for the vehicles that are going to be served by the new VNF. Third, Wrap decision (163.9 ms on average) accounts for the time to prepare/translate the scaling request received by the wrapper to trigger the associated procedure at the core MANO platform (i.e., OSM). Finally, update monitoring (16.3 ms on average) measures the time to update the monitoring jobs to also monitor the new

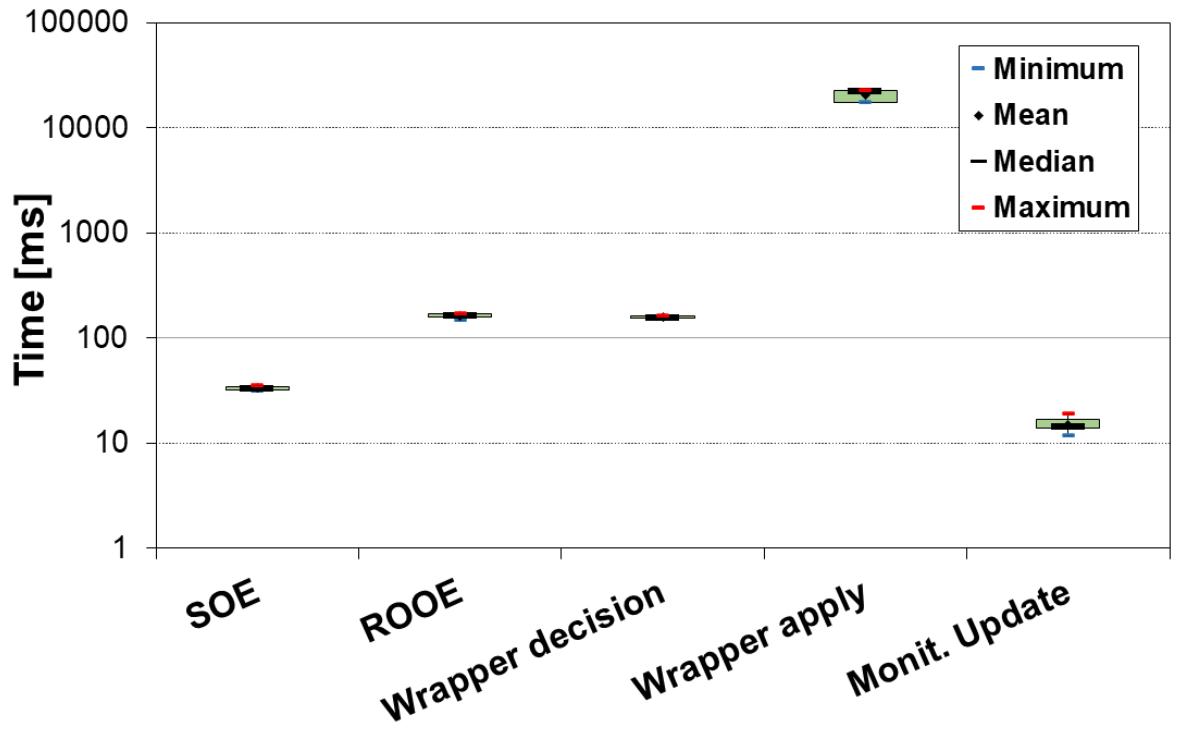


Fig. 2.12 Scale-in breakdown time analysis for the EVS service

created CD VNF instance and to make this data accessible through the corresponding interface.

Fig. 2.12 shows the components of the scale-in operation for the EVS service. in Fig. 2.12. This is the process through which the 5GT-SO autonomously decides to downscale the resources assigned to the service by changing the scaled-out EVS to the initial EVS IL with only one CD VNF. The scale-in process is triggered by the 5GT-MON by noticing that the CPU consumption is below a certain threshold. The process is equivalent to the above one but for freeing resources (logical links and VMs) instead of creating them. Terminating services and freeing resources (scale in) takes less time (21.051 s on average) than allocating resources (30.862 s on average for scale out), as can be observed in Fig. 2.12. More specifically, all components described above other than those related with the core MANO platform (i.e., OSM) are very similar: (i) SOE processing (33.5 ms), (ii) ROE processing (163.7 ms), (iii) Wrap decision (157.9 ms), and (iv) Monitoring update (15.0 ms). However, the main component (Wrap apply), related to releasing the resources by interacting with the 5GT-MTP (and Openstack), is 10 s smaller (20.676 s vs. 30.481 s on average).

2.6 Related Work

Several resource and service orchestration solutions have been investigated (both within and outside NFV scope) to effectively improve user experience and SLAs [26, 27]. *SLA management* is also a quite well-investigated topic within network and service management in different areas, e.g., cloud computing [28], enterprise networks [29], web services [30], and considering multi-domain scenarios [31]. Nonetheless, an important challenge still needs to be addressed: how to automate SLA management operations to avoid, or promptly respond, to possible agreement violations. Existing solutions to this issue often differ in the level of dynamicity of the considered context, i.e., how quickly the resource usage changes. Most of them, however, focus on making monitoring and enforcement tasks as flexible as possible [32].

In the context of network slicing and of the NFV ecosystem, SLA management has mainly focused on the automated scaling of network services, performed by NFV orchestrators to meet the target KPIs despite load surges [33]. In particular, most works have dealt with mechanisms based on virtual resource usage prediction [34], or strategies for virtual resource reallocation [35]. In these cases, policy-based rules and input data used for triggering scaling are provided manually for each network service. In [36], scaling rules are specified in the network service descriptors, which is indeed a preferable option toward more agile and less error-prone scaling solutions in automated slice management operations. In [37], an autonomic policy-based network service deployment with SLA is presented where high-level parameters (performance, availability, security) specified in the SLA are linked to low-level requirements encapsulated in the respective policies. The policy associated with a network service-level SLA is included in a specific descriptor that is created, validated, and uploaded to a catalogue and then applied when the network service is instantiated. Other scaling solutions rely on Artificial Intelligence (AI)-assisted operations to adapt slice or VNF capacity under varying user demand [38] or while minimizing the waste of resources [39].

Recently, consensus has emerged on the need for a network slicing-aware NFV orchestration where management and orchestration functionalities are extended to the slice level (i.e., managed at the telco OSS/BSS), so as to handle end-to-end operations and efficiently address SLAs negotiated with verticals [40]. The study in [41] addresses the modeling, deployment, and orchestration of an end-to-end network slice, which includes the RAN, core, and transport network. Slice management functions are performed

through the Open Network Automation Platform (ONAP) [3], and the authors propose an architecture to enforce negotiated SLAs. This solution exploits monitoring information and the policy enforcement component from ONAP to realize automated closed-loop management, while scaling operations are presented at the conceptual level. A data-driven approach for intelligent slice management is presented in [42]. Therein the authors propose a framework for data-driven slicing resource provisioning, including the development of slice traffic predictors, resource allocation models, and constrained SLA enforcement. Both the above works, however, address SLA during slice deployment (i.e., SLA enforcement), while they do not focus on scaling operations. Additionally, some recent works have envisioned SLA solutions in the context of network slicing for 5G services, also leveraging AI-based solutions. Most of them, however, address SLA from a theoretical point of view [43–46], or, even if they present operational solutions to NFV orchestration, they focus on the service deployment phase without specifically addressing scaling operations [47, 48].

To the best of our knowledge, there exist only few works on the application and benchmarking of SLA management in combination of scaling operations, triggered by management and orchestration platforms (MANO) in experimental setups. In addition to the system evaluation we present, and demonstrated in [21], our framework is designed to integrate a hierarchical SLA management in NFV orchestration, involving multiple levels at which service scaling can be handled.

In this context, relevant works to ours are [49][50]. The study in [49] presents a benchmarking analysis with respect to scaling, between the SONATA MANO platform and other open-source MANO solutions like OSM and Cloudify. In [49], however, the SLA management is performed at the NFV-NS level only, unlike in our work where this is just one of the possible levels at which we can act. [50], instead, presents an integrated SLA management framework within the SONATA MANO platform for real 5G environments, aiming at binding business requirements between network operators and verticals, with measurable attributes. The framework proposed therein is demonstrated as a web and multi-platform application that allows the management of the whole SLA lifecycle for a network service. After the network service instantiation into a network slice, the SLA framework is populated with infrastructure monitoring information, in order to assess the agreement with real-time usage data, and efficiently avoid or manage possible violations. However, the scaling needs of the verticals or shared slice subnets are not considered in the scaling process.

[3] https://www.onap.org/

Ultimately, most of the above previous works on SLA management focus on the network service and the resource level, with the aim to let them adapt to the time-varying resource demand as well as resource availability from the underlying infrastructure. To our knowledge, none of them accounts for the vertical services and the application level, or takes into account dynamic changes in the service demand and/or in the application needs.

As far as *service arbitration* is concerned, a large body of work has addressed call and service admission control in the context of wireless networks, focusing on radio resource allocation (see, e.g., [51], or [52] for a survey on this topic). Relevant examples of works on resource allocation include [53, 54]. In particular, [53] leverages reinforcement learning to proportionally allocate budget-constrained radio resources to competing services whose properties are partially unknown at the time of decision making. [54], instead, introduces methods for computational resource arbitration among virtual networks within a node, and for migrating network functions among nodes within a virtual network. Note however that, unlike resource arbitration in 5G, to the best of our knowledge the problem of service arbitration aimed at guaranteeing the SLAs between verticals and the 5G provider has not been previously tackled. Indeed, none of the existing studies have considered the support of vertical services in a 5G network, accounting for SLAs in place between verticals and network provider. This scenario implies not only a finite resource budget for a set of services, but also that resources are properly allocated (i) using a coarse knowledge of the resource status such as that available from a business perspective, and (ii) in a way that the allocation itself can be varied over time so as to adapt to the network and services dynamics as well as to the target KPIs specified by the verticals. An initial study on such aspects was presented in our conference paper [55].

In summary, though previous works dealt with various aspects of SLA management, this has been done in a quite focused way for the specific problem at hand. When moving to complex and heterogeneous virtualized networks, the number of network components, and most importantly, of architectural objects to manage (e.g., vertical service, network slice, network service, virtual function, virtual link) substantially increases. This also has implications in the layers and entities included in the MANO stack (see Fig. 2.1). Therefore, when deploying a given service, typically there exist SLA constraints involving different architectural objects associated to different architecture layers and stakeholders (e.g., verticals, service providers, operators, infrastructure providers). Unlike previous studies, our work presents a global hierarchical framework for SLA management that is capable of handling SLA at various layers, it is hence concerned with various key architectural objects related with the vertical service, network slice, and network service

(including associated resources). Furthermore, the understanding of the dynamics of SLA management and scaling procedures in operational deployments and the availability of representative datasets allow for an efficient integration in our proposed approach of functional AI-based solutions, as presented in [56].

2.7 Conclusions

5G networks have expanded the scope of traditional mobile networks to support the digital transformation of vertical industries such as automotive, factories, media, e-Health, and robotics. It follows that nowadays telco providers need to simultaneously deploy and manage multiple vertical services over a shared mobile network infrastructure. In this chapter, we presented a hierarchical service and SLA management framework, which leverages service scaling mechanisms at different levels, namely, application-, service- and resource-level, and we have implemented the proposed framework and different scaling functions over the 5GT platform. We demonstrated the performance of our solution using a proof-of-concept testbed. Our results, obtained through the real field tests with relevant automotive vertical services, show the feasibility of the proposed solution and its ability to automatically deploy and update service instances, while fully meeting the established SLAs. Importantly, the hierarchical service and SLA management framework presented here has been adopted as a baseline solution for future 5G vertical industry technology developments, as the ones considered in the 5Growth project [57][58].

It is important to mention that the 5G-TRANSFORMER project has been participated by many academic institutions, research centers and industrial companies in Europe, each with their contribution to the project. The work presented in this chapter, specifically the SLA management framework and the Proof-of-Concept development and testing, focuses on the author's original contribution but it would not have been possible without the contribution of the various partners.

In the next chapter, we will use the 5Growth platform, derived from the 5G-TRANSFORMER project, to develop the ML-as-a-Service (MLaaS) concept. Indeed, Machine Learning is envisioned as the technology needed to make networks predictive and proactive, essential features for truly autonomous networks. We will propose the MLaaS Platform, a new component of the 5Growth architecture able to train and serve ML models to other elements of the architectures to enhance their decision-making capabilities. We will propose two ML-driven algorithms as examples of the policies that can be enabled by the MLaaS Platform, namely the network slice subnet sharing, and,

once again, the service scaling. The main difference with the service scaling as proposed in this chapter is that now the scaling command will not be directly generated by the 5GT-SO when a predefined CPU load threshold is reached, but will be provided by an ML model specifically trained to prevent SLAs violations.

Chapter 3

ML-driven Provisioning and Management in Automated Cellular Networks

One of the main tasks of new-generation cellular networks is the support of the wide range of virtual services that may be requested by vertical industries, while fulfilling their diverse performance requirements. Such a task is made even more challenging by the time-varying service and traffic demands, and the need for a fully-automated network orchestration and management to reduce the service operational costs incurred by the network provider. In the previous chapter, we tackled these problems by proposing a framework able to meet Service Level Agreements (SLAs) by leveraging the service scaling mechanism. Instead, in this chapter, we address these issues by proposing a softwarized 5G network architecture, based on the 5G-TRANFORMER architecture from the previous chapter, that realizes the concept of ML-as-a-Service (MLaaS) in a flexible and efficient manner. The designed MLaaS platform can provide the different entities of a MANO architecture with already-trained ML models, ready to be used for decision making. In particular, we show how our MLaaS platform enables the development of two ML-driven algorithms for, respectively, network slice subnet sharing and run-time service scaling. The proposed approach and solutions are implemented and validated through an experimental testbed in the case of three different services in the automotive domain, while their performance is assessed through simulation in a large-scale, real-world scenario. In-testbed validation shows that the use of the MLaaS platform within the designed architecture and the ML-driven decision-making processes entail a very limited

time overhead, while simulation results highlight remarkable savings in operational costs, e.g., up to 40% reduction in CPU consumption and up to 30% reduction in the OPEX.

Part of the work described in this chapter has been already published in C. Casetti et al., "ML-driven Provisioning and Management of Vertical Services in Automated Cellular Networks," in *IEEE Transactions on Network and Service Management*, doi: 10.1109/TNSM.2022.3153087, ©2022 IEEE.

3.1 Introduction

5G systems have been touted as capable of delivering an advanced platform primed to cater for vertical services, creating an ecosystem ripe for technical and business innovation. This crucial selling point for 5G has been addressed, in the years leading up to its commercial deployment, by several research projects attempting to identify challenges, problems and requirements of vertical industries that could be targeted by 5G capabilities. The '5G Promise' hinges, first and foremost, upon the ability to create an interface that effectively matches offer and demand between a network provider and a vertical, its customer. In other words, high-level Service Level Agreement (SLA) business requirements for the service instances that a vertical requests, must be mapped onto slice- and infrastructure-related requirements, which are reflected by the underlying network-level setup.

Among the projects [59, 58] that have addressed Machine Learning (ML) in the context of 5G networks, 5Growth [58] has sought to take this mapping one step forward, by applying ML capabilities to both the characterisation of the network context where traffic slices are deployed, and to the scaling of instantiated virtual services vis-a-vis unexpected surges in resource demand. To this end, we enhance the 5Growth architecture with an ML-as-a-Service (MLaaS) platform.

Leveraging a modular design, the MLaaS platform is equipped with a computing cluster that supports a large variety of ML models, which are uploaded to be trained and whose lifecycle is seamlessly managed. An interface with a monitoring platform collecting real-time data through Kafka feeds data to the model. Such ML models are then used for fully-automated service provisioning and management within the 5Growth architecture, a paradigm that is widely recognized as highly needed for 5G-and-beyond networks [60, 61]. In particular, looking at the virtual services requested by a vertical, we leverage the ML models to address two important challenges: (i) when and how to share

network slice subnets among concurrent service instances, and (ii) when and how to scale such services, while accounting for both key performance requirements and OPEX.

We do so by providing the following main contributions:

- *Architectural design:* we define the internal architecture of the MLaaS platform, as well as enhancements to the 5Growth Vertical Slicer and Service Orchestrator entities, and the corresponding workflows, enabling the use of trained ML models for fully-automated service provisioning and management;
- *Algorithm design:* we introduce two ML-driven algorithms solving the problem of, respectively, network slice-subnet sharing [62] at the Vertical Slicer and run-time service scaling [63] at the Service Orchestrator. Both algorithms can swiftly adapt to time-varying load conditions, by leveraging the output of ML models to dynamically set their driving input parameters;
- *Experimental testbed validation and performance results:* we demonstrate the feasibility of our approach through a testbed implementing the whole 5Growth network architecture and the workflows between the aforementioned entities. Experimental results show the reduced impact of the ML-driven approach in terms of overall service instantiation and scaling time. Further, we assess the performance of the proposed ML-driven algorithmic solutions through simulations in a large-scale, real-world scenario, achieving up to 40% reduction of resource consumption in the case of slice-subnet sharing, and about 30% reduction of the OPEX in the case of service scaling.

As better discussed in Sec. 3.8, the scope of our work and the solutions we present differ substantially from existing research on network slicing and resource allocation. Unlike previous work, we address *network slice-subnet sharing among different services*, rather than resource allocation sharing among different network slices. Moreover, while designing our service scaling scheme, we account for *both SLA violation costs and operational costs*, beside time-varying traffic load demands. In both cases, we design a novel architecture of the 5G network platform that can effectively leverage MLaaS for fully-automated service provisioning and management. Finally, as already mentioned, we provide a complete evaluation of the proposed framework that not only shows the performance of our solution, but also validates the interaction and functional synergy between the 5G architectural entities.

The rest of the chapter is organized as follows. Sec. 3.2 introduces the recent 3GPP standards for slice management and describes how the 5Growth network architecture,

including our MLaaS platform, provides a custom implementation of such standard specifications. Sec. 3.3 provides an overview of the proposed solutions for (i) network slice-subnet sharing and (ii) runtime slice adaptation to dynamic traffic conditions. The two solutions are then detailed in Sec. 3.4 and Sec. 3.5, respectively. Both experimental results obtained through the testbed we developed and simulation results in a large-scale scenario are shown in Sec. 3.7, for the automotive services described in Sec. 3.6. Finally, Sec. 3.8 discusses previous work, while Sec. 3.9 draws some conclusions.

3.2 Network Platform Architecture

This section presents some preliminaries on the 3GPP architecture for network slice management and orchestration, as well as for data analytics functions supporting closed-loop, cross-layer network control automation (Sec. 3.2.1). This allows us to highlight how the design of the proposed MLaaS platform, and its interactions with other 5G network entities, is fully compliant with 3GPP standards. We then present the 5Growth architecture [58, 64], mapping its components into the relevant ones of the standard 5G Management System (Sec. 3.2.2). Finally, we detail the architecture of our proposed MLaaS platform and how this addresses the need for ML models for fully automated service management, network orchestration, and resource control within the 5G network architecture (Sec. 3.2.3).

3.2.1 3GPP management system and data analytics

Network slicing is one of the key features of 5G networks that allows creating multiple and concurrent logical networks, called network slices, over a shared physical infrastructure, each of them with its own key performance indicators (KPI) requirements, and security and isolation guarantees. An end-to-end 5G network slice spans both the radio access network and the 5G core network, and it includes a number of Network Functions (NF) which can be virtualized, deployed, orchestrated, and managed through the 5G Management System. As per 3GPP standards, a *network slice subnet* is a representation of the management aspects of a set of NFs managed through the 5G Management System and their required resources (e.g., compute, storage, and networking resources). These NFs, when virtualized, can be modelled as Virtual Network Functions (VNFs) and they can be combined together in an NFV Network Service (NFV-NS). In this sense, the

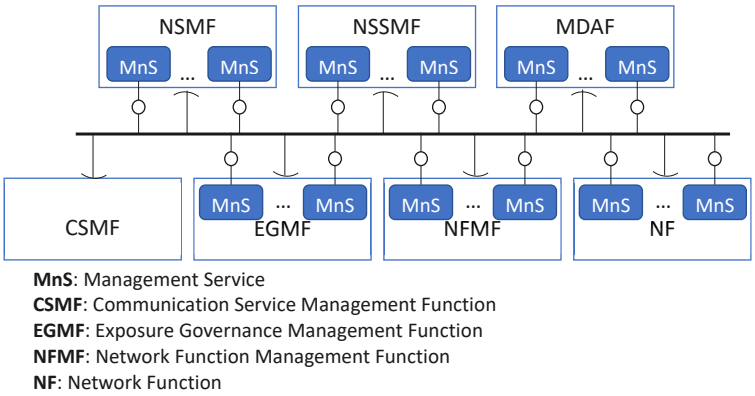


Fig. 3.1 3GPP Service Based Management Architecture [65], including the MDAF block for fully-automated service management and resource orchestration.

NFV-NS provides the real implementation and deployment of a network slice subnet, i.e., a *network slice subnet instance (NSSI),* in the virtual infrastructure.

The latest 3GPP standards [65] propose a Service Based Management Architecture (SBMA) for the 5G Management System (see Fig. 3.1), which includes Management Functions to deliver a variety of Management Services (MnS) to handle the management of single functions (NFMF), slices and communication services, as well as the exposure of the various services towards external entities (EGMF). In particular, the SBMA includes a Management Data Analytics Function (MDAF) to provide analytics services in support to automated network management and orchestration decisions. These decisions drive the logic of the Network Slice and Network Slice Subnet Management Functions (NSMF and NSSMF), which are in charge of handling the lifecycle of the 5G network slices and the orchestration of their resources across the various NSSI realizing an end-to-end slice.

At last, notice that the MDAF consumes monitoring data or records retrieved from the network and its management system, e.g., related to existing network slices and network service requests, or VNF performance, and yields analytics results to drive decisions related to the lifecycle management of network slices or the orchestration of network services.

3.2.2 The 5Growth architecture: a custom implementation of the 3GPP management system

The 5Growth architecture, depicted in Fig. 3.2, is based on a hierarchy of functional elements operating at the different layers of a 5G network management system, from

vertical service and network slice management, down to the orchestration of NFV services and infrastructure resources. They offer the management and orchestration (MANO) functionality [66] required to handle the lifecycle management of all architectural objects involved (i.e., vertical services, network slices, network slice subnets, NFV composite and nested network services, VNFs, and virtual links).

These management elements are assisted in their decisions and automated actions by an MLaaS platform (5Gr-MLaaSP), which is used to build ML models trained with multisource data collected through a cross-layer monitoring system. Thus, the 5Gr-MLaaSP prepares the models that drive the closed-loop actions taken at the different architectural layers on the basis of a multi-variable real-time context derived from the records maintained at each layer (e.g., for service demands, network slice instances, and their subnets) as well as from real-time monitoring data (e.g., on virtual resource consumption). In more details, the 5Growth architecture includes three functional elements: the Vertical Slicer (5Gr-VS), the Service Orchestrator (5Gr-SO), and the Resource Layer (5Gr-RL).

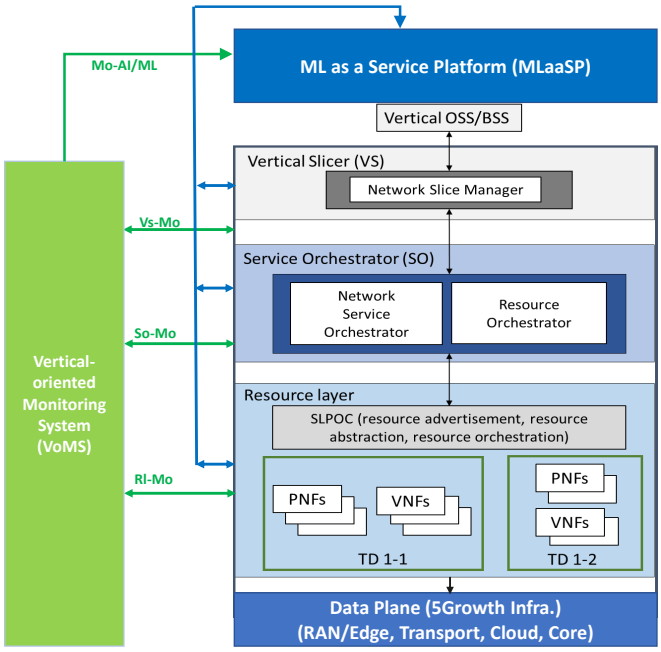


Fig. 3.2 The 5Growth architecture.

The 5Gr-VS handles the lifecycle of vertical services, their mapping into end-to-end network slices, and the provisioning and management of the slice subnets. The Vertical Service Management Function (VSMF) of the 5Gr-VS processes requests for Vertical Service Instances (VSI), defined through vertical service descriptors specifying

the desired characteristics in terms of application-level parameters. On the basis of such characteristics, the VSMF identifies the Network Slice Instance (NSI) required to properly host the service, implementing the functionalities of an extended, vertical service-aware Communication Service Management Function (CSMF) within the 3GPP Management System. The provisioning of a network slice is handled by an enhanced NSMF embedded in the 5Gr-VS. This procedure involves the creation of the NFV-NSs implementing the NSSIs composing the end-to-end NSI associated with the VSI. These 5Gr-VS functionalities provide a concrete, custom and data-driven implementation of the 3GPP NSMF and NSSMF components, which coordinate the instantiation and deployment of the NFV-NS corresponding to NSSIs over an NFV MANO-like system represented by the 5Gr-SO. Each NFV-NS is deployed with the deployment flavour and *instantiation level (IL)* able to guarantee the vertical service requirements specified in the original request. Network slice subnets can be shared among multiple slice instances and, thus, among multiple service instances. Where needed, the 5Gr-VS may also trigger their scaling (i.e., modifying the IL of the corresponding NFV network services) to properly host additional vertical service instances. Sec. 3.4 describes the ML-driven solution implemented to decide how to efficiently share network slice subnets among concurrent vertical services with different latency requirements.

The 5Gr-SO handles the lifecycle management of NFV-NSs that build the slice subnets. For this purpose, it can handle both regular and composite NFV-NSs. It receives the NFV-NS requests from the 5Gr-VS and the available resources in the infrastructure from the 5Gr-RL (see below), and maps such requests over the infrastructure to fulfill their requirements, including sending requests for configuring virtual function inter-connectivity through the transport network. In this direction, it coordinates the automated provisioning, monitoring, AIML model set up, and scaling of the virtual functions that compose the NFV-NS, according to model outputs.

The 5Gr-RL implements resource allocation operations in the underlying NFV infrastructure, abstracting the capabilities of the access, transport, and edge/cloud computing resources exposed to the 5Gr-SO. A Vertical-oriented Monitoring System (5Gr-VoMS) collects metrics and logs from these three functional elements, implementing a centralized and multi-layer monitoring platform. 5Gr-VoMS stores data related to the usage of physical and virtual infrastructure resources, to measure the performance of NSSIs or end-to-end network services, or service-level metrics collected from vertical applications. Monitoring data is used to feed the decision engines at each layer and, in particular, is used as input for the 5Gr-MLaaSP to build training datasets. The 5Gr-MLaaSP constitutes the 5Growth concrete implementation of the MDAF within the 3GPP Management

System. More specifically, in this work, the 5Gr-MLaaSP is used to build trained ML models to support decisions about two orchestration actions performed at different levels of the 5Gr architecture, namely, network slice-subnet sharing and NFV-NS scaling at the 5Gr-VS and 5Gr-SO levels, respectively.

The multi-layer nature of the 5Growth architecture is fully compliant with the principles of the control loop applied at different layers of the 3GPP Management System [67], as represented in Fig. 3.3, with the mapping to the 5Growth components. In the 5Growth architecture, the 5Gr-VS and 5Gr-SO make decisions and enforce actions at the level of network slices and NFV network services, respectively, thus operating at the network slice level and at the network slice subnet level. In fact, in 5Growth the NSSIs are built and operated as NFV network services, where their internal lifecycle is managed directly by the 5Gr-SO, while their composition and sharing in end-to-end network slices is handled at the 5Gr-VS. Following this model, on one hand the 5Gr-VS makes higher-level decisions that map the vertical service demand into network slices which can be decomposed in cross-service subnets shared among multiple slice instances. On the other hand, the lifecycle of the NFV network services building the single NSSIs is handled at the 5Gr-SO level, which makes decisions and enforces actions for their automated scaling. Both components are supported by the 5Gr-MLaaSP, which provides the data analytics functionalities. The 5Gr-MLaaSP builds datasets for training ML models using the multi-layer data collected by the 5Gr-VoMS. Such entity records statistics, metrics and KPIs at different layers, from single VNFs, e.g., in terms of consumed virtual resources or application indicators, up to NFV Network Services, Network Slices, and Vertical Services.

In summary, our architecture (i) translates vertical service requirements into network service requirements, and (ii) integrates ML in the management and orchestration workflows. In this sense, our approach is in accordance with the ETSI NFV guidelines, in which a generic architecture is devised to handle virtual services and virtual functions. In other words, the internal logic of the service should be handled by the service itself, while the architectural framework should deal only with services requirements that are expressed through generic parameters (e.g., virtual link bandwidth, number of CPUs, geographic location), and not through parameters that have to do with the specificity of each service. This makes the proposed architecture able to cope with any type of service.

At the same time, however, the architecture also enables decisions to be made depending upon the specific service. This is realized by downloading through the 5Gr-MLaaSP open API, and then by running, a trained ML model that is suitable for the

service at hand. Indeed, the 5Gr-MLaaSP features a library of ML models that are fully aligned with the service offer of a given operator. Thus, by combining the generality of the architecture with a rich set of ML models, our framework can handle any kind of service, whilst also adapting to its specific operational goals.

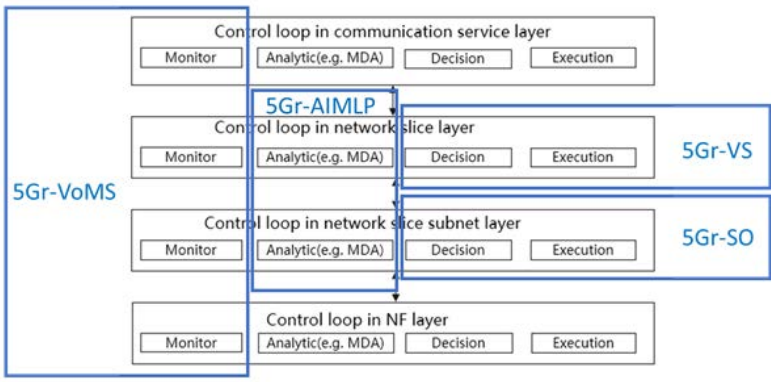


Fig. 3.3 Closed-loop control applied to different management system layers [67] and mapping onto the 5Growth architecture.

It is also worth noting that the 5Gr-MLaaSP design matches the most recent updates in the internal architectures of the 3GPP data analytics functions (NWDAF - the 5G Core Network function for data analytics - and MDAF). In particular, the old monolithic structure of the NWDAF is evolving in the latest versions of the 3GPP Rel. 17 specifications [68] to better fit the adoption of AI/ML techniques. Based on this model, the NWDAF is split into two different logic functions: the former is devoted to the training of ML models, with the capability of providing trained models towards external functions, while the latter implements the analytics service, e.g., performing inference and computing statistics or predictions. The NWDAF Analytics logical (AnLF) function acts as consumer of the NWDAF Model Training logical function (MTLF), exploiting its APIs for the discovery and exchange of trained models. A similar concept is applied in the design of the 5Gr-MLaaSP that implements the MTLF functionalities, while the decisions are delegated to other components of the 5Growth architecture that act as consumers of the 5Gr-MLaaSP services. In detail, the 5Gr-VS and 5Gr-SO are both in charge of retrieving the most suitable trained model from the 5Gr-MLaaSP (e.g., periodically or on-demand, based on the configured policies), which is then used to feed the analytics algorithms responsible for decision making. This approach fully decouples the training from the real-time analytics phase, allowing for autonomous updates of the trained models.

Importantly, there are various timescales at which models may be updated. And what is also relevant is when and how often models are actually used. As mentioned, the framework allows gathering data for training/updating the models that are already stored in the MLaaSP, and offering them to model consumers through open APIs. This allows, for instance, that every time a new service is instantiated, the updated version of the model is downloaded and run. Since dynamicity is also reflected in virtual services being continuously deployed and terminated, the operator benefits from such updates as new services are deployed.

3.2.3 MLaaS for automated network management

The architecture and the fundamental workflow of the 5Gr-MLaaSP we designed and developed are depicted in Fig. 3.4, along with the other architectural entities with which the main interactions take place. The main components of the 5Gr-MLaaSP are as set forth below.

External Interface, through which an authorized external user can upload a model. The model may be already trained and onboarded, or it may still need to be trained, in which case, the user can provide a suitable dataset to be exploited for the training phase. In both cases, the user can specify (i) the scope of the model, i.e., the type of decision-making process to be used for (e.g., slice sharing, or service scaling), and (ii) the type of service the model/dataset should be used for (e.g., vehicle collision detection, digital twin). When the external user uploads a yet-to-be-trained model, it is the 5Gr-MLaaSP that takes care of the training and records the corresponding timestamp and, potentially, a validity time lapse. If no dataset is uploaded along with the model, the 5Gr-MLaaSP exploits the data collected through the 5Gr-VoMS platform about, e.g., the NSSIs or the network services performance. The configuration of the monitoring platform to gather the monitored data, aggregate it (e.g., through Apache Kafka), and feed it as input for real-time model execution can be properly set up. Models stored in the 5Gr-MLaaSP can be accessed by a 5Growth architecture entity through an open Representational State Transfer (REST) application programming interface (API).

Model Register, which records the models uploaded to the platform, their metadata, and pointers to the stored models and associated files.

Lifecycle Manager, which is in charge of the models lifecycle. Upon the uploading of a new model, it adds the corresponding entry to the Model Register and, if it is a yet-to-be-trained model, it triggers the training process using the appropriate AI/ML

framework. After a model is trained, the Lifecycle Manager monitors its status: it can trigger a new training job either periodically, or whenever new data is available from the monitoring platform.

Interface Manager, which processes the requests for ML models coming from the architectural stack and forwards them to the proper block inside the computing cluster.

Computing Cluster, which is based on Apache Hadoop – one of the few enterprise-grade frameworks guaranteeing high efficiency, concurrency, reliability, and availability. It leverages Yet-Another-Resource-Negotiator (YARN) for the computing resources management, and the HadoopDistributed File System (HDFS) for the storage of datasets and models. The YARN cluster nodes have access to different ML frameworks, according to the requested model type. Apache Spark is used to train classic supervised and unsupervised models, while BigDL is used for Deep Neural Networks[1].

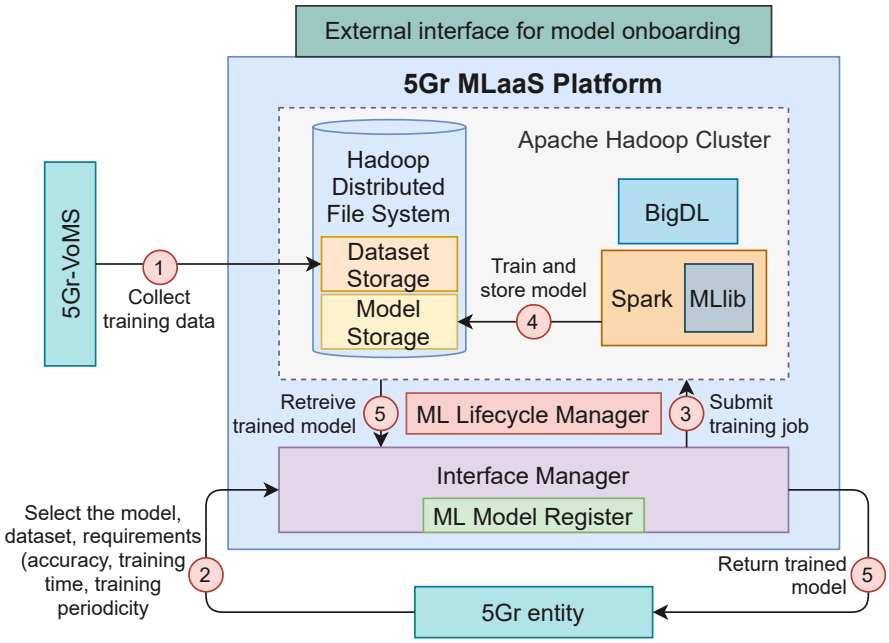


Fig. 3.4 5Gr-MLaaSP architecture and its interaction with other architectural entities.

The envisioned workflow is as follows:

1. Monitoring data from the 5Gr-VoMS can be continuously collected, reformatted, and consolidated to build or update a training dataset. Training datasets are then saved in the HDFS Dataset Storage;

[1]Note that the MLaaSP could be extended to leverage also Ray (https://ray.io/), which can be used to pre-train reinforcement learning models.

2. 5Gr-entities such as 5Gr-VS and 5Gr-SO trigger decision-making processes (e.g., slice-subnet sharing, service scaling) to support service lifecycle management operations (e.g., deployment, scaling) of a vertical service and its underlying network service. To enact such decision-making processes, the 5Gr-entity requests to the 5Gr-MLaaSP the available model catalogue that suits each of the problems at hand. The 5Gr-MLaaSP offers a set of available models, including already trained models as well as models that can be trained on-demand, as indexed by the Model Register. The 5Gr-entity selects the model, and may specify some requirements, e.g., accuracy, training time, or training periodicity so that the Lifecycle Manager can automatically keep the model fit;

3. In case the selected model requires to be trained, either because it has never been trained, its validity has expired, or it needs to be updated, a training job is submitted to YARN. If the requested model is ready to be used, it is directly fetched from the Model Storage;

4. Using the proper dataset from the Dataset Storage, the model is trained using either Spark MLlib, BigDL or Ray, depending upon the model type. The trained model is then saved in the HDFS Model Storage, while the ML Lifecycle Manager tracks the new trained model state and updates the Model Register accordingly;

5. The trained model is finally retrieved from the HDFS and returned to the requesting 5Gr-entity, which is responsible for its online execution.

We underline that, thanks to the ability to continuously collect data through the monitoring platform, the MLaaSP can update an ML model whenever necessary, or deemed useful.

3.3 MLaaS for Automated Network Management in the 5Growth MANO Stack

To highlight how management and orchestration procedures in 5G networks can benefit from the MLaaS approach, we show how MLaaS is exploited at two different layers of the 5Growth stack, namely, the 5Gr-VS and the 5Gr-SO, to solve two different automated network management problems. This is enabled by the available 5Gr-MLaaSP open REST API, which allows consuming ML models from any external entity.

With regard to the first management problem, the 5Gr-VS is in charge of handling the vertical requirements, and in this sense, it deals with business relationships between the 5Growth provider and its customer (the vertical). This vertical service requirements are eventually translated into slice requirements by the 5Gr-VS, which is also in charge of generating the most efficient NFV-NS requests towards the 5Gr-SO based on the slice requirements. As detailed in Sec. 3.4, efficiency at this layer comes from NSSI sharing [62]. That is, if two slices have similar requirements and have part of the slice/service structure in common, instead of fully instantiating a new end-to-end slice, a slice subnet may be reused, with consequent resource savings.

In particular, we focus on latency requirements, hence NSSI sharing requires careful evaluation so that the target latency of both pre-existing and newly requested service instances can be met. This is the reason why the 5Gr-VS embeds an algorithm for classifying slice requests into latency classes as a function of the network context. However, dynamically adapting to such a context (e.g., to the traffic handled by the service entities) requires a careful definition of the latency classes, into which the requested services fall, that are eventually used by the aforementioned algorithm. This is the problem that the 5Gr-VS solves with the help of the ML model provided by the 5Gr-MLaaSP. In fact, at instantiation time, the 5Gr-VS requests, through the open API explained above, the previously-trained model, stored in the 5Gr-MLaaSP database. That is, the 5Gr-VS requests the model to solve the latency class definition problem. Once downloaded (together with the auxiliary code needed to run the model), the 5Gr-VS continuously runs the model to decide on the latency classes used depending upon the scenario conditions.

The second management problem that illustrates the flexibility of the MLaaS approach is still related to fulfilling the vertical SLA requirements, but at the 5Gr-SO, the entity that receives the requirements from the 5Gr-VS and matches them with the resources made available by the underlying infrastructure. As discussed in Sec. 3.5, in this case the focus is on online scaling of nested NFV-NSs [63] based on actual operational data (not on requirements). As we go down the MANO stack, NSIs are requested in the form of composite NFV-NSs to the 5Gr-SO, and NSSIs hence become nested NFV-NSs.

Despite all the care taken when instantiating the service, there may be unexpected situations that create a sudden demand (e.g., entailing a sudden virtual CPU consumption increase) that could place SLA compliance at risk. Scaling is the solution we adopt to react to such operational events. In a general scenario, there are multiple factors affecting the scaling needs of virtual services. It depends upon the type of service and the instantiation level under execution, their latency and CPU requirements, as well

as the available resources from the underlying infrastructure. An ML-driven approach helps leveraging all the relevant monitored data to make real-time automated decisions based on the best instantiation level (IL) that should be running at each instant to fulfill the service requirements based on the context in which the service is running. In this direction, at instantiation time the 5Gr-SO, in coordination with the 5Gr-VoMS, deploys the required probes to monitor the critical metrics. After that, the corresponding Kafka topics are created to gather such metrics and feed them to the ML model. Such an ML model follows exactly the same process as explained above, i.e., the 5Gr-SO requests to the 5Gr-MLaaSP the model for solving the scaling problem for the services that are being instantiated. Once it is downloaded, together with the auxiliary code, metrics are fed to the model, which is continuously run by the SLA manager module inside the 5Gr-SO to decide what is the correct IL for the running service. If such IL does not match the current one, a scaling operation of the nested NFV-NS is triggered if convenient, when both SLA violation costs and operating costs are accounted for.

3.4 ML-driven Slice-subnet Sharing for Efficient Service Provisioning

We now present an algorithmic solution for NSSI sharing, named slice-subnet sharing algorithm (SSA), which leverages an ML-driven parameter setting and is executed at the 5Gr-VS upon a new request made by a vertical for service instance deployment. After providing an overview of slice-subnet sharing at the 5Gr-VS (Sec. 3.4.1), we detail the SSA in Sec. 3.4.2, and describe how the SSA and the ML-driven configuration of the latency classes are integrated in the 5Gr-VS in Sec. 3.4.3.

3.4.1 Slice-subnet sharing at the 5Gr-VS: An overview

To improve the efficiency of service deployment, it is of paramount importance to avoid the deployment of new, unnecessary NSSIs, when verticals request to the 5Gr-VS the deployment of service instances. Rather, already existing NSSIs should be reused whenever possible and allowed by isolation and KPI constraints, so that fewer virtual machines (VMs) are activated. On the other hand, sharing the same NSSI among services with different target latency may result in wasted computational capacity, and, thus, in higher operational costs. Indeed, when services with different latency constraints use the same NSSI, the most stringent constraint will have to be met also for the traffic

associated with the least demanding service. This entails a waste of computing resources, which decreases with the increase in similarity among the values of the services target latency. Intuitively, under a low computing load, it is more beneficial to share NSSIs, even among services with quite different target latency, so as to fully utilize the already operating computing resources. On the contrary, as the computing load increases, only services with very similar target latency should share an NSSI, to avoid the waste of resources highlighted above. However, understanding the level of slice-subnet sharing that the 5Growth provider should allow under dynamic traffic and network conditions is a hard task.

We address this challenging issue by developing a slice-subnet sharing algorithm at the 5Gr-VS, which, as detailed in the next section, allows sharing an NSSI only if *possible* and *convenient*. In particular, it determines whether reusing an NSSI is beneficial based on whether services belong to the same *latency class*. Given an interval of possible target latency values, we define as latency classes the set of non-overlapping latency ranges, covering such interval. Clearly, the higher the number of latency classes, the smaller the latency range covered by each class.

It is easy to see that the set of latency classes to be used is the critical factor that drives the sharing algorithm: the narrower the latency classes, the more similar the target latency of services that can share an NSSI; instead, the broader the classes, the wider the difference in target latency of the services that can reuse the same NSSI. As detailed in Sec. 3.4.3, given the complexity of the problem, the time-varying system load, and the diverse types of services that the system has to deal with, we envision an ML-approach to determine the best set of latency classes and feed them as input to the SSA. As a result, the SSA is an ML-driven algorithm that leverages the output of a classification model as input to make slice-subnet sharing decisions.

3.4.2 The slice-subnet sharing algorithm (SSA)

The SSA, presented in Algorithm 1, is executed at the 5Gr-VS every time a new service s has to be instantiated. Beside the latency classes configuration, the algorithm takes as input: (i) the newly requested service instance r, along with the set $\mathcal{V}_s$ of slice subnets v composing the service, and the expected service traffic load λ_r; (ii) the target latency, D_r^v, associated with each slice subnet v, its complexity factor θ_v indicating the amount of virtual CPU (vCPU) required by v to process a traffic unit; and (iii) the maximum computing capability, $\bar{\mu}_\rho$, that can be allocated to an existing NSSI, $\rho \in \mathcal{R}$. Then

Algorithm 1 Slice-subnet Sharing Algorithm (SSA)

Input: Latency classes, request $r = \langle \mathcal{V}_s, D_r^v, \lambda_r \rangle$, $\mathcal{R}$, θ_v

1: $\mathcal{V}_r \leftarrow \mathcal{V}_s$ ▷ Given service request r, initialize $\mathcal{V}_r$ to the set of slice subnets composing the service
2: $\mathcal{O} \leftarrow \emptyset$ ▷ Initialize the output set $\mathcal{O}$ to empty set
3: **for all** $v \in \mathcal{V}_r$ **do**
4: $j^v \leftarrow$ `assign_latency_class`(D_r^v) ▷ Determine the slice-subnet latency class
5: **for all** $\rho \in \mathcal{N}$ **do** ▷ For each running NSSI ρ in $\mathcal{R}$
6: **if** (ρ implements $v \in \mathcal{V}_r$) $\wedge$ $j^v = j^\rho$ **then** ▷ Check if the NSSI implements a slice subnet in $\mathcal{V}_r$ and its latency class
7: **if** $\theta_v[\Lambda(\rho) + \lambda_r] + \frac{1}{\min_{\hat{r}} D_{\hat{r}}^\rho} \leq \bar{\mu}_\rho$ **then**
8: $\mu_\rho \leftarrow \theta_v[\Lambda(\rho) + \lambda_r] + \frac{1}{\min_{\hat{r}} D_{\hat{r}}^\rho}$ ▷ Adjust capability of the NSSI
9: $\mathcal{O} \leftarrow \mathcal{O} \cup (\rho, \mu_\rho)$
10: $\mathcal{V}_r \leftarrow \mathcal{V}_r \setminus v$
11: **if** $\mathcal{V}_r = \emptyset$ **then** ▷ Check if all the slice subnets are instantiated
12: **break**
13: **if** $\mathcal{V}_r \neq \emptyset$ **then** ▷ If still slice subnets to instantiate
14: **for all** $v \in \mathcal{V}_r$ **do**
15: $\rho \leftarrow$ `create_NSSI`(v)
16: $\mu_\rho \leftarrow \theta_v \lambda_r + \frac{1}{D_v^r}$
17: $\mathcal{O} \leftarrow \mathcal{O} \cup \{\langle \rho, \mu_\rho \rangle\}$
18: **return** $\mathcal{O}$

the algorithm initializes two sets: $\mathcal{V}_r$ to set $\mathcal{V}_s$ and the SSA output, $\mathcal{O}$, to the empty set (Lines 1–2). $\mathcal{O}$ will eventually include tuples composed of two elements: the NSSI identifier and the amount of computing resources assigned to the NSSI. Once identified the latency class of each slice subnet v in $\mathcal{V}_r$ (Line 4), the algorithm looks for NSSIs already instantiated that can be reused for the deployment of service request r. In particular, the SSA determines whether any of the current NSSIs, $\rho \in \mathcal{R}$, can be shared with the new service, i.e., whether it implements a slice subnet that is included in $\mathcal{V}_r$ and falls in the same latency class as the slice subnet in $\mathcal{V}_r$ (Lines 5–6).

For each shareable NSSI, ρ, identified by the SSA, the computing capability, μ_ρ, may need to be adjusted based on the current load of the NSSI (i.e., $\Lambda(\rho)$) and the additional load associated with the newly requested service instance (i.e., λ_r), till the maximum value $\bar{\mu}_\rho$. This is done by modeling the processing time of a VM through an M/M/1 queue, as done in [69–73] (Line 8). The tuple ⟨NSSI id, computing allocation⟩, i.e., $< \rho, \mu_\rho >$, is then added to the output set (Line 9) and v is removed from the set $\mathcal{V}_r$ of slice subnets to instantiate. The sharing process ends when either all existing NSSIs

have been processed or all slice subnets have been instantiated, i.e., $\mathcal{V}_r$ becomes empty (Line 11). Finally, if some slice subnets cannot share any existing NSSI, new NSSIs are created, and the necessary computing resources are allocated (Lines 13–16).

3.4.3 ML-driven SSA parameter setting

To determine the best set of latency classes to be fed to the SSA, the 5Gr-VS interacts with the 5Gr-MLaaSP, performing the steps depicted in Fig. 3.5 and detailed below:

1. The current number of deployed NSSIs and the resource utilization metrics (e.g., vCPU consumption) are fetched continuously from the 5Gr-VS Catalog and the monitoring platform, so as to build datasets that can be used for updating the ML model;
2. the ML model is trained and stored;
3. the 5Gr-VS Arbitrator block requests the trained model to the 5Gr-MLaaSP to be used for determining the SSA input parameters when needed;
4. upon receiving a request for service instance deployment, the VSI/NSI Coordinator within the 5Gr-VS passes the request to the Arbitrator;
5. the Arbitrator retrieves the number of currently deployed NSSIs from the 5Gr-VS Catalog;
6. it then executes the trained ML model to determine the latency classes and pass them to the SSA (running at the Arbitrator);
7. the SSA determines which NSSI(s) can be shared and whether the computing resources assigned to an NSSI need to be adjusted; it then provides this information to the VSI/NSI Coordinator, which triggers the service instantiation process at the 5Gr-SO.

3.5 ML-driven Service Scaling for SLA Management and OPEX Minimization

In the previous section, slice-subnet sharing has been discussed as a use case of ML-driven provisioning of vertical slices, exploiting the interaction between 5Gr-MLaaSP

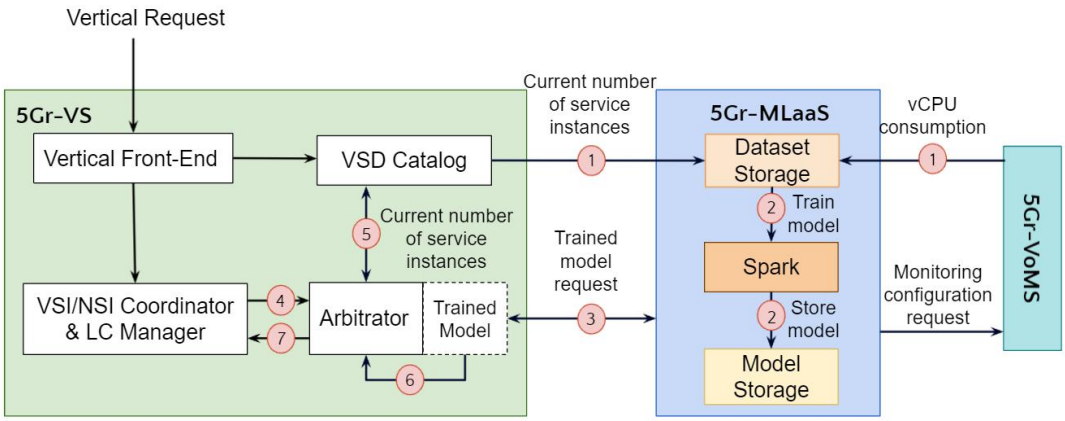


Fig. 3.5 Interaction between 5Gr-VS and 5Gr-MLaaSP, and 5Gr-VS internal structure.

and 5Gr-VS. We now focus on the 5Gr-SO, and its interaction with the 5Gr-MLaaSP, for an ML-driven SLA management and OPEX minimization. After providing in Sec. 3.5.1 an overview of the proposed approach, we detail our algorithmic solution in Sec. 3.5.2, and the design for ML-driven operations at the 5Gr-SO in Sec. 3.5.3.

3.5.1 Service scaling at the 5Gr-SO: An overview

The ultimate goal of the 5G network provider is to maximize the profit via minimizing the operational expenditure (OPEX), which is mainly due to:

- The cost of NSIs (instantiated as NFV-NSs at the 5Gr-SO), such as VNF license cost;
- The cost of operating NSI/NFV-NSs, i.e., the cost of keeping instances up and running, such as energy cost;
- The SLA violation cost, i.e., the penalty that the provider should pay if the maximum value of the target metric (in this case, latency), upon which the provider and the vertical agreed, is violated.

The challenges for minimizing OPEX are twofold. First, these costs are conflicting; to minimize the SLA violation cost, the provider should allocate sufficiently large resources to handle the peak load of the service, which would significantly increase the instantiation and operation cost. Vice versa, if the provider aimed to minimize the provisioning and operation cost by allocating the minimum resources to the services, it would incur a considerable SLA violation cost. Thus, the problem consists in finding the optimal

trade-off between the costs, i.e., the *optimum IL* that minimizes the OPEX. The second challenge is that the optimum IL depends upon the traffic load, which is time-varying, and it is a non-trivial task to understand when *scaling out/in* should be performed and to which IL, so as to avoid SLA violation costs. To address this latter point, we leverage an ML-based approach and, as detailed in Sec. 3.5.3, we design the internal architecture of the 5Gr-SO to accommodate ML-driven operations in an effective and efficient manner.

3.5.2 NFV-NS resource scaling algorithm

The resource scaling algorithm is a logic in the SLA Manager of the 5Gr-SO, which uses an ML model, already trained and maintained by 5Gr-MLaaSP, to determine the suitable IL of the NFV-NS.

Let us first introduce the mathematical expression for the aforementioned costs and the problem we address. We denote the NFV-NS latency threshold and its associated penalty, as specified in the SLA, by δ and p, respectively. Let τ be the service latency; we define the cost for violating the SLA as:

$$\sigma_{\mathsf{sla}} = p(\tau - \delta)\,. \tag{3.1}$$

The NFV-NSs are composed of a number of VNFs, but there is typically a bottleneck VNF that dominates the service latency, as also better highlighted in Sec. 3.6. Thus, to comply with the SLA, the provider needs to scale in/out the bottleneck VNF. The descriptor of the NFV-NS, specifies a set $\mathcal{L}$ of ILs where $\forall l \in \mathcal{L}$ corresponds to n_l instances of the bottleneck VNF. The cost of creating a new instance of the VNF is σ_{ins}, and the cost of using an instance per unit of time is σ_{opr}. There is no cost for termination.

Given a time period $\mathcal{T}$, let $\mathcal{N}$ be the set of NFV-NS requests arriving in this period, and $\mathcal{C}$ be the set of instances created during this period; moreover, let t_c^i and t_t^i be, respectively, the instantiation and termination times of instance i. The objective is to minimize the OPEX, defined as

$$OPEX = \sum_{r \in \mathcal{N}} \sigma_{\mathsf{sla}}^r + \sum_{i \in \mathcal{C}} \Big(\sigma_{\mathsf{ins}} + \sigma_{\mathsf{opr}}(t_t^i - t_c^i)\Big). \tag{3.2}$$

As mentioned, to achieve such a goal, the network provider needs to find the optimum IL to be applied at each time $t \in \mathcal{T}$.

To this end, we adopt the following approach. We first determine the required IL to not violate the SLA through an ML model that has been trained within the 5Gr-MLaaSP and delivered to the 5Gr-SO. The 5Gr-MLaaSP trains the ML model using a dataset where the features are (i) average CPU utilization over the active instances u_{cpu}, (ii) average memory utilization u_{ram}, (iii) service latency τ, and (iv) current IL l; the label is the target IL to satisfy the target latency. Then, in the operation phase, the following steps are performed at every period j:

1. The monitoring data u^j_{cpu}, u^j_{ram}, τ^j, and l^j are collected;
2. The exponential moving averages of the monitoring data are updated as $\bar{x} \leftarrow \alpha x^j + (1-\alpha)\bar{x}$;
3. The ML model runs using the averages $\bar{u}^j_{\mathsf{cpu}}$, $\bar{u}^j_{\mathsf{ram}}$, $\bar{\tau}$, and l^j, and provides l^{j+1}_{ml} so as to comply with the SLA (but without necessarily minimizing the OPEX);
4. The ML-Driven Service Resource Scaling (ML-RS) algorithm runs using l^{j+1}_{ml} and determines l^{j+1} that yields the best trade-off between the costs;
5. If $l^j \neq l^{j+1}$, the SLA Manager triggers the scaling operation to deploy the new IL.

Though fast switching between ILs can decrease $\sum_{r\in\mathcal{N}} \sigma^r_{\mathsf{sla}}$ and the operation cost, i.e., $\sum_{i\in\mathcal{C}} \left(\sigma_{\mathsf{opr}}(t^i_t - t^i_c)\right)$, it incurs significant instantiation cost $\sum_{i\in\mathcal{C}} \sigma_{\mathsf{ins}}$. To alleviate it, three steps are taken into account in this solution. First, the monitoring period is large enough to avoid triggering the scaling operation per request. Second, instead of instantaneous monitoring data, we use the exponential moving average as input to the model. Third, the ML-RS algorithm, presented in Algorithm 2, takes into account the SLA violation cost, instantiation cost, and operation cost to obtain the target IL.

This algorithm, in addition to the IL suggested by the ML model, l^{j+1}_{ml}, takes the SLA violation and operation costs in this monitoring interval, which are respectively denoted by Σ^j_{sla} and Σ^j_{opr}. It also takes the scaling direction SD^{j-1} suggested by the ML model in the previous interval where $SD = 0$ implies no scaling, and $SD > 0$ ($SD < 0$) means scaling out (in). The algorithm at the beginning (Line 1) finds the scaling direction suggested by the ML model in this interval. If it is "scale out" but the model has changed the direction, $SD^{j-1} \leq 0$, the SLA violation cost of this interval is saved as the accumulated SLA violation cost Σ_{sla} (Line 4). However, if the model is continuously requesting to scale out, the accumulated SLA violation cost is updated and, if it is large enough that implies it is beneficial to create new instances to decrease the violation cost,

Algorithm 2 ML-Driven Service Resource Scaling (ML-RS)

Input: $l_{\mathsf{ml}}^{j+1}, \Sigma_{\mathsf{sla}}^{j}, \Sigma_{\mathsf{opr}}^{j}, SD^{j-1}$

1: $SD^{j} \leftarrow \text{sign}(n_{l^j} - n_{l_{\mathsf{ml}}^{j+1}})$
2: **if** $SD^{j} > 0$ **then** ▷ scaling out suggestion by ML model
3: **if** $SD^{j-1} \leq 0$ **then** ▷ a new scaling out suggestion
4: $\Sigma_{\mathsf{sla}} \leftarrow \Sigma_{\mathsf{sla}}^{j}$
5: **else** ▷ ML model keeps requesting scaling out
6: $\Sigma_{\mathsf{sla}} \leftarrow \Sigma_{\mathsf{sla}} + \Sigma_{\mathsf{sla}}^{j}$
7: **if** $\Sigma_{\mathsf{sla}} > \beta\sigma_{\mathsf{ins}}$ **then** ▷ large SLA violation cost
8: $l^{j+1} \leftarrow l_{\mathsf{ml}}^{j+1}$
9: **else**
10: $l^{j+1} \leftarrow l^{j}$
11: **else if** $SD^{j} < 0$ **then** ▷ scaling in suggestion by ML
12: **if** $SD^{j-1} \geq 0$ **then** ▷ a new scaling in suggestion
13: $\Sigma_{\mathsf{opr}} \leftarrow \Sigma_{\mathsf{opr}}^{j}$
14: **else** ▷ ML model keeps requesting scaling in
15: $\Sigma_{\mathsf{opr}} \leftarrow \Sigma_{\mathsf{opr}} + \Sigma_{\mathsf{opr}}^{j}$
16: **if** $\Sigma_{\mathsf{opr}} > \gamma\sigma_{\mathsf{ins}}$ **then** ▷ large operation cost
17: $l^{j+1} \leftarrow l_{\mathsf{ml}}^{j+1}$
18: **else**
19: $l^{j+1} \leftarrow l^{j}$
20: **return** l^{j+1}

then, the suggested IL is selected (Line 10). In a similar way, when the model suggests scaling in (Line 11), the algorithm decides to scale in only if the previous suggestion was also scaling in and the operation cost is large enough.

The conditions in Lines 7 and 16 aim to let the trade-off between the costs, where $0 < \beta < 1$ and $0 < \gamma < 1$ are tunable parameters, depend upon the dynamics of the traffic load. The higher the traffic load dynamic, the larger these parameters to avoid too many instantiations and terminations by small changes in the traffic load. However, if traffic load changes smoothly, the value of the parameters can be small to minimize the SLA violation cost and the operation cost.

3.5.3 ML-driven 5Gr-SO design

In accordance with the discussions in Sec. 3.2, when instantiating an NFV-NS, the workflow describing the interaction between the 5Gr-SO and 5Gr-MLaaSP follows a model in which the MTLF and AnLF are located in different architectural entities. More specifically, the 5Gr-MLaaSP trains the model based on the previously uploaded

dataset and makes it available to external entities. The 5Gr-SO downloads the model and continuously runs it during the lifetime of the service for which scaling decisions are needed.

This high-level architectural idea requires multiple steps to be deployed, which are mainly related to the configuration of a complete data engineering pipeline, since monitoring job configuration and data collection until execution of the management and orchestration procedure based on the decision made by the model. Thus, once the complete pipeline is in place and integrated with the 5Gr-SO operation, the closed-loop automated network management decisions can be made, since relevant metrics are gathered and ingested by the ML model, which infers the best possible IL, which in turn generates the corresponding scaling procedure, when needed.

More specifically, the workflow is as follows (see [74] for further details). At the end of the NFV-NS instantiation process, the 5Gr-SO requests the 5Gr-VoMS to configure the monitoring jobs for the service, as specified in its network service descriptor (NSD). If the NSD also embeds an AI/ML information element (IE) for solving a specific problem (in this case, scaling), the SLA manager inside the 5Gr-SO detects it and starts the configuration of the data engineering pipeline.

First, a dedicated Kafka topic is created. Second, data scrapers are also created together with the 5Gr-VoMS that filter the relevant information to be fed to the scaling topic. After that, the model is downloaded from the 5Gr-MLaaSP/MTLF, and the SLA manager creates an Apache Spark streaming job in charge of feeding the ML model with real-time information (including the current IL) for scaling decision making (AnLF). If the inferred IL by the model is different from the current one, a scaling procedure requesting the change of the IL is triggered. Furthermore, during scaling procedure execution, the model inference process is stopped to avoid unexpected transient effects and is created again for the scaled service at the end of the scaling process.

It is worth mentioning that in the current system design, the model does not need to be changed after a scaling operation [2]. Indeed, all possible states (i.e., ILs) of the service are considered during the training phase, and the monitored metric values related to different instances are averaged during inference so that a single model can be used regardless of the current IL[3]. The current approach has the advantage of scaling very well with the number of possible ILs. On the contrary, handling multiple ML models as

[2]It is assumed that the service implementation provides the corresponding logic (e.g., load-balancing) to effectively support the scaling operation.

[3]Note, however, that the current IL is an input to the model during inference.

the value of service IL changes would require the download from the MLaaSP of as many trained ML models as the number of ILs allowed for the service, with only one specific model being used at a time according to the current IL value.

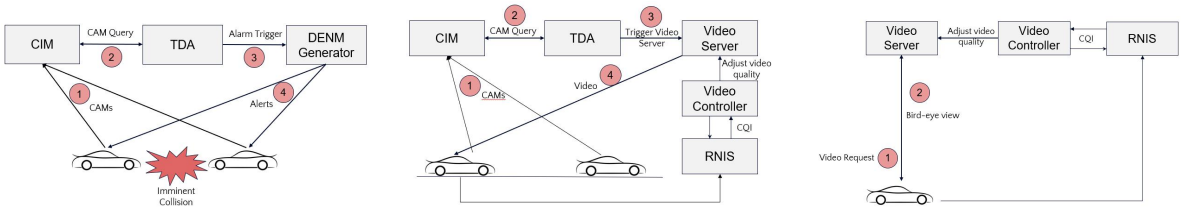


Fig. 3.6 Service structure: collision detection (left), see through (center), and destination-aware bird-eye view (right).

3.6 Automotive Services

To validate our approach to network slice provisioning and management, we take as reference services three relevant use cases in the automotive domain, namely, (i) vehicle collision detection at intersections (CD), (ii) see-through (ST), and (iii) destination-aware bird-eye view (DaBEV). The first two are representative of safety services, while the third is an example of convenience services for vehicular users [75]. The three services are depicted in Fig. 3.6 and detailed below.

Collision detection. The CD service detects vehicles on collision course and sends them an alert message. It exploits two types of messages defined by ETSI: Cooperative Awareness Messages (CAMs), which are periodically transmitted by vehicles and carry the position, speed, acceleration, and heading of the sender, and the Decentralized Environmental Notification Messages (DENMs), which are delivered to vehicles to notify them about events or dangerous situations. The service includes the following VNFs:

the *Cooperative Infrastructure Manager (CIM)*, which receives, decodes, and stores CAMs sent by the vehicles within the area covered by the CD service (see step 1 in Fig. 3.6(left));

the Trajectory and Detection Algorithm (TDA), which queries the CIMs for new CAMs (step 2) and runs a trajectory-based algorithm (i.e., the one presented in [23] and evaluated in [76]), to detect pairs of vehicles on collision course;

the *DENM Generator*, which, triggered by the TDA (step 3), encodes and sends (unicast) alarm messages to the vehicles detected to be on collision course (step 4), so that, e.g., the emergency braking system aboard vehicles can be activated.

Upon requesting a CD service instance, the vertical specifies the geographical area (e.g., set of intersections) that has to be covered and the estimated number of users to serve. Also, since the CD should be combined with other collision avoidance mechanisms based on physical sensors aboard the vehicles, the maximum target CD latency specified by the vertical is set to 20 ms [20]. Notice that the dominant contribution to the service processing time is due to the TDA, which is thus the botteneck VNF of the CD service.

See-through. It provides a real-time view of the surrounding area of the requester, to avoid collisions when an overtaking manoeuvre is executed. The service includes:

the *CIM* and the *TDA*, which, as above, store the vehicle's CAMs (step 1 in Fig. 3.6(center)) and run the trajectory-based algorithm (step 2), respectively. Upon detecting pairs of vehicles that would be on collision course if one of them moved to the left lane, the TDA triggers the Video Server (step 3);

the *Video Server*, which fetches the video from the smart-city cameras providing the best view of the surroundings of the tagged vehicles, and encodes and transmits video frames to the vehicles (step 4);

the *Video Controller*, which reports periodically to the video server the video quality to be used, based on the channel quality experienced by the vehicles;

the *Radio Network Information Service (RNIS)*, which is co-located with the radio point of access and exposes radio contextual information, namely, the Channel Quality Indicator (CQI); unlike the above functions, it is implemented as a physical network function (PNF).

The quality of the video should be high enough to guarantee at least a rate of 30 frames per second (fps), i.e., a good representation of the position and movements of the involved vehicles, sacrificing the resolution if necessary and maintaining the maximum latency below 150 ms [77]. In the ST service, it is the Video Server that exhibits the highest level of complexity, followed by the TDA.

Destination-aware bird-eye view. It provides a real-time view of the area between the requester and its intended destination, leveraging smart-city cameras located along the vehicle's route. It includes:

the *Video Server*, receiving the vehicle request (step 1), and fetching the smart-city videos, encoding them, and transmitting the bird-eye view to the requester (step 2);

the *Video Controller*, providing, as above, the Video Server with the video quality to be used;

the *RNIS*, providing the video controller with the vehicles' CQI; as before, it is implemented as a PNF.

The ideal output should be a high resolution (e.g., 1920×1080) and low fps (e.g., 7 fps), since this is a convenience service and does not have to provide the user with a highly dynamic context. For each received request, the processing load on the Video Server (which is the dominant VNF here) depends upon the number of input video cameras from which a stream is required to represent the bird-eye view of the area of interest. The maximum service latency is set to 1 s.

Given the above services, we consider that CIM and TDA VNFs form the *Trajectory Detection Slice Subnet (TDSS)*, while Video Server and Video Controller compose the *Adaptive Video Slice Subnet (AVSS)*. Finally, we remark that all the above services require the use of a radio access network for vehicle-to-infrastructure connectivity.

3.7 Validation and Performance Evaluation

In this section, we first detail the real-world scenario we used to build our training datasets, and to derive large-scale simulation results (Sec. 3.7.1). Then, through our experimental testbed, we validate the interaction between the 5Growth entities (5Gr-VS and 5Gr-SO) and the 5Gr-MLaaSP, as well as the proposed ML-driven algorithms for slice-subnet sharing at the 5Gr-VS and run-time scaling at the 5Gr-SO, in a small-scale scenario (Sec. 3.7.2). Finally, we show the performance results of both solutions, via simulation in the aforementioned large-scale, real-world scenario (Sec. 3.7.3).

3.7.1 Large-scale reference scenario, datasets, and ML model

We consider an 11 km^2 area of the city of Turin, Italy, comprising 24 major crossroads and a total of 112-km road stretches. Vehicle mobility is simulated thanks to the Simulation of Urban Mobility (SUMO) and the Turin SUMO traffic (TuST) trace [78]. As the latter refers to 24-hour traffic in a weekday, we select 6 different time slots that are representative of different vehicle densities during the day; the 14 metropolitan zones included in the trace are depicted in Fig. 3.7. Also, we consider the Metro Node of the cellular network as the point of presence (PoP) where service instances should be deployed. The number of vehicles travelling on the whole area varies from 2,110 in the 3am–4am time slot to 32,117 in the 6pm–7pm time slot.

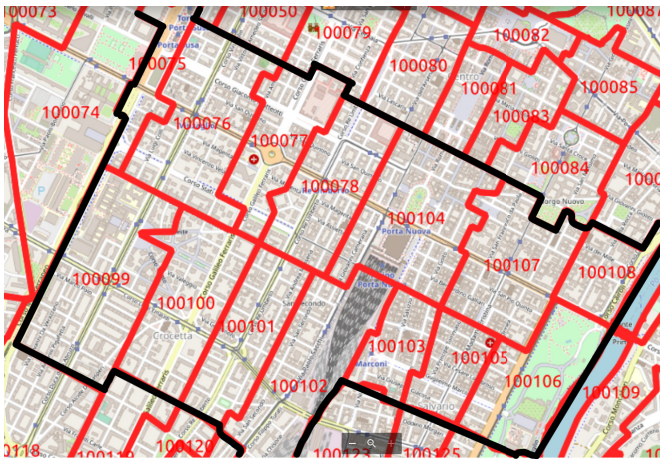


Fig. 3.7 Large-scale scenario: Turin metropolitan area.

In the scenario under study, a different number of CD instances are requested by the automotive vertical, depending upon the vehicle density in the considered time slots, and the CD service is provided to all vehicles entering an intersection covered by such service. Tab. 3.1 reports the relation between the average number of vehicles per km in the overall geographical area, and the number of CD instances that have to be deployed. Unless otherwise specified, upon entering the area, 50% of the vehicles request the ST service, while 30% request the DaBEV service, and the arrival of requests for these services is modeled as a Poisson process with a rate value set as in Tab. 3.1. The lifetime of the ST and DaBEV is set according to the time taken by the vehicles to travel across the urban area. Finally, we consider that each VNF is implemented in a VM, and each active VM can use up to 8 vCPUs.

Table 3.1 CD instances in the considered area

Veh. density [veh./km]	No. of CD instances	Vehicle rate [veh./s]
1.10	1	0.59
3.04	3	2.24
8.36	8	6.79
11.04	9	7.11
12.11	10	8.09
15.38	11	8.92

For slice-subnet sharing, each simulation is performed for a given latency class configuration, among the many that we have considered, and vertical's request arrival rate. Such data is then labeled so as to identify the best configuration with respect to the CPU consumption. For service scaling, we experimentally derived three datasets, one for

each of the considered automotive services. The testbed has been configured to run the CIM, TDA, Video Server, and Video Controller in the machine accommodating the Edge Host, which also runs a srsRAN virtualized LTE eNB. A second machine, which hosts the UE, connects to the Edge Host through the LTE link and acts as a client generating requests to the offered automotive services. The rate of such requests varies over time to emulate the behaviour of multiple users. The CPU consumption is monitored at runtime, while the end-to-end latency is computed offline by post-processing the experiment logs. The final datasets are built adding as label the IL that can successfully handle the computational load associated with each rate of user requests. We considered that each service has two possibile ILs, hence the dataset (and the NSD) includes two labels: "small IL" and "big IL". To label the datasets, we set the target processing times for the considered services to values that, based on our testbed experiments, allow meeting the maximum latency reported in Sec. 3.6.

To perform the prediction of the best parameter values to use within the SSA and the the ML-RS algorithm, we employed a Random Forest (RF) Classifier, which leverages multiple decision trees at training time when predicting the correct label to assign to unseen data. The output prediction of each tree is taken into consideration when calculating the final label to assign via a majority vote. This method, called also bagging or bootstrap aggregation, reduces the variance by decorrelating the output of the different trees by choosing a subset of the predictors considered for each tree [79].

The trained ML models we use for determining the latency classes to feed to the SSA and the IL to feed to the ML-RS algorithm are generated within the 5Gr-MLaaSP using Spark, which leverages a pipeline approach, applying different transformations to the considered dataset (i.e., normalization) combining them into a single workflow. A hyperparameter search is performed on the model in order to obtain the best possible result. The considered hyperparameters are: the maximum depth of the trees, the minimum number of samples required to split an internal node, the minimum number of samples per leaf node, the number of decorrelated trees generated during the training phase, and the split criterion in the trees generation, whether is Gini index or entropy. The values of such hyperparameters, along with the suitable ones, are reported in Tab. 3.2. The test accuracy obtained using the best model hyperparameters for the slice-subnet sharing model is equal to 0.9835, and the related confidence intervals are presented in Tab. 3.3. Similarly, Tab.3.4 shows the results obtained training the service scaling ML models.

Table 3.2 Values of the hyperparameters of the RF model

Parameter	**Tested values**	**Best value**
Number of estimators	$\{100, 200, 300\}$	100
Maximum depth	$[1, 10]$	9
Minimum sample per split	$[2, 10]$	2
Minimum samples per leaf	$[1, 5]$	2

Table 3.3 RF model for slice-subnet sharing: confidence intervals

Confidence level	**Confidence interval**
90%	[0.98331, 0.98358]
95%	[0.98331, 0.98364]
98%	[0.98332, 0.98365]
99%	[0.98332, 0.98545]

To find the best combination of hyperparameters, the model has been validated using a k-fold cross validation method, which divides the training dataset in k groups and generates k evaluation scores by training the model on $k-1$ folds and testing it on the remaining one. The final result is given by the mean of the calculated scores. We chose $k = 10$ since it provides an estimate with low bias and modest variance.

The training phase, using a 26,000-sample dataset and including the very time-consuming search for the best model hyperparameters configuration, lasted 45 minutes using a machine with a 2.2 GHz Intel i7-8750H processor and a 16 GB DDR4 RAM.

3.7.2 In-testbed validation

We now present the validation of the interaction between the 5Gr-MLaaSP and the 5Gr-entities, as well as of the ML-driven decision-making processes at the 5Gr-VS and the 5Gr-SO.

Testbed results for slice-subnet sharing at the 5Gr-VS. The in-testbed validation at the 5Gr-VS demonstrates the feasibility of the proposed architecture and of the ML-based slice-subnet sharing algorithm presented in 3.4.1. Further, it shows that the impact on the life-cycle management actions, in terms of introduced delays, is negligible. For this, we deployed the 5Growth MANO platform as per the architecture depicted in Fig. 3.2. This platform is used to manage the collision detection (CD) and see-through (ST) services described in Sec. 3.6. The 5Gr-VS of the experimental setup implements the architectural blocks and the interaction introduced in Sec. 3.4.3. It is

Table 3.4 Accuracy of the service scaling models

Service	Model accuracy	Confidence interval (95% CL)
CD	0.9918	[0.9914, 0.9921]
DaBEV	0.9912	[0.9906, 0.9919]
ST	0.9953	[0.9949, 0.9958]

configured with a default arbitration policy, which determines the trained ML to be retrieved from the 5Gr-MLaaSP, it runs the model to determine the latency classes configuration, and executes the SSA for slice-subnet sharing.

The tests performed to gather the results consisted in repeating ten times the instantiation of the CD and the ST services. During the instantiation of the services at the 5Gr-VS, we measure the statistical distribution of the time required to execute the ML-based model and the SSA. Each run of the tests starts with the instantiation of the CD service. During this operation, the 5Gr-VS maps the service to one NSI containing two NSSIs: NSSI_A implementing the TDSS logic (i.e., TDA and CIM), and NSSI_B implementing the DENM Generator logic of Fig. 3.6. Upon being executed, the SSA determines that all the NSI and NSSIs of the CD service are to be provisioned since there are no candidate slice subnets to be shared. Thus, the 5Gr-VS provisions the NSI and NSSIs, and requests the corresponding network services to the 5Gr-SO.

Once the CD service has been deployed, the test proceeds with the instantiation of the ST service. In this case the service is mapped to one NSI, and two NSSIs: NSSI_A as before, and NSSI_C containing the AVSS (i.e., Video Server and Video Controller). Given the latency constraints of the new service, in this case the SSA determines that the NSSI_A can be re-used and the scaling actions to be performed to accommodate the additional traffic demand.

Fig. 3.8 presents the results for the CD and ST services. The depicted boxplots cover the experienced maximum, minimum, average, median, 20th and 80th percentile values across the ten performed repetitions. We note that the same boxplot representation is going to be used used in the boxplots graphs presented in the rest of this section. Fig. 3.8 shows that the delays introduced by the ML-driven latency class configuration and the SSA are approximately equal to 2.5 s for both considered cases. This proves both the feasibility of our approach for ML-driven service provisioning, and the reduced impact in terms of overall service instantiation time.

Finally, we remark that, although the results presented in Fig. 3.8 do not account for the time required to train the ML model within the 5Gr-MLaaSP, our experiments

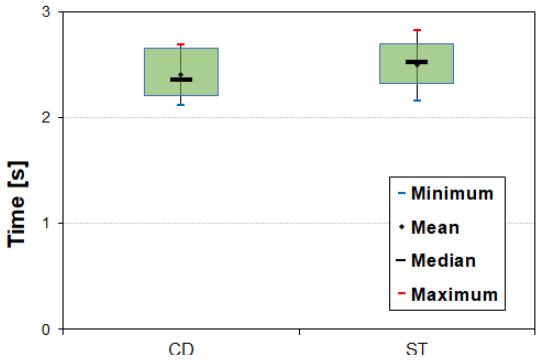


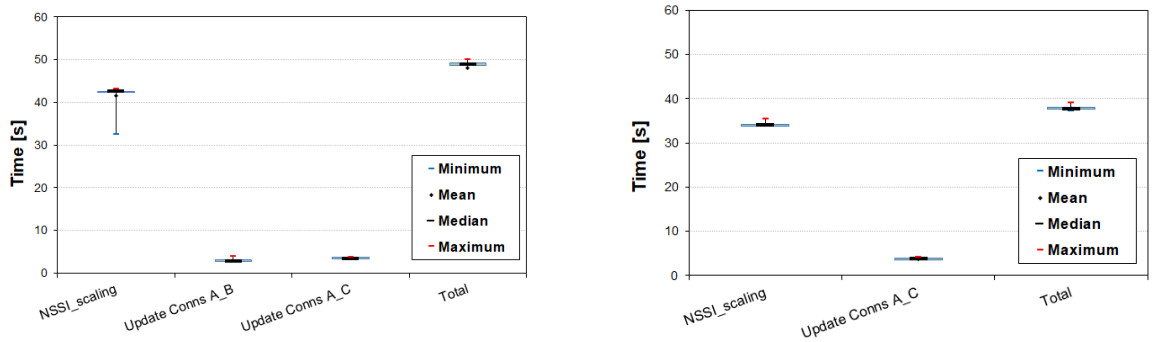
Fig. 3.8 Time (in seconds) of ML-driven service provisioning for the collision detection and see-through services.

showed that the model training takes about 25 s, which is acceptable, even if the model needed to be retrained with fresher data collected from the monitoring platform at every service instantiation.

Testbed results for resource scaling at the 5Gr-SO. As before, we use a 5Growth MANO platform instance as depicted in Fig.3.2. The platform controls an NFVI infrastructure composed of three NFVI-PoPs, which are managed by dedicated instances of a Virtual Infrastructure Manager (VIM), implemented with Openstack software. A transport network, emulated with GNS3 software, interconnects these NFVI-PoPs. The transport network has five packet-switches following a ring topology of four elements with an additional packet switch in the middle to provide additional path redundancy. These packet switches are controlled by an instance of an ONOS SDN controller. In these experiments, we considered a distributed NFVI-PoP scenario modeling an edge scenario, where resources are limited and may not be enough for the deployment of a whole NSI. Moreover, this kind of distributed deployment allows showing the impact of updating the interconnections between the different NSSIs that are part of an NSI when a scaling operation is performed. It is worth mentioning that this resource-driven handling of the interconnections between NSSIs upon scaling parts of an NSI has been scarcely tackled in the literature (further details can be found in [80]).

Since the focus of the experimental evaluation is on measuring the scaling performance of the system, the deployment of NSIs is as follows. Initially, the 5Gr-VS requests to the 5Gr-SO the deployment of the CD service as an NSI[4] (NSI_1) is composed of two NSSIs (NSSI_A and NSSI_B), defined as above. Then, the 5Gr-VS requests to the 5Gr-SO the deployment of the ST service (NSI_2) and the 5Gr-VS determines the sharing of the

[4]As mentioned in Sec. 3.3, the NSIs are requested by the 5Gr-VS to the 5Gr-SO in the form of composite NFV-NSs, and its nested NFV-NSs become the NSSIs of the service.



(a) NSSI_A scale out operation

(b) NSSI_C scale out operation

Fig. 3.9 ML-driven scale-out of an NSI shared deployment.

NSSI_A, so that the 5Gr-SO only needs to deploy the remaining AVSS (referred to as NSSI_C) to complete the instantiation of the ST service. In this experiment, each NSSI is deployed in a different NFVI-PoP. During the instantiation of the different NSIs, in particular of NSSI_A and NSSI_C, the 5Gr-SO contacts the 5Gr-MLaaSP to download the required ML model to drive scaling operations. The ML-based scaling decision is driven by the performance of the TDA and Video Server VNF instances present in NSSI_A and NSSI_C, respectively. The following graphs present the experienced time to perform ML-driven scaling operations in the above deployment. Each experiment has been repeated ten times.

Fig. 3.9 shows the statistical distribution of the time required to scale out NSSI_A and NSSI_C, and the implications of the different performed operations in the overall deployment. As the traffic density increases, the ML model for NSSI_A determines that a new IL including a new instance of the TDA VNF is required to fulfill the service requirements. Consequently, more requests are done to the Video Server VNF, and the ML model for NSSI_C also determines that a new IL including a new instance of this VNF is required.

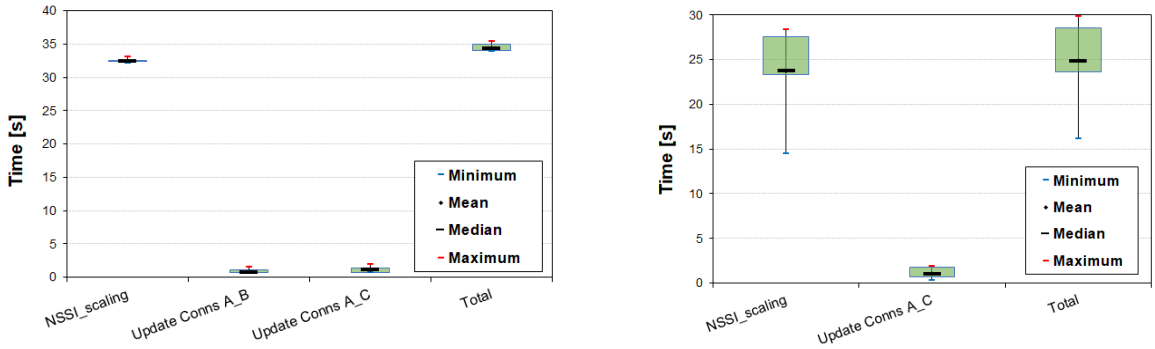
The scale out of NSSI_A produces changes in the overall deployment affecting also NSSI_B and NSSI_C because both deployed NSIs share the NSSI_A instance. These changes take 47.98 s on average. Out of this time, 86.7% is devoted to adding the new TDA VNF instance. This time also includes the operations carried out by the 5Gr-SO due to the ML-driven scaling procedure followed, namely data-engineering pipeline configurations actions, as explained in the last steps of the workflow presented in Sec. 3.5.3. In total, these operations represent a 5.88% of the average NSSI_A scaling time required to add the new TDA VNF instance. The most time-consuming operation in

this set of actions associated with the ML-driven scaling process is the termination of the inference job, running as an Apache Spark job (around 2.5 s on average). This high time is experienced because the inference job is under execution when the termination request arrives prior to proceeding with the scaling operation, as observed in [74]. The rationale of stopping the inference job during the NSSI being scaled is to avoid overloading the 5Gr-SO with wrong scaling decisions issued while NSSI_A is updated.

The remaining 13.3% of the overall deployment scaling time is devoted to connecting this new TDA VNF instance with the VNFs deployed to associated NSSIs, namely, NSSI_B and NSSI_C. The average time required to update the interconnections with NSSI_C is larger (3.431 s vs. 2.960 s) than the one with NSSI_B because NSSI_C includes two VNFs.

When scaling out the Video Server VNF of the NSSI_C, the overall deployment scaling time (37.94 s on average) is lower due to two facts. On one hand, the scaling of NSSI_C only requires the update of the interconnections with a single NSSI, i.e., the associated NSSI_A. On the other hand, there is a difference in performance in the hardware running in the NFVI-PoPs where NSSI_A and NSSI_C are deployed. In these experiments, all VNF descriptors considered the same characteristics in terms of resources (i.e., CPU, RAM, and storage). While it takes 41.59 s on average to scale just the NSSI_A (i.e., add the new instance of the TDA VNF), the same operation for NSSI_C (i.e., adding a new VNF instance) takes 34.14 s on average. In this case, the impact of ML-related operations follows the same trends as before and represents the 7.19% of the NSSI_C scaling time. This is a higher percentage than before because the overall deployment scaling time is lower for this case, but the time required by the ML operations is approximately the same in both cases.

Fig. 3.10 shows the statistical distribution of the experienced time when scale-in operations occur, e.g., due to a decrease in the traffic density, the ML model determines that NSSI_C and NSSI_A can return to their initial IL (24.91 s and 34.48 s, respectively). The overall observed trends are the same as with the scale-out operations explained before. Regarding the impact of ML operations (10.22% for NSSI_C and 7.33% for NSSI_A), the most time-consuming operations is the termination of the inference job before the deletion of the corresponding VNFs required by the new target IL. In this case, the experienced scaling time is lower mainly because the time to deallocate resources (VMs associated with the VNFs and transport network connectivity services) is smaller than to allocate them. We also observe the difference in performance between the hardware of the different NFVI-PoPs hosting the VNFs of the different NSSIs.



(a) NSSI_A scale in operation

(b) NSSI_C scale in operation

Fig. 3.10 ML-driven scale-in of an NSI shared deployment.

In summary, for the deployed services under evaluation, ML-related operations have taken, on average, roughly from 5% to 10% of the total scaling time. Since these operations take approximately the same time independently from the service, the higher shares are obtained when scaling operations take less time in absolute terms (e.g., allocation/deallocation of VMs for scale out/in operations or less inter-nested or inter-PoP connections to be established).

3.7.3 Numerical results in a large-scale scenario

We now consider the real-world, large-scale scenario described in Sec. 3.7.1 and assess the performance of the proposed solutions via simulation.

Simulation results for slice-subnet sharing at the 5Gr-VS. Fig. 3.11(top) shows the performance gain in percentage with respect to the case where no NSSI sharing is applied. Specifically, the results are presented in terms of savings in number of vCPU utilization and in number of VMs instantiated, and for different values of vehicle arrival rate. As expected, the gain decreases as the workload grows, however it is interesting to notice that NSSI sharing allows for substantial saving of computing resources. The fluctuation of the number of active VMs is due to the fact that the best latency class configuration selected through the ML approach aims at the minimization of the vCPU consumption, which may not be the same as the configuration that would minimize the number of used VMs.

Under the same scenario, Fig. 3.11(middle) shows the average number of instances that share the same NSSI. Interestingly, we notice that, as the vehicle arrival rate increases

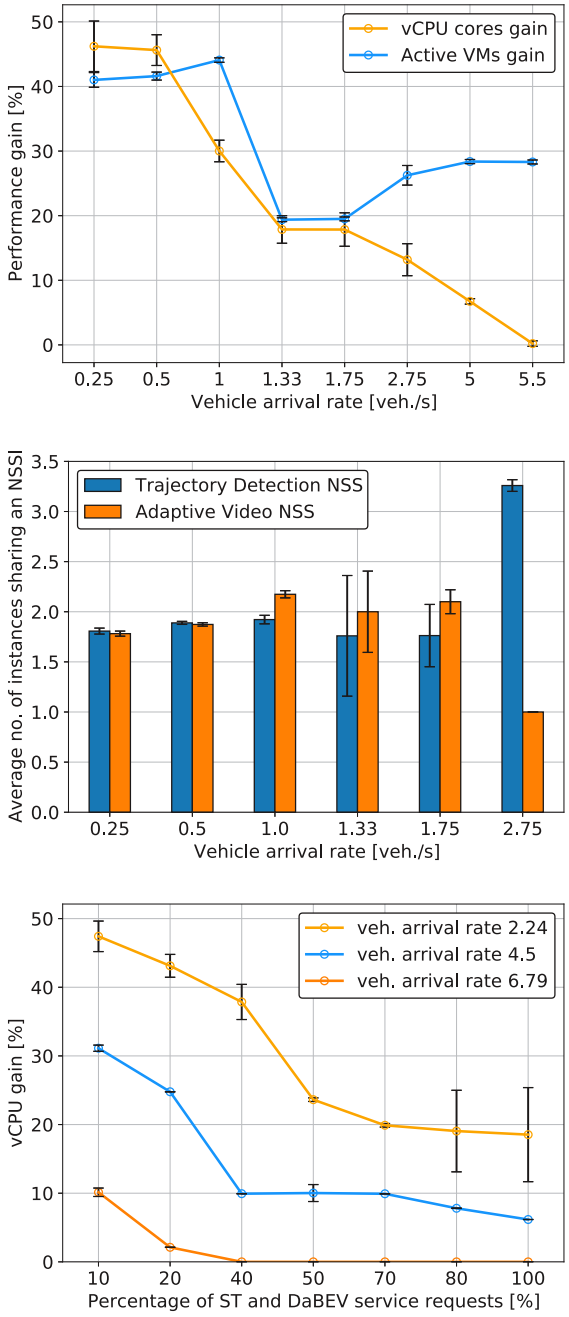


Fig. 3.11 Simulation results on slice-subnet sharing vs. vehicle arrival rate. Top: gain in terms of vCPU consumption and number of active VMs, compared to no sharing (ST and DaBEV set to 50% and 30%, resp.); Middle: average no. of instances sharing the same NSSI (ST and DaBEV set to 50% and 30%, resp.); Bottom: vCPU consumption gain vs. percentage of ST and DaBEV requests, relatively to no sharing and for different vehicle arrival rates. Each plot reports also the 95% confidence interval over 100 simulations (in some cases it is so small to be scarcely visible).

(hence the number of ST and DaBEV requests grows), the number of instances sharing the same Trajectory Detection slice subnet decreases more slowly than in the case of the Adaptive Video slice subnet. Indeed, ST and CD have a much more similar target latency than ST and DaBEV, thus making the benefit of the latter two services sharing a NSSI drop faster.

Next, in Fig. 3.11(bottom) we investigate the impact of the percentage of ST and DaBEV requests for different values of vehicle arrival rates. The plot confirms that the vCPU gain with respect to the case of no NSSI sharing is substantial for lower rates and vanishes in the highest rate scenario, as the best latency class configuration tends to place all the NSSIs in different latency classes.

In conclusion, the above results demonstrate that the proposed ML-drive slice-subnet sharing for service provisioning is highly beneficial, with up to 40% reduction of vCPU in light load conditions, and a reduction of the number of instantiated VMs that ranges between 20% and 40%.

Simulation results for resource scaling at the 5Gr-SO. We now evaluate the performance of the SLA management solution proposed for OPEX minimization. Three different strategies are compared: the L-INS and H-INS strategies, where NFV-NSs are instantiated using the "small IL" and "big IL" instantiation levels (resp.) and such ILs are never changed, and our proposed ML-RS solution. Note that, in this case, "small IL" and "big IL" correspond to 2 and 5 (resp.) instances of the bottleneck VNFs of the NFV-NSs. Moreover, according to the target values in Sec. 3.6 and our experimental implementation of the services, we set the latency thresholds, δ, for the nested NFV-NSs to: 5 ms, for the TDSS, 120 ms for the ST AVSS, and 950 ms for the DaBEV AVSS. The SLA violation penalty associated with the latency thresholds is $p = 2$ unit per second. Moreover, we set $\sigma_{\mathsf{ins}} = 1,000$ units, and $\sigma_{\mathsf{opr}} = 0.12$ unit/s.

To evaluate the efficiency of the strategies, their performance as a function of traffic load is compared in Fig. 3.12 where the SLA violation cost, the service provisioning cost, which is the sum of instantiation and operation costs, and the total OPEX cost per strategy are depicted. These results, obtained by averaging over 20 runs, are presented as functions of parameter ℓ, by which the arrival request rate is multiplied. Also, the costs are computed over a 24-hour period in Fig. 3.12(a)-(b), and over a 48-hour period in Fig. 3.12-(c).

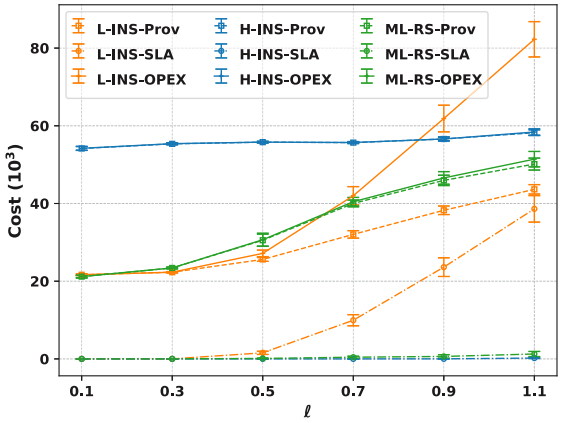
We observe that, under lightly loaded conditions, i.e., $\ell \leq 0.3$, all strategies comply with the SLA, so there is no SLA violation cost. However, because of over-provisioning, the OPEX of H-INS is significant while L-INS and ML-RS use the minimum number of

instances and have the same OPEX. By increasing ℓ, the SLA violation cost of L-INS grows exponentially but the proposed solution can maintain a negligible SLA violation by efficient resource scaling that minimizes the OPEX. More specifically, when $0.3 < \ell < 1.0$, ML-RS yields the same SLA violation cost as the H-INS, but with a considerable lower OPEX due to the lower provisioning cost via the appropriate resource scaling. In the highly loaded conditions where $\ell > 1$, even the "big IL" is not sufficient to handle the offered load, i.e., the H-INS-SLA > 0. In this case, ML-RS at the beginning scales out the service and never scales it in, which makes H-INS and our solution have similar performance.

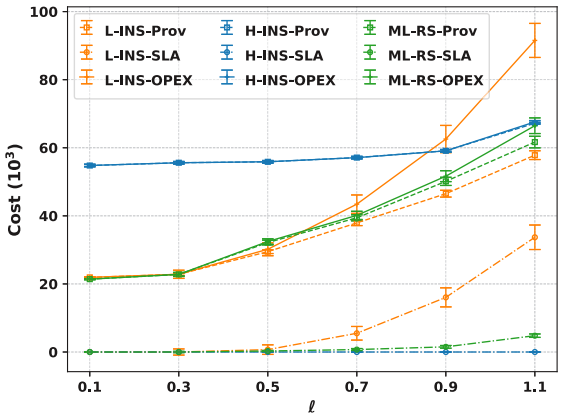
In summary, these results show how the ML-based approach is capable of adapting the service setup (in this case, IL) to improve the OPEX for the 5Growth provider by finding an appropriate trade-off between SLA compliance and service provisioning cost. Remarkably, the OPEX is halved in comparison to the overprovisioning strategy in lightly loaded conditions, while the SLA violation is negligible in highly loaded conditions.

3.8 Related Work

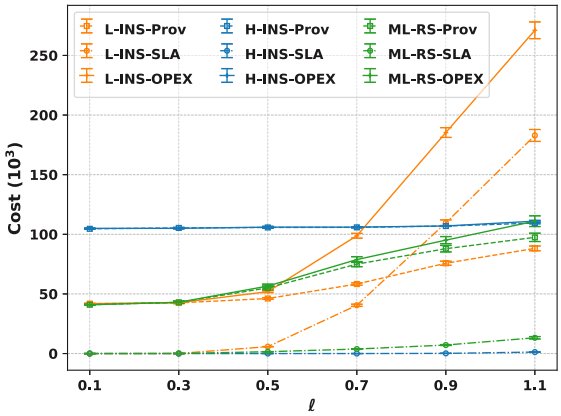
In new-generation cellular networks, services are provided through the provisioning of logical networks according to the well-known network slicing paradigm, on which useful surveys can be found in [81–84], [85]. In this context, the MANO architecture [66] is often used for the life cycle management and the runtime operation of network slices. Beside the 5Growth project [58], also 5GCity [86] proposes an orchestration platform, while 5G-MoNArch [87] aims at developing an experiential network intelligence that combines AI/ML with network orchestration and management. Additionally, there are similarities between our proposed architecture and that of the MATILDA project [88]. The emphasis of this chapter, however, is on the realization of the MLaaS concept through the creation of a separate building block (5Gr-MLaaSP) providing an ML catalog that can be consumed by any other block of the architecture. In this sense, the 5Gr-MLaaSP can serve the needs of a wide variety of current problems related to service management, and those that will appear. On the other hand, learning processes in the MATILDA architecture are bound to the intelligent service orchestrator, and, hence, are integrated with the logic of this block, which focuses on specific problems. We believe that an external 5Gr-MLaaSP offers a generic and future-proof architectural solution, which also follows the recommendations of such SDOs as 3GPP or O-RAN [61]. It is also worth mentioning that, as in the case of the 5Growth architecture, the MATILDA project



(a) Collision Detection service in a 24 hour period



(b) Bird Eye service in a 24 hour period



(c) Bird Eye service in a 48 hour period

Fig. 3.12 Provisioning cost, SLA violation cost, and OPEX with respect to the scale of the arrival rate of the service requests. Each plot reports also the 95% confidence interval over 20 simulations.

[89, 90] envisions a separation of concerns between the vertical application domain and the network operator domain where slices are deployed.

ML techniques are also leveraged in [91] for radio resource scheduling and management, and in [92] for resource orchestration and an optimal usage of physical resources. In [93], deep reinforcement learning is investigated for network slice reconfiguration, with the aim to minimize long-term resource consumption. Particularly relevant to our work are also the studies in [94, 95]. Indeed, [94] proposes a hierarchical orchestration architecture to deal with multi-domain scenarios, as well as a service auto-scaling algorithm. The latter foresees both a ML-driven proactive provisioning technique and a reactive resource adaptation, so that the service target latency can be met in spite of the time-varying traffic demand. [95], instead, focuses on a negotiation game between verticals and network providers, for service chains auto-scaling. Interestingly, [95] proposes an ML-driven scaling decision process at the service orchestrator, which however does not account for the trade-off between the cost of resources and that of SLA violations and is evaluated through numerical tests only. Such a trade-off is instead investigated in [96], where an ML-based scaling management, specifically designed for Kubernetes edge clusters, is presented.

Several works have adopted algorithmic approaches to resource allocation and service admission control. Among these studies, [97] proposes elastic NFV resource allocation according to predefined resource pressure thresholds, while [98] predicts scaling events feeding application-level metrics to a neural network, which however poses privacy concerns. Also, [99] leverages deep learning models to reliably and dynamically orchestrate end-to-end slices, and allocate radio, computing, and storage resources. Others focus on admission control aspects, exploiting ML techniques [100] or auction mechanisms and game theory [101, 102].

It is worth underlying that all the above studies have a different scope from that of slice-subnet sharing that we consider. Indeed, they focus on resource allocation or sharing among different network slices, instead of network slice-subnet sharing among different services. Further, our work provides a complete, thorough evaluation of the proposed solutions, via both large-scale simulations and experimental tests carried out through our testbed. Importantly, the latter ones also demonstrate the interaction and functional synergy between the 5G architecture entities, thus validating our system design. Finally, we mention that a preliminary version of the ML-driven service scaling presented in this work has been presented and demonstrated in our conference [74] and demo [103] papers.

3.9 Conclusions

We tackled the problem of fully-automated service provisioning and management in a virtualized 5G network platform. We proposed a system design that allows the 5Gr-VS and 5Gr-SO architectural layers to effectively interact with the MLaaS platform we created, and to leverage ML-driven decision-making processes. We also defined algorithmic solutions for ML-driven slice-subnet sharing and run-time service scaling, under dynamic traffic conditions. Through in-testbed validation, we demonstrated the feasibility of our approach, the designed functional synergy between the 5G architecture entities, as well as the limited time overhead introduced by the ML framework. Further, simulation results in a large-scale, real-world scenario showed remarkable savings in operational costs, e.g., up to 40% reduction in vCPU consumption and up to 30% reduction in the OPEX.

This section concludes the first part of this thesis, where we addressed the problem of SLAs satisfaction in 5G networks through centralized management and orchestration. We showed the clear benefits that this approach has to offer, first using a threshold-based service scaling mechanism, then integrating ML in the 5Growth platform for a more advanced management capability. The centralized management is ideal for high-level and coarse-grained decisions that can potentially operate on the entire network, but it is not suitable for micromanagement of edge network segments, if fine-grained decisions are considered. In this case, not only scalability with the number of nodes would be difficult, but it would be impossible to manage the edge node with low latency control loops. Starting from the next chapter, we will see the complementary approach of deploying decision-making at the edge, colocated with the network services and the user applications over which control actions are taken. Specifically, we will first present CAREM, a reinforcement learning framework able to dynamically allocate radio resources in heterogeneous vRANs.

Chapter 4

A Context-aware Radio Resource Management in Heterogeneous Virtual RANs

New-generation wireless networks are designed to support a wide range of services with diverse key performance indicators (KPIs) requirements. A fundamental component of such networks, and a pivotal factor in the fulfillment of the target KPIs, is the virtual radio access network (vRAN), which allows high flexibility in the control of the radio link. However, to fully exploit the potentiality of vRANs, an efficient mapping of the rapidly varying context to radio control decisions is not only essential, but also challenging owing to the interdependence of user traffic demand, channel conditions, and resource allocation. Here, we propose CAREM, a reinforcement learning framework for dynamic radio resource allocation in heterogeneous vRANs, which selects the best available link and transmission parameters for packet transfer, so as to meet the KPI requirements. To show its effectiveness, we develop a testbed for proof-of-concept. Experimental results demonstrate that CAREM enables efficient radio resource allocation under different settings and traffic demands. Also, compared to the closest existing scheme based on neural networks and the standard LTE technology, CAREM exhibits an improvement of one order of magnitude in packet loss and latency, while it provides a 65% latency improvement relative to the contextual bandit approach.

This chapter starts the second part of this thesis, where we will shift the specific focus of the 5G network management problem from the centralized orchestration, potentially applied to the entire network, to the distributed local management of edge resources.

Specifically, CAREM, compared to the approaches presented in the previous chapters, does not manage edge applications; instead, it focuses on the management of vRANs. However, the fine-grained control policy of a vRAN can not be centralized in the core network, but have to be colocated with the vRAN itself. In fact, as we will better see in this chapter, the time scale of the control loop required to effectively allocate vRAN radio resources is at least an order of magnitude lower than the time scale that applies to network orchestration decisions such as service scaling. This tight timing is not feasible unless the control logic connects to the vRAN to receive performance metrics and to apply control decisions with a low latency interface.

Part of the work described in this chapter has been already published in S. Tripathi, C. Puligheddu, C. F. Chiasserini and F. Mungari, "A Context-Aware Radio Resource Management in Heterogeneous Virtual RANs," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 1, pp. 321-334, March 2022, doi: 10.1109/TCCN.2021.3115098, ©2022 IEEE.

4.1 Introduction

The envisaged paradigm of new-generation mobile technologies is aimed to serve a broad spectrum of applications having diverse requirements on various key performance indicators (KPIs), ranging from high reliability and low latency to large-scale connectivity and massive data rates [104]. To accommodate such an ambitious vision, new generation wireless access networks are required not only to integrate various flexible multi-access technologies such as mmWave and massive MIMO [105], but also to provide a versatile radio resource management (RRM) system that can ensure efficient spectrum utilization and seamless interoperability [106].

A powerful concept addressing such needs is the virtualization of the radio access network (RAN), wherein the legacy communication system is decoupled by centralizing the softwarized radio access through virtual machines or containers running on servers at the edge of the cellular network [107, 108, 108]. While this makes the network more agile and minimizes the requirement of expensive dedicated hardware, the edge may host several applications competing for resources, thereby limiting the efficiency of radio functions [109]. Besides, with an unprecedented increase in the number of devices trying to concurrently access the virtual RAN (vRAN), it is expected to observe in the near future a 1,000-fold growth in network traffic [110]. This will contribute to complex interference dynamics and will require sophisticated techniques for RRM, which can

effectively cope with both the diverse performance requirements of the applications to be supported and the rapidly varying network and channel conditions.

Evidently, the unification of hybrid technologies under the new-generation cellular umbrella adds to the complexity of the problem, thereby making the use of conventional theoretic approaches often inadequate to achieve optimum traffic and resource management, owing to intricate mathematical modeling and complex dependencies between network and channel variables. It has thus become indispensable to design innovative solutions that can effectively deal with the system complexity thanks to a fully automated, data-driven approach.

Recently, machine learning (ML) techniques have shown to hold enormous potential in addressing the challenges of applying standard mathematical optimization frameworks to resource allocation problems in vRANs and in allowing an automatic system control [111]. A plethora of learning-based techniques including supervised, unsupervised, reinforcement learning (RL), and deep learning have been proposed [112–114] for heterogeneous networks in general, to tackle resource allocation problems (see Sec. 4.2 for a more detailed discussion). However, it is worth noting that, while deep learning approaches are computationally intensive, the primary challenge associated with simpler ones such as supervised/unsupervised learning is the creation of an exhaustive dataset for training the model. Besides, in case of a rapidly changing environment, frequent retraining of the model is required to achieve the desired accuracy, which can be expensive when there are stringent latency constraints. To this end, it is required to devise a framework that is easy to train in non-stationary environments, yet effective in making intelligent choices in an autonomous fashion using near real-time feedback on channel conditions and temporal variation of user demand so as to improve performance and reliability of the network.

In this work, we leverage the advantages offered by ML and develop a context-aware, RL-based solution to radio resource management in heterogeneous vRANs. Our scheme, named CAREM (Context-Aware Radio rEsource Management), is devised considering a formulation for RRM based on sequential decision making [115], which, thanks to a persistent interaction between the learning agent and its environment, can effectively cope with time-varying operating conditions. The key contributions of this work are as follows:

1. We design CAREM, a framework using differential semi-gradient State-Action-Reward-State-Action (SARSA) for periodic RRM in a multi-user vRAN scenario. CAREM efficiently identifies the radio link to be used, allocates radio resources, and sets such transmission parameters for packet transfer as the modulation and

coding scheme (MCS) while meeting two of the main KPI requirements identified by 3GPP [116], namely, packet loss and latency.

2. Since each heterogeneous link features a maximum available resource capacity, we define an algorithm ensuring that, if multiple users are assigned to the same link, the allocation is Pareto-efficient fair and the overall allocated resources do not exceed the link maximum capacity.

3. We investigate the complexity of the proposed CAREM framework, and introduce a two-fold approach to expedite the convergence. Firstly, high dimensionality of context variables is addressed using a practical tile coding approach. Secondly, the action space is designed as a subset of discrete positive integers, which in turn limits its cardinality and facilitates simultaneous selection of several action components (e.g., link, fraction of radio resources, MCS) using a single action.

4. A proof-of-concept is provided in the context of heterogeneous communications and multiple users, by designing a testbed implementing CAREM over 3GPP LTE and IEEE 802.11p links using software defined radios (SDR).

5. The CAREM performance is evaluated under different settings, including different decision periodicity, number of links and connected users, and values of traffic load. The results show that, as the learning converges, CAREM can be efficiently used for link, MCS and radio resource selection in vRANs. With respect to the 100-ms decision, 1-s decisions significantly reduce the computational demand, without any noticeable performance degradation. Further, when compared against the closest existing RRM technique in [109], contextual bandit approach [117], and standard LTE, CAREM always shows significantly better performance, with an improvement of one order of magnitude in both packet loss and latency with respect to the radio policy in [109] and standard LTE, and a 65% latency improvement relatively to contextual bandit.

We remark that the use of a reward signal for associating the best decisions to different contexts, and the dependence of future contexts on current decisions are two important aspects of this problem which makes it different from a much simpler contextual bandit formulation [117]. To effectively tackle these challenges, we adopt a model-free, full-blown, RL approach using the differential semi-gradient SARSA algorithm. Unlike Q-learning [118], which is a popular off-policy RL approach useful for episodic tasks, SARSA has low per-sample variance, thereby making it less susceptible to convergence

problems. Also, in a continuous task setting such as RRM where it is required to care for agent's performance during the exploration phase, online learning using SARSA is preferred due to its conservative nature of avoiding high risk actions that generate large negative rewards from the environment. To our knowledge, no existing work has presented such comprehensive and dynamic framework for RRM, keen on fast and reliable data transmission in heterogeneous vRANs.

The rest of the paper is organized as follows. Sec. 4.2 reviews the related works, while Sec. 4.3 and Sec. 4.4 introduce, respectively, our system model and the proposed RL framework. Sec. 4.5 describes the implementation of our solution and the developed testbed, and Sec. 4.6 discusses performance evaluation results. Finally, Sec. 4.7 concludes the paper.

4.2 Related Work

Owing to the intricate channel-network dynamics and complexity of heterogeneous networks, several works have aimed at devising strategies for effective resource utilization, while meeting the stringent KPI requirements of different applications to be supported in a cellular network. The state-of-the-art primarily revolves around the idea of learning environment variables and their evolution over time for optimizing resource utilization and improving real-time system performance. In particular, learning-based techniques have been developed to address multichannel access, scheduling and allocation of resource blocks, modulation and coding schemes, computation resources, transmit power and data rate, while maximizing KPI satisfaction, with particular emphasis on throughput, latency, packet loss, channel utility, and user fairness.

The problems of dynamic rate allocation as well as of joint channel and rate selection for throughput maximization have been studied in [119–124]. While [119, 120] use a multi-armed bandit formulation and exploit unimodal feature of reward over the arms using UCB policies, an algorithm based on Thompson sampling is used in [121] for achieving link-rate selection in logarithmic time regret. Likewise, a ML approach, also based on multi-armed bandit using tug-of-war dynamics, is presented in [122, 123] for channel selection in IoT networks. These works however do not explore the contextual information from the environment for transmission parameter optimization. To address this limitation and further improving the performance, a structured RL approach using contextual unimodal multi-armed bandit is proposed in [124], for dynamic rate selection and distributed resource allocation.

Unlike the bandit model, the RL approach is more popular in recent literature for radio resource provisioning problems, especially if the action corresponds to a decision making scenario with discrete choices. RL-based schemes are proposed for selecting the radio access technology in heterogeneous networks using network-centric [125], and user-centric approaches [126]. In [127], a policy gradient actor-critic algorithm is studied for user scheduling and resource allocation in energy-efficient heterogeneous networks. The works in [128] and [129] investigate dynamic spectrum access in cognitive radio networks using the RL framework, with the aim to achieve high controllability in spectrum sharing and to minimize the sensing duration. Owing to delay-sensitivity and massive volume of data traffic in 5G access networks, an RL-based scheduling scheme is introduced in [130, 131] to minimize the packet delay and drop rate. The study in [109], instead, proposes a deep deterministic policy gradient algorithm based on actor-critic neural network and a classifier for resource control decisions. This is the most relevant work to ours, as it specifically addresses a virtualized access network and presents an implementation of the solution in a full-fledged testbed. Under high mobility and high traffic demand, RL-based radio resource control in 5G vehicular networks is tackled in [132], with the goal of adaptively changing uplink to downlink ratio in a frequency band.

Advanced ML such as deep learning techniques are of interest for resource allocation problems when the size of state-action space is large, leading to slow convergence of RL approaches. A deep Q-network for channel selection is proposed in [133, 134] to adaptively learn in time-varying scenarios subject to maximization of network utility. The study in [135] envisions an adaptive deep actor-critic, RL-based framework for channel access in dynamic environment for multi-user scenarios. Deep RL is explored for selection of suitable MCS for primary transmissions in cognitive radio networks in [136]. In a similar setting, the study in [137] investigates a deep learning dynamic power control method for a secondary user to coexist with the primary user. A distributed dynamic power allocation using multi-agent deep RL is developed in [138], which exploits channel state and quality of service information to maximize a sum-rate utility function. Deep RL is also applicable to radio resource management in vehicular networks including channel selection, optimal sub-band allocation, and power control, as shown in [139–142].

At last, we mention that a preliminary version of this work has appeared in our conference paper [143].

Novelty. First, unlike most of prior art on RRM, we address the selection of link, radio resources, and packet transmission parameters, so that the target values of packet loss rate and latency KPIs are achieved for each traffic flow. While in terms of actions

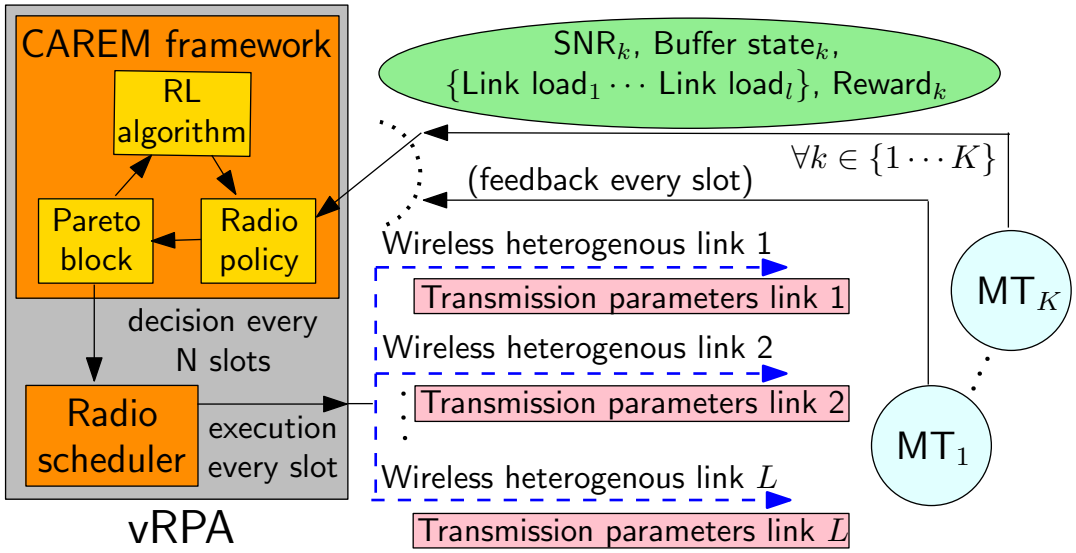


Fig. 4.1 vRAN system model and architecture of CAREM framework. CAREM gathers per slot contextual information for each active MT and makes decisions every N slots. Through an RL algorithm, a radio policy and a Pareto block, it selects the best available radio link and transmission parameters for every MT, which is input to the radio scheduler that executes these decisions every slot.

of the RL framework [126, 142, 136] are somewhat aligned to our work, their learning objectives and KPI requirements are very different. Furthermore, with respect to all the above works, including [109], we address connectivity between a radio point of access and multiple users over *heterogeneous* links. In addition, the RRM policy in CAREM (i) is spontaneously learned and updated over time by its continuous interaction with the environment, thus being able to adapt continuously to time-varying channel and network dynamics, and (ii) provides a fair Pareto-efficient allocation of capacity-constrained resources, thus leading to an effective management of multi-user connectivity. Finally, we provide a proof-of-concept of the proposed solution, implementing it in a multi-technology, multi-user, SDR-based testbed.

4.3 System Architecture

In this section, we present the system model considered for provisioning of radio resources via CAREM. Although our approach and methodology are general and can apply to any number and type of vRAN technologies, while describing the framework we refer for concreteness to the communication environment we implemented in our testbed where LTE and IEEE 802.11p links are available.

We leverage SDR interfaces enabling point-to-point communications between virtual radio point of access (vRPA) and K users, hereinafter referred to as Mobile Terminals (MTs), implemented at the edge of the network. The architecture of the proposed CAREM framework in a vRAN is presented in Fig. 4.1. We also envision that user applications such as video streaming, gaming, road safety services (e.g., for vehicles or vulnerable road users) are deployed at the edge through containerized infrastructure, and possibly co-located with radio functions including radio resource management, scheduling, admission control, and reliable packet delivery. In the following, we focus on the downlink data transfers from the vRPA to the MTs, although our framework can be easily extended to uplink scenarios as well.

On the cellular downlink, the vRPA determines the number of radio resources per MT, i.e., of resource blocks (RBs), required for the transmission of data packets, based upon the signal-to-noise ratio (SNR) reported by each MT through the Channel Quality Indicator (CQI). Conversely, on the IEEE 802.11p link, the vRPA accesses the channel to transmit to the MTs using the CSMA-based scheme foreseen by the corresponding standard To minimize packet loss over the radio links, at the physical layer data packets are modulated and encoded using a suitable MCS (namely, twenty-nine and eight different MCS values are possible on the LTE and IEEE 802.11p links, respectively). Furthermore, at the MAC layer, an automatic repeat request error control is in place, i.e., an unsuccessfully transmitted packet can be resent till a maximum number of allowed retransmission attempts. Beside SNR, the knowledge of the amount of data waiting to be transmitted towards a MT and of the traffic load supported on available links is also essential for radio resources provisioning, so as to minimize packet loss. The information on the buffer occupancy can be acquired using buffer state reports at the MAC layer.

The proposed CAREM framework is a dynamic resource controller that is included as an extended functionality within the vRPA and that interacts with the radio scheduler implemented therein. At the logical level, it is composed of as many RL agents as the number of quality of Service (QoS) classes to be supported, each agent differing from the others in the target KPIs. Then, each RL agent can sequentially handle multiple MTs. The RL agent considers the status corresponding to each MT, namely, SNR, buffer state, and also the status of aggregate traffic load already hosted on the available links, and selects the best action. The latter includes: link, MCS value, and number of RBs or channel utilization time, so as to maximize the associated reward, hence meet the KPI requirements.

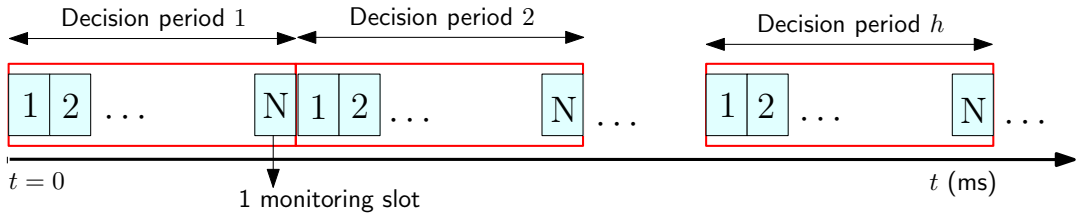


Fig. 4.2 Relation between a decision period and monitoring slots, each decision period consists of N monitoring slots.

For each MT, the SNR, buffer state, and link load values are periodically monitored in a time-slotted fashion. From the learning of the environment variables, a decision on the action to be adopted is made by the CAREM framework every N time slots. During the decision making process, the link load values are continuously updated as the RL agent sequentially selects an action for each MT. Subsequently, these decisions are enforced as radio policies by the scheduler on a per-slot basis, till the next decision-making event. Specifically, while the periodicity with which the scheduler operates remains constant, we consider that CAREM may make decisions with periodicity equal to $N \geq 1$ slots. A clear demarcation of decision period and monitoring slot is depicted in Fig. 4.2.

4.4 The CAREM Framework

The joint impact of channel and network dynamics on RRM in wireless networks is far from being trivial; therefore, for efficient resource mapping in non-stationary environments, we adopt a model-free approach that does not require an environment model. Our CAREM framework, depicted in Fig. 4.1, continuously maps the variations in transmission channel and traffic load into a context, thus learning the best action for each given context. For sake of clarity and without loss of generality, below we focus on a single RL agent, handling multiple MTs connected to the vRPA and receiving traffic flows belonging to the same QoS class.

Each RL agent includes three blocks, interacting with each other as depicted in Fig. 4.3: (i) the radio policy selecting the best action for each MT connection, (ii) the Pareto block, refining the previous action with respect to the allocation of capacity-constrained resources, and (iii) the RL algorithm implementing a differential semi-gradient SARSA. Note that in all our experiments we observed that a single-agent implementation can cope with a large number of MTs, while entailing a negligible latency due to the decision-making process. Nevertheless, whenever the container, or virtual machine, in which the RL agent is implemented needs to be scaled out due to an exceedingly high computing

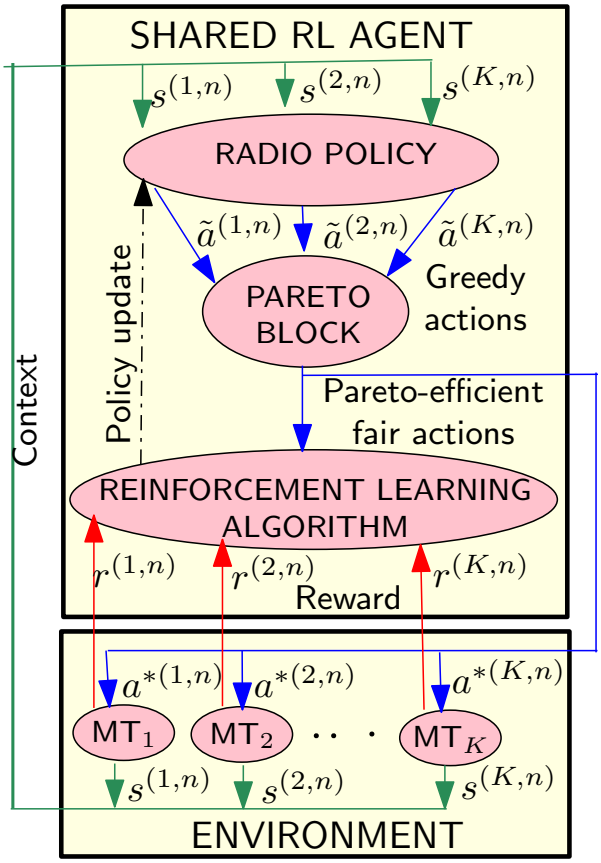


Fig. 4.3 Components of the CAREM framework. Radio policy maps contextual information gathered from the environment into actions. Whenever needed, the Pareto block refines the actions so that they meet the link capacity. The RL algorithm updates the radio policy using differential semi-gradient SARSA.

burden, such techniques as those developed, e.g., within the 5G-Transformer [144] and the 5Growth [58] projects, can be effectively applied.

Below, we detail the three main components of CAREM.

4.4.1 Radio policy

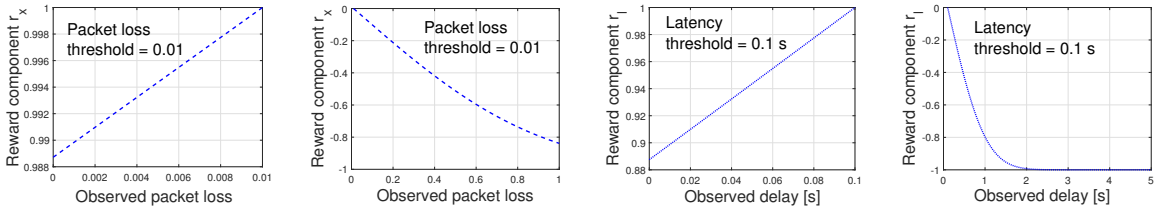
The radio policy block continuously maps observation of contexts from the environment to decisions in the form of actions, for each MT. The goal of the RL model is to train the agent to find a policy that eventually maximizes the cumulative reward from an uncertain environment. The elements composing the radio policy are introduced below; unless otherwise specified, we refer to a generic decision-making period, which is composed of N monitoring slots.

Context Space. For the generic MT k ($k \in \{1, \ldots, K\}$) connected to the vRPA, in monitoring slot n ($n = 1, \ldots, N$) the agent observes a context vector $s^{(k,n)} \in \mathcal{X}$, applies action $a^{(k)} \in \mathcal{A}$, which was selected at the end of the previous decision period and holds for the whole current one, and receives a reward value $r(s^{(k,n)}, a^{(k)})$ as feedback. As discussed in Sec. 4.3, the environment variables, namely, SNR, buffer state, and links load, influence the choice of the link, MCS, and radio resource allocation. Let $\gamma^{(k,n)}$ and $\sigma^{(k,n)}$ be, respectively, the SNR and the buffer state reported by the k-th MT during the n-th monitoring slot. Also, let $\zeta^{(k,l,n)}$ denote the aggregate link load already on link l ($l \in \{1, \ldots, L\}$) during the n-th monitoring slot while making a decision for MT k. We can then write the generic context vector as $s^{(k,n)} := \{\gamma^{(k,n)}, \sigma^{(k,n)}, \zeta^{(k,1,n)} \cdots \zeta^{(k,L,n)}\}$.

Action Space. Let us denote the amount of capacity-constrained resource allocated to MT k by $\rho^{(k)}$, e.g., the number of RBs in LTE or channel utilization time in IEEE 802.11p. Further, for MT k, we map the tuple (link, MCS, resource allocation), $\{l^{(k)}, \omega^{(k)}, \rho^{(k)}\}, \forall k \in \{1, \cdots, K\}$ in the n-th monitoring slot within the same decision-making period into a new, single action. Thus, the action space comprises choices for the selection of the appropriate link, MCS, and the amount of radio resources over the chosen link. We recall that an action is selected at the end of every decision period, and it is applicable to the subsequent N monitoring slots.

Next, without loss of generality and for notation simplicity, we focus on two links only, and denote the number of different MCS values supported over each link by i and j, respectively. Also, we discretize the quantity of radio resources that can be allocated over each link, and indicate them with p and q, respectively. Then, the action space is given by $\mathcal{A} := \{a^{(k)} \in [0, (ip + jq - 1)]\}$, such that $a^{(k)} = \{0, \ldots, ip - 1\}$ when the first (e.g., LTE) link is selected, and $a^{(k)} = \{ip, \ldots, ip + jq - 1\}$ in case of the second (e.g., IEEE 802.11p) link. The advantage of such definition of an action is that it limits the action space to a subset of discrete positive integers with low cardinality, and facilitates simultaneous selection of several resources with a single action.

Reward. Given a traffic flow, we consider as KPIs the packet loss rate and the latency observed at the MAC layer during packet transmission. To meet the KPI requirements at the MT, it is required to provide the traffic flow with radio resources such that the observed KPIs are always less or equal to their target values (hereinafter also referred to as thresholds). Beside meeting the KPI thresholds, it is essential to keep the observed KPIs as close as possible to the respective KPI thresholds for optimum utilization of network resources: substantially better values than the target ones would indeed translate into a waste of resources. Thus, the choice of reward function should be such that it



(a) (b) (c) (d)

Fig. 4.4 Variation of reward component as a function of packet loss r_x (fulfilled (a) and unfulfilled (b) target KPI) and as a function of latency r_l (fulfilled (c) and unfulfilled (d) target KPI). The reward components take positive values as long as the KPI targets are fulfilled, and negative otherwise; further, the closer the KPI to its target value, the higher the reward.

equally accounts for both the KPIs and its value increases as the observed KPIs approach the corresponding thresholds and vice versa.

Let the observed packet loss rate, target packet loss rate, observed latency, and target latency be denoted with x_o, x_{th}, l_o, and l_{th}, respectively. We define the reward value r as the sum of two reward components corresponding to packet loss $r_x(\cdot)$ and latency $r_l(\cdot)$, respectively. Thus, for the k-th MT, at the n-th monitoring slot, we have:

$$r(s^{(k,n)}, a^{(k)}) = r_x(s^{(k,n)}, a^{(k)}) + r_l(s^{(k,n)}, a^{(k)}) \tag{4.1}$$

where packet loss and latency components are given by:

$$r_x(s^{(k,n)}, a^{(k)}) = 1 - \text{erf}(x_{th} - x_o) \tag{4.2a}$$
$$r_l(s^{(k,n)}, a^{(k)}) = 1 - \text{erf}(l_{th} - l_o) \tag{4.2b}$$

if the target KPIs are met, and by:

$$r_x(s^{(k)}, a^{(k,n)}) = \text{erf}(x_{th} - x_o) \tag{4.3a}$$
$$r_l(s^{(k)}, a^{(k,n)}) = \text{erf}(l_{th} - l_o) \tag{4.3b}$$

otherwise.

Since the maximum and minimum value of the erf function lies between $+1$ and -1, we have: $-2 \leq r(s^{(k,n)}, a^{(k)}) \leq 2$. Our choice of erf for estimating individual reward components is motivated by its shape, which takes 0 value at the origin, and gradually increases (decreases) and saturates to the maximum (minimum) value in the positive (negative) direction. Consequently, for the individual reward components, in the positive

region of operation, i.e., when the KPI threshold is met, the reward value is positive and it further increases to its maximum value at +1 as the observed KPI approaches its target KPI value. Likewise, in the negative region of operation, i.e., when the KPI threshold is not met, the value of the individual reward components is negative, which further reduces and saturates to the minimum value −1 as the observed KPI moves away from the KPI threshold. Note that the penalty in the reward component for observed KPI overshooting the KPI threshold is larger than for observed KPI undershooting the KPI threshold by the same amount. This is so because an observed KPI value that exceeds the KPI threshold adversely affects the QoS at the user end, which is a critical issue from the system design point of view. On the contrary, the case of an observed KPI undershooting the KPI threshold is an acceptable situation, however even in this case, we do not want the observed KPI value to deviate from the KPI threshold by a large amount, as this would lead to the system performing exceptionally well at the cost of extra resource consumption. The variation of the reward components as a function of the observed KPI values for packet loss and latency is shown in Fig. 4.4, which highlights how the behavior of the individual reward components in the typical range of observed KPI values is as desired.

We recall that the goal of the RL agent is to eventually maximize the cumulative reward measured as the sum of immediate reward and future rewards in the long run. To this end, we consider the generic decision period h and, extending the previous notation, we let $a^{(k,h-1)}$ denote the action for MT k selected in decision period $(h-1)$ and applied in decision period h. We then define the average reward over h as,

$$\overline{r}(\mathbf{s}^{(k,h)}, a^{(k,h-1)}) = \frac{\sum_{n=1}^{N} r(s^{(k,n)}, a^{(k,h-1)})}{N} \tag{4.4}$$

where $\mathbf{s}^{(k,h)}$ is the vector of states observed for MT k in the N monitoring slots in decision period h, while $a^{(k,h-1)}$ is the action for MT k selected in decision period $h-1$ and applied in decision period h. Then, we adopt the definition of cumulative reward for the k-th MT, observed during decision period h, as the differential return $G^{(k,h)}$ [145],

$$G^{(k,h)} = \sum_{\ell=0}^{\infty} \overline{r}(\mathbf{s}^{(k,h+\ell)}, a^{(k,h+\ell-1)}) - (l+1)\widehat{r}(k,\pi) \tag{4.5}$$

where $\pi : \mathcal{X} \rightarrow \mathcal{A}$ denotes the radio policy mapping the context space of each MT into actions, and $\widehat{r}(k, \pi)$ in [145]:

$$\widehat{r}(k, \pi) = \lim_{h \to \infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[\overline{r}(\mathbf{s}^{(k,t)}, a^{(k,t-1)}) | \overline{s}^{(k,1)}, a^{(k,0)} \sim \pi] \,. \tag{4.6}$$

In (4.6), we consider $h = 0$ to be the time at which the algorithm execution started, and $\overline{s}^{(k,1)}$ is the mean state computed averaging over the state values observed in the N monitoring slots of the initial decision period. Thus, $\widehat{r}(k, \pi)$ is obtained as the average of the reward conditioned on $\overline{s}^{(k,0)}$ and the subsequent actions taken according to policy π.

Action value estimation. Given decision period h, at the end of the corresponding N monitoring slots, actions need to be evaluated in order to ultimately select the best one. To this end, we compute the average context over the N monitoring slots in h for each given MT k, as

$$\overline{s}^{(k,h)} = \sum_{n=1}^{N} \frac{y_n s^{(k,n)}}{\sum_{n=1}^{N} y_n} \tag{4.7}$$

where $y_n > 0$ and $y_N > y_{N-1} > \cdots y_1$ are the weights assigned such that the latest context has the highest weight. Although they can be arbitrarily set, in our experiments we fix them to $1, \ldots, N$, in accordance with the temporal sequence of the monitoring slots. We then quantify the goodness of taking an action in such a context using action values. For MT k, if $a^{(k,h)}$ is selected based on state $\overline{s}^{(k,h)}$ under policy π, then its action value $q_\pi(\overline{s}^{(k,h)}, a^{(k,h)})$ is defined as expected differential return conditioned on $\overline{s}^{(k,h)}$ and $a^{(k,h)}$, following policy π. Mathematically,

$$q_\pi(s, a) = \mathbb{E}_\pi[G^{(k,h)} | \overline{s}^{(k,h)} = s, a^{(k,h)} = a] \,. \tag{4.8}$$

Apparently, a policy, π, can be better than any other policy π' if $q_\pi(s, a) \geq q_{\pi'}(s, a)$. Since the context vector comprises SNR, link aggregate traffic load and buffer state, context space $\mathcal{X}$ is real and an uncountable number of states are possible. Consequently, tracking action values corresponding to different contexts is not scalable. To overcome this problem, we use a practical method for action value estimation using function approximation in an F-dimensional space, yielding the following approximated function,

$$\hat{q}_\pi(\overline{s}^{(k,h)}, a^{(k,h)}, w) = \sum_{f=1}^{F} w_f x_f(\overline{s}^{(k,h)}, a^{(k,h)}) \tag{4.9}$$

where $\mathbf{w} = [w_1, \ldots, w_F] \in \mathbb{R}^F$ and $x_f(\overline{s}^{(k,h)}, a^{(k,h)})$ denote the weight and feature vector, respectively. Here, feature vector $x_f(\overline{s}^{(k,h)}, a^{(k,h)})$ is generated using tile coding [146], which converts a point in the 2-dimensional context vector into a binary feature vector such that vectors of neighboring points have a high number of common elements. The continuous space of context variables is tucked up with tiles, and each tile corresponds to an index in the binary feature vector. Several offset grid of tiles, called tilings, are then stacked over the space to create regions of overlapping tiles. We have used 8 tilings, with 512 tiles each. Every context vector falls in one tile in each of the 8 tilings, which correspond to 8 features.

Action selection. The estimation of the action values is followed by an ϵ-greedy action selection policy [145], which selects the best action for each MT so as to maximize its cumulative reward over an infinite time horizon. We consider an ϵ-greedy action selection with $\epsilon = 0.5$ and ϵ-decay factor $= 0.999$. Thus, for MT k, if the average context over decision-making period h, $\overline{s}^{(k,h)}$, and the action value estimates for all possible actions $a^{(k,h)} \in \mathcal{A}$ in $\overline{s}^{(k,h)}$ are obtained as $\hat{q}_\pi(\overline{s}^{(k,h)}, a^{(k,h)}, \mathbf{w})$, the greedy action for the MT, $\tilde{a}^{(k,h)}$, is chosen with probability $1 - \epsilon$ such that $\tilde{a}^{(k,h)} = \text{argmax}_a\, \hat{q}_\pi(\overline{s}^{(k,h)}, a^{(k,h)}, \mathbf{w})$. The ϵ parameter decays by a factor of 0.999 in the subsequent decision period. This favors higher exploration while the environment is still unfamiliar; with progression of time, instead, it allows for further exploitation of the environment knowledge gained during the exploration, so as to maximize the expected return.

4.4.2 Pareto block

At the input of the CAREM framework, contexts from different MTs are considered independently from each other, which may sometimes lead the radio policy to choose actions for different MTs where the sum of individual allocations of a capacity-constrained resource exceeds its respective maximum availability. We solve this issue by introducing a novel algorithm that further fine tunes the resources allocated by the radio policy. Such refinement makes sure that if multiple MTs are using the same link, the allocation is Pareto-efficient fair and the sum of allocated resources to the MTs adheres to the maximum capacity constraint of the link.

We focus again on a given decision period, thus omitting the dependency on h, and we map the Pareto-efficient fair allocation of a capacity-constrained resource on a given link l across all MTs assigned to l, onto a multi-criteria optimization problem as detailed below. Let us denote the set of MTs assigned to link l by $\mathcal{K}_l$. Given a set

of coefficients $v_k \geq 0, k \in \mathcal{K}_l$, such that $\sum_{k\in\mathcal{K}_l} v_k = 1$, it is required to find a solution $S = \{\rho^{(k)}\}_{k\in\mathcal{K}_l}, S \in \Phi$, that maximizes $\sum_{k\in\mathcal{K}_l} v_k\Gamma^{(k,n)}(S)$ such that $\sum_{k\in\mathcal{K}_l} \rho^{(k)} \leq \rho_{max}$. Here, Φ is the set of feasible solutions, $\rho^{(k)}$ is the capacity-constrained resource allocated to the k-th MT during the considered decision-making period, $\Gamma^{(k,n)}(S)$ is the criteria function denoting the reward of MT k in the n-th monitoring slot following the resource allocation strategy S, and ρ_{max} is the maximum availability of the capacity-constrained resource.

The optimization problem is solved using an iterative multi-objective search and update algorithm described as follows. The Pareto block is invoked if the sum of allocated capacity-constrained resource across all MTs exceeds ρ_{max}. To start with, the capacity-constrained resource $\rho^{(k)}$ is extracted from the greedy action $\tilde{a}^{(k)}$ to form solution $S_1 = \{\rho^{(k)}\}_{k\in\mathcal{K}_l}$. Then each $\rho^{(k)}$ in S_1 is scaled so that $\sum_{k\in\mathcal{K}_l} \rho^{(k)}(S_1) \leq \rho_{max}$. Note that, beside S_1, other solutions S_i are possible as well where $\rho^{(k)}(S_i) \geq \rho^{(k)}(S_1)$, as intuitively, if an action $\tilde{a}^{(k)} = \{l^{(k)}, \omega^{(k)}, \rho^{(k)}\}$ is feasible for a given $\rho^{(k)}$, it will also be feasible for any other allocation $\rho'^{(k)} \geq \rho^{(k)}$. In such case, $\sum_{k\in\mathcal{K}_l} \rho^{(k)}(S_i) \geq \rho_{max}$, however since scaling is anyways imperative to satisfy the maximum capacity constraint, it is in best interest to consider all such possible solutions. In view of this argument, we create an expanded solution set $\mathcal{S}_e = \{S_1, S_2, \ldots\}$ such that $\forall S_i \neq S_1$,

$$\rho^{(k)}(S_i) \geq \rho^{(k)}(S_1), \forall k \in \mathcal{K}_l, \wedge \sum_{k\in\mathcal{K}_l} \rho^{(k)}(S_i) \leq |\mathcal{K}_l|\rho_{max}\,. \tag{4.10}$$

Further, we create scaled expanded solution set $\mathcal{S}_s$,

$$\mathcal{S}_s = \{S_i/|\mathcal{K}_l|\}_{\mathcal{S}_e}\,. \tag{4.11}$$

Subsequently, Pareto dominant solution set $\mathcal{S}$ is obtained through an iterative search and update over $\mathcal{S}_s$ using the following condition $\forall S' \in \mathcal{S}$ and $S'' \in \mathcal{S}_s$,

$$\Gamma^{(i)}(S') > \Gamma^{(i)}(S''), \Gamma^{(j)}(S') \geq \Gamma^{(j)}(S''), \forall i, j \in \mathcal{K}_l, i \neq j\,. \tag{4.12}$$

Finally, an optimal solution $S^* = \{\rho^{*(k)}\}_{k\in\mathcal{K}_l}$ is chosen from the Pareto dominant solution set $\mathcal{S}$ such that $\forall S' \in \mathcal{S}, S^* \in \mathcal{S}$,

$$\max \min_{i\in\mathcal{K}_l}(v_i\Gamma^{(i)}(S^*)) \geq \max \min_{i\in\mathcal{K}_l}(v_i\Gamma^{(i)}(S'))\,. \tag{4.13}$$

Since S^* maximises the minimum criterion function across all $k \in \mathcal{K}_l$, it is the required Pareto-efficient fair solution. For each link, the procedure to obtain such solution, i.e.,

Algorithm 3 Allocation of the capacity-constrained resource on radio link l

Input: greedy actions, $\tilde{a}^{(k)} = \{l^{(k)}, \omega^{(k)}, \rho^{(k)}\}, \forall k \in \mathcal{K}_l$
Extract capacity-constrained resource allocation $S_1 = \{\rho^{(k)}\}_{k \in \mathcal{K}_l}$ from greedy actions
if $\sum_{k \in \mathcal{K}_l} \rho^{(k)} \leq \rho_{max}$ **then** ▷ Check on the resource allocations on link l
    $S^* \leftarrow S_1$ ▷ Pareto-efficient fair solution
else
    Create expanded solution set, $\mathcal{S}_e$ using (4.10)
    Rescale $\mathcal{S}_e$ to create $\mathcal{S}_s$ using (4.11)
    Identify Pareto dominant solution set $\mathcal{S}$ using (4.12)
    Compute Pareto-efficient fair solution S^* using (4.13)
    Output: Pareto-efficient fair actions, $a^{*(k)} = \{l^{(k)}, \omega^{(k)}, \rho^{*(k)}\}, \forall k \in \mathcal{K}_l$

action $a^{*(k)}$ for each MT k, is summarized in Algorithm 3. Using theorems defined in [147], it can be proved that the obtained solution is Pareto efficient.

4.4.3 Learning algorithm

In the absence of any prior knowledge of the environment, here we exploit the concept of experience-based learning using sample sequences of context, actions, and rewards observed from the actual interaction of the RL agent with the environment. SARSA, an acronym for quintuple $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$, is an on-policy algorithm where learning of the RL agent at time t is governed by its current state S_t, choice of action A_t, reward R_t received on taking action A_t, state S_{t+1} that the RL agent enters after taking action A_t, and finally the next action A_{t+1} that the agent chooses in new state S_{t+1} [145]. Then, given the average context vectors for the different MTs and the possible actions, the key steps involved in the learning of the SARSA approach are: (i) estimation of action values $q_\pi(s, a)$, (ii) action selection for each MT, (iii) Pareto-efficient fair action tuning, and (iv) update of the action-value estimates.

Action value update. Action values satisfy the recursive Bellman equations given as,

$$q_\pi(s, a) = \sum_{r, s'} p(s', r|s, a)[r - \widehat{r}(k, \pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a')] \tag{4.14}$$

where

$$p(s', r|s, a) = \mathbb{P}\Big\{\overline{s}^{(k,h)} = s', \overline{r}(\mathbf{s}^{(k,h)}, a^{(k,h-1)}) = r| \\ \overline{s}^{(k,h-1)} = s, a^{(k,h-2)} = a\Big\}, \tag{4.15}$$

with $\pi(a'|s')$ being the probability of taking action a' in state s' under policy π. This fundamental property forms the basis of the update of the action values of the present context, based on an error term defined as the difference between a target action value and the current action value. Details on Bellman equation and the derivation of the update rule can be found in [145]. Here we consider the temporal difference learning, in which the target action value for the context in the given decision period h is the bootstrapping estimate of action values for the context in the subsequent decision period $(h+1)$. Since the difference in action value estimates of successive contexts drives the learning procedure, error is termed as temporal difference error δ, given by

$$\begin{aligned}\delta = \overline{r}(\mathbf{s}^{(k,h)}, a^{*(k,h-1)}) - \widehat{r}(k,\pi) + \hat{q}_{\pi}(\overline{s}^{(k,h+1)}, \\ a^{*(k,h+1)}, \mathbf{w}) - \hat{q}_{\pi}(\overline{s}^{(k,h)}, a^{*(k,h)}, \mathbf{w})\,.\end{aligned} \tag{4.16}$$

In (4.16), $a^{*(k,h)}$ is the Pareto-efficient fair action for MT k selected in decision period h and applied in decision period $(h+1)$. Also, we recall that $\overline{s}^{(k,h)}$ and $\overline{r}(\overline{s}^{(k,h)}, a^{*(k,h-1)})$ are (resp.) the weighted mean context and mean reward observed over decision period h.

Subsequently, δ is used to update $\widehat{r}(k,\pi)$ and weight vector $\mathbf{w}$ using gradient descent as,

$$\widehat{r}(k,\pi) \leftarrow \widehat{r}(k,\pi) + \beta\delta \tag{4.17}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(\overline{s}^{(k,h)}, a^{*(k,h)}, \mathbf{w}) \tag{4.18}$$

where α and β are the step sizes for updating weight vector and average reward conditioned on initial state and the subsequent actions, respectively. Note, however, that the bootstrapping target itself depends on the weight vector. Consequently, it is biased and does not produce a true gradient descent, hence this is referred to as a semi-gradient method. Step sizes α and β govern the learning rate of the algorithm by deciding how much closer the estimate moves towards the target in a single iteration. If the chosen step sizes are too small, it takes a large time to reach the best values of weights and average reward, hence slower learning, which is clearly undesirable. On the contrary, although a large step size may reduce the training time, there is a possibility of overshooting the true optimum position and oscillating between local optima. To this end, in our experiments, we considered different choices and found $\alpha, \beta = 0.01$ to be the best suited one.

The workflow of the CAREM RL algorithm is summarized in Algorithm 4, where for simplicity we focus on MT k and decision period $(h+1)$. Parameters including the decision-making periodicity, N, and step sizes, α and β, are initialized at the start of the algorithm. Given a decision period $(h+1)$, after observing the context vector, the

radio policy gives as output a greedy action for MT k. Once Algorithm 4 is run for all MTs, the greedy actions are further tuned by the Pareto block to obtain Pareto-efficient fair actions $\{a^{*(k,h+1)}\}$, subsequently reinforcement learning takes place using differential semi-gradient SARSA. Specifically, the temporal difference error δ, the average reward conditioned on initial state and subsequent actions, and the weight vector are updated using (4.16), (4.17), and (4.18), respectively. Note that, although the SNR and buffer state for MT k may be independent of those experienced by other MTs, due to the sequential decision making process, information of the link allocated to MT k is used to update the aggregate link load on each link while making decisions for subsequent MTs.

Algorithm 4 RL algorithm in CAREM for MT k in decision period $(h+1)$

Define parameters: decision-making periodicity N, step sizes $\alpha, \beta \in (0, 1]$
for the n-th monitoring slot in the $h+1$-th decision period, $n = 1, 2, \cdots, N$ **do**
    if $n = 1$ **then**
        $s^{(k,n)} \leftarrow \overline{s}^{(k,h)}, a^{*(k)} \leftarrow a^{*(k,h)}$
    else
        Observe $s^{(k,n)}, a^{*(k)} \leftarrow a^{*(k,h)}$
    Evaluate reward per slot $r(s^{(k,n)}, a^{*(k)})$ using (4.1)
Find mean reward over the $h+1$-th decision period, $\overline{r}(\mathbf{s}^{(k,h+1)}, a^{*(k,h)})$ using (4.4)
Find weighted mean context $\overline{s}^{(k,h+1)}$ using (4.7)
Compute action values $\hat{q}_\pi(\overline{s}^{(k,h+1)}, \cdot, \mathbf{w})$ for all possible actions using (4.9)
Choose $\tilde{a}^{(k,h+1)}$ using the ϵ-greedy policy

4.4.4 Computational complexity analysis

Based on Algorithms 1 and 2, the most complex operations are given by the following steps: (i) greedy action selection for the K MTs, (ii) Pareto-efficient fair action selection on L links, (iii) computation of weighted mean of context and mean reward per decision period for the K MTs, and (iv) update of weight vector for learning radio policy based on KPI observation from the K MTs. Corresponding to each of these steps and considering that the number of radio resources is finite, the computational complexities are given by $\mathcal{O}(K|\mathcal{A}|)$, $\mathcal{O}(K)$, $\mathcal{O}(K)$, and $\mathcal{O}(KN)$, respectively. Hence, the overall complexity is given by $\mathcal{O}(K|\mathcal{A}|) + \mathcal{O}(K) + \mathcal{O}(K) + \mathcal{O}(KN) \approx \mathcal{O}(K|\mathcal{A}|) + \mathcal{O}(KN)$, where the first term is the dominant one as $|\mathcal{A}|$ is much larger than N.

We recall that every decision period is comprised of N monitoring slots. The smaller the value of N, the more frequently CAREM makes decisions and the more the performed

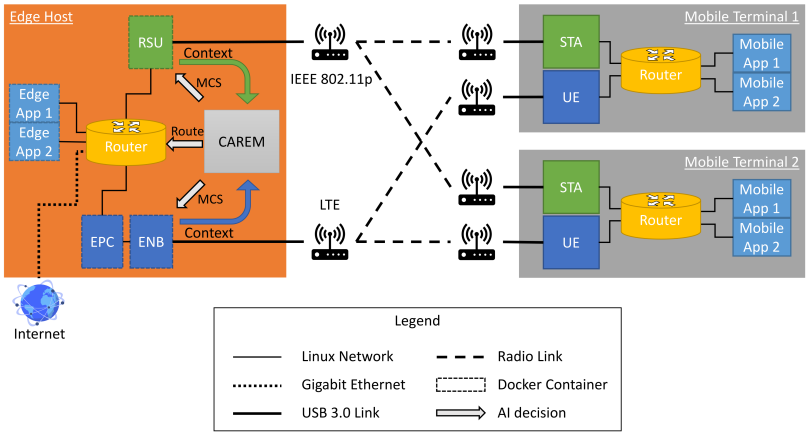


Fig. 4.5 Testbed architecture (for clarity two links and two mobile terminals only are shown). Edge Host provides connectivity to the MTs through a heterogeneous vRAN. For clarity, two links (3GPP LTE and IEEE 802.11p) and two MTs only are shown.

computations, with worst case scenario given at $N = 1$ wherein computations on the order of $\mathcal{O}(K|\mathcal{A}|) + \mathcal{O}(K)$ are done every monitoring slot. It follows that choosing a high N value leads to a significant computation gain, at the cost (as shown in Sec. 4.6) of a marginal performance degradation.

4.5 Testbed Design and Implementation

The testbed architecture, illustrated in Fig. 4.5, is composed of two main interconnected blocks: the edge host (left block) and the mobile terminal (right block). For clarity of presentation, only two links and two MTs are shown. The purpose of the edge host is to provide computational resources and mobile connectivity for services offered by the edge applications, which are then consumed by the mobile applications running at the MTs. Connectivity between the edge host and the MTs is provided through a heterogeneous vRAN integrating the 3GPP LTE (bottom link in Fig. 4.5) and IEEE 802.11p (top link) technologies, both implemented through SDR solutions.

The LTE vRAN is based on srsRAN [148], an open-source SDR LTE stack implementation that offers EPC, eNB, and MT applications. It is compliant with LTE Release 9 and supports up to 20 MHz bandwidth channels as well as transmission modes from 1 to 4, all using the FDD configuration. The IEEE 802.11p transceiver is implemented through a GNU Radio flowgraph, released by the WiME project [149], and it is interoperable with commercial IEEE 802.11p devices. We mention here that, because of processing

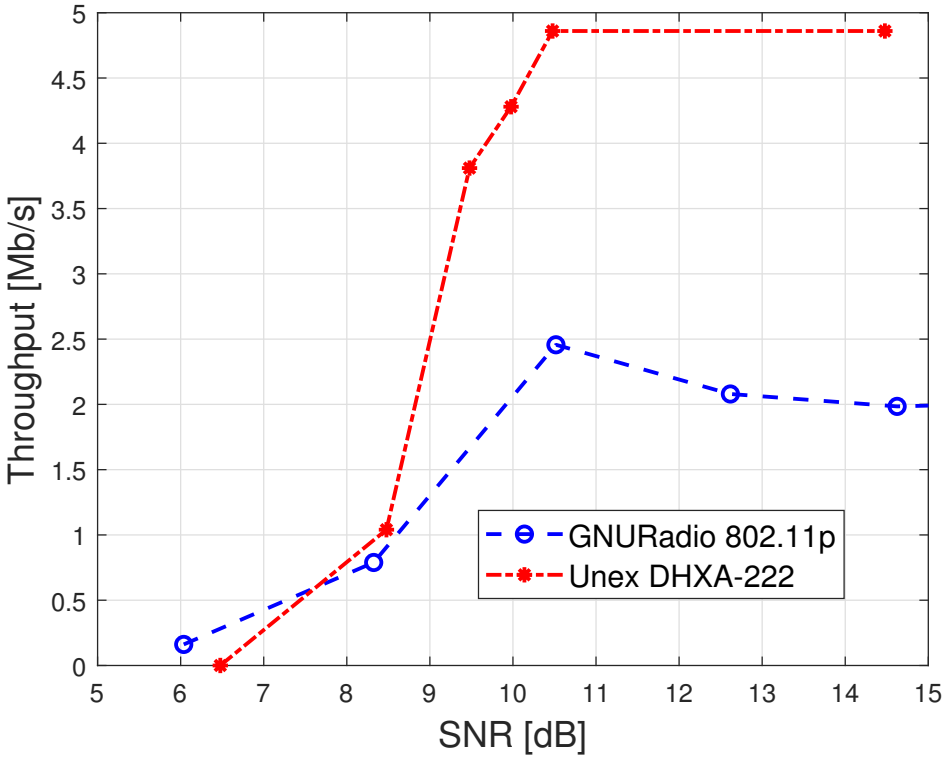


Fig. 4.6 Throughput of IEEE 802.11p with MCS set to 2: comparison between the SDR interface implemented through the GNURadio card (blue curve) and that of the off-the-shelf device, namely, Unex card (red curve). The SDR IEEE 802.11p transceiver performance has been properly scaled to match that of commercial cards.

delay limitations, the IEEE 802.11p transceiver lacks important features of the standard such as ACKs and CSMA/CA mechanisms.

The core component of edge host is the proposed CAREM framework, which controls the operation of the heterogeneous vRAN. The algorithm periodically selects the appropriate link, MCS to be used, and allocated resources on the selected link for downlink packet transmission. To interact with the host operating system network stack, both the SDR solutions expose a tun/tap interface to which an IP address is assigned. A router is connected to those interfaces to steer traffic over the radio links, the host applications, and the internet, according to the link selected by CAREM. The link selection is enforced with dynamic modification to the Linux kernel routing table.

The SDR applications, as well as the edge applications, are implemented and executed within docker containers, to control resource usage and isolate the different applications. The SDR applications have been patched to allow for the dynamic selection of the MCS used for the data radio transmission, according to the radio policy. The srsRAN eNB

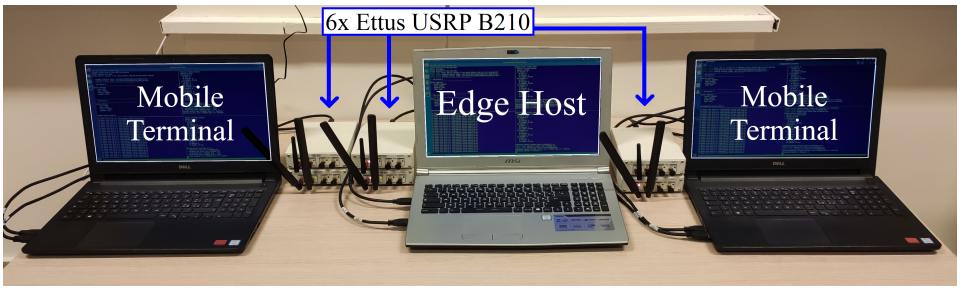
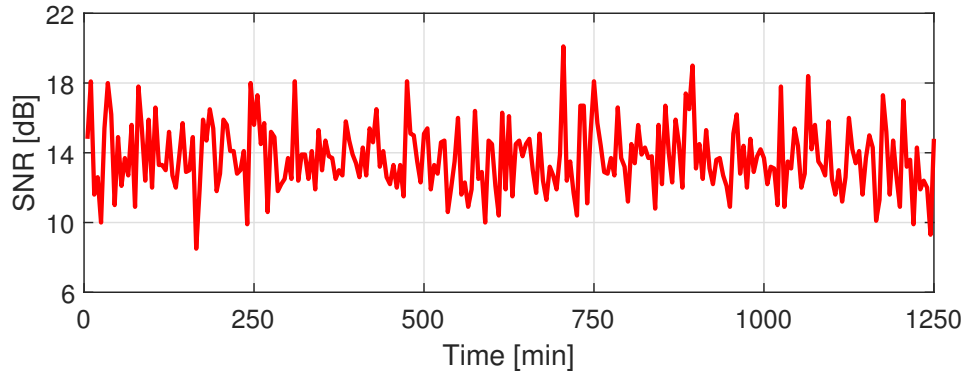


Fig. 4.7 Testbed implementation setup with two MTs and one Edge Host: each MT is connected to two USRP B210 boards, implementing LTE and IEEE 802.11p, respectively.

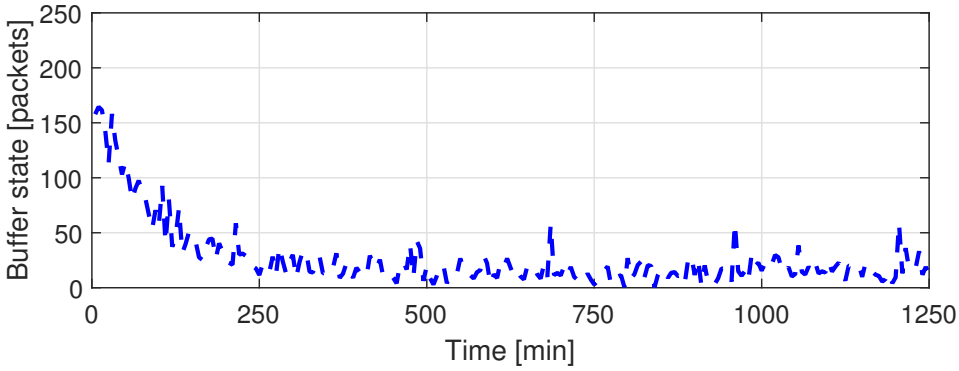
application has been patched to run a dedicated thread that listens to and applies the MCS and the RBs allocation selected by CAREM to the communication with a specific MT. As for IEEE 802.11p, the GNU Radio flowgraph has been modified by adding an XMLRPC server block, which exposes a remote procedure call interface to dynamically set the MCS to be used. The airtime allocation instead has been implemented by limiting the IP flow throughput with Linux Traffic Control according to the radio policy. Indeed, since the transceiver does not support the CSMA/CA mechanism, the physical throughput is known given the MCS. Furthermore, both the SDR applications have been modified in order to collect such context data as the average SNR and the buffer state report through a sidelink connection.

The UDP throughput of the SDR IEEE 802.11p transceiver has been compared to the throughput of a commercial wireless card, namely, the Unex DHXA-222, based on the Qualcomm Atheros AR9462 chipset. Using the same MCS (MCS 2: QPSK, 1/2), different levels of SNR have been tested. As shown in Fig. 4.6, the two solutions exhibit similar throughput for SNR below 8.5 dB, where a high packet loss is observed. At higher SNR, i.e., above 10 dB, the maximum achievable throughput is instead limited by the physical data rate. Throughput saturates at around 2 Mb/s using the SDR transceiver – a value that is more than 50% lower than the Unex card throughput. Consequently, for the sake of fair comparison between the IEEE 802.11p and the LTE technology, the measured packet loss of the SDR IEEE 802.11p transceiver has been scaled so that its throughput (and packet loss) performance matches that exhibited by commercial cards.

The testbed implementation setup used to evaluate CAREM is shown in Fig. 4.7. The edge host and the MTs are installed in Ubuntu 18.04 systems. The edge host system is equipped with an Intel i7-7700HQ 4-core CPU and 16 GB of DDR4 RAM, while the one used for the MTs integrates an Intel i7-8550U 4-core CPU and 16 GB of DDR4 RAM. Each Ubuntu system is connected to two ETTUS Universal Software Radio Peripheral



(a)



(b)

Fig. 4.8 Variation of context with time: SNR (a) and per-MT buffer state for 3 Mbps traffic load (b). SNR being an independent variable varies randomly, while the buffer state reduces close to zero as the algorithm learns to select better actions.

(USRP) B210 boards, one for LTE and the other for IEEE 802.11p, using USRP Hardware Driver (UHD) v3.15.

4.6 Performance Evaluation

In this section, we first detail the experimental settings of the testbed under which we derived our performance results. We then assess the performance of CAREM by showing the convergence of reward values and the behavior of the KPIs in response to the action selection. Finally, we present a comparison of CAREM with the closest competitive technique in [109], a relatively simpler contextual Bandit (CB) approach, and the standard LTE cellular system.

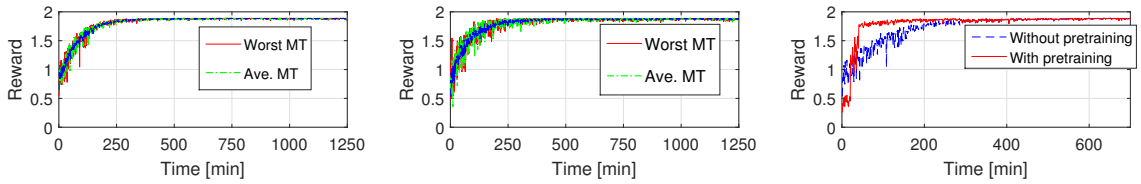
4.6.1 Experimental settings

We evaluate the performance of the CAREM framework using our testbed implementation. We consider two cases: (a) $N = 1$, which corresponds to per-slot (i.e., 100-ms) decision making, and $N = 10$, where decision is periodically made every 10 monitoring slots (i.e., every second).

In our performance evaluation, we consider two scenarios, hereinafter referred to as 2-link and 3-link scenario, respectively. In the 2-link scenario, we consider a 10-MHz bandwidth LTE and an IEEE 802.11p link, and 5 MTs connected; the traffic load at the vRPA for each MT is equal to 1 Mbps. In the 3-link scenario, instead, we add a 5-MHz LTE link and consider 3 MTs, each associated with 3-Mbps traffic load, plus 4 MTs, each associated with 1-Mbps traffic load. Fig. 4.8 shows an instance of the time evolution of two context components, the SNR and the buffer state, with the latter referring to the per-MT 3 Mbps downlink traffic load. Here we observe that the SNR is an independent variable and randomly takes values between 8 dB and 21 dB, while the evolution of the buffer state is action dependent, in the sense that over a course of time, as the algorithm is expected to learn to select better actions, the buffer state gradually reduces to zero. Finally, looking at the variation of the KPI values observed in our testbed and whether they meet their respective thresholds, we set such thresholds as per the 3GPP specifications for 5G [116], i.e., at 0.1 s for latency and 0.01 for packet loss.

4.6.2 Convergence analysis

We first focus on the 2-link scenario and evaluate the performance of CAREM in terms of convergence of reward values on time-sequenced context. The variation of reward values as a function of time is depicted in Fig. 4.9(a)-(b), for both $N = 1$ and $N = 10$ decision-making settings, and for best, worst and average MT in the system. Here, the best (worst) MT is the one experiencing the highest (lowest) value of reward averaged over the experiment duration. Instead, average MT is a benchmark scenario wherein during each monitoring slot the reward is evaluated by averaging the reward over all MTs. For both operational settings, we observe that the variation in the convergence behavior is negligible for best, worst, and average MT. Thus, we remark that even in the presence of multiple MTs, each having a different temporal evolution of context vector, the learning of the CAREM framework is efficient. Further, although the variation in reward values if higher for $N = 10$, its convergence is as good as that for $N = 1$. Thus, a



(a) (b) (c)

Fig. 4.9 Convergence of reward for decision making periodicity $N = 1$ (a) and $N = 10$ (b), with reward corresponding to the best (blue), worst (red), and average (green) MT performance. Comparison between CAREM convergence with and without pre-training, for $N = 1$ and worst MT (c). Uniform convergence across different MTs and for different decision periodicities indicating efficient learning. Pre-training helps to achieve faster convergence.

longer decision making periodicity lowers the computation complexity with respect to per-slot decision making without affecting the convergence behavior of the algorithm.

Finally, Fig. 4.9(c) shows how convergence can be further sped up when the RL model is pre-trained, as it is often deemed as required before a model starts operating in real-world systems. To highlight the difference between the cases with and without pre-training, we focus on a smaller time range on the plot x-axis and on the performance of the worst MT. It is worth remarking that the limited difference between the two curves shown in the plot further confirms that CAREM can quickly converge even in absence of pre-training.

4.6.3 2-link scenario: KPIs, throughput, and action selection

The results derived for the 2-link scenario, presented in Fig. 4.10, show that, except for an initial exploration period, the observed KPI values remain below their respective thresholds. This holds for all MTs, as can be seen by observing the curves referring to the best, worst and average MT performance. Compared to the $N = 1$ decision making (Fig. 4.10(a)), the observed packet loss is slightly higher for $N = 10$ (Fig. 4.10(b)), as in the latter case the action executed by CAREM during a decision making interval may not be the optimum choice for all the slots in that interval. No significant degradation however is noticeable for either KPIs, thus suggesting that a larger decision making periodicity can be a viable solution to reduce the computational burden. Further, in case of pre-training (results omitted here for lack of room), CAREM can learn even faster, leading to a very significant reduction of the time during which KPIs are above threshold.

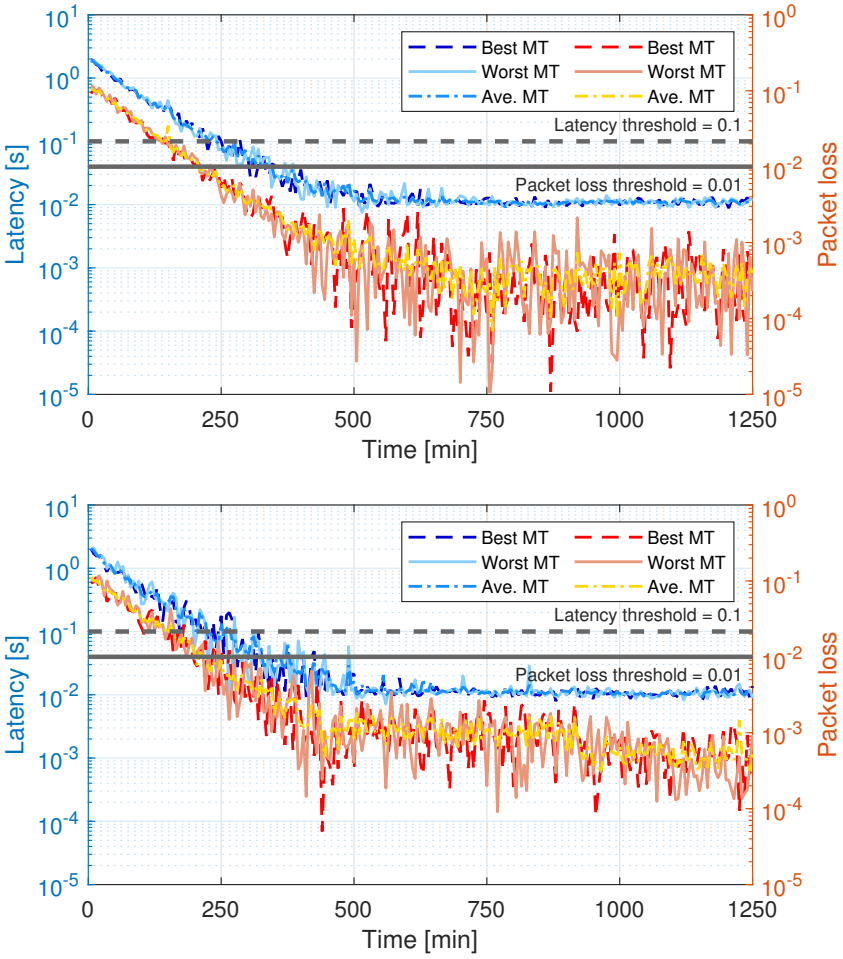


Fig. 4.10 2-link scenario: Time evolution of the latency (blue shades) and packet loss (red shades) KPIs for the best (solid line), worst (dashed line), and average (dashed dotted line) MT performance. KPI thresholds are depicted in gray. Decision making with periodicity $N = 1$ (a) and $N = 10$ (b). KPI requirements are met in all cases.

Next, we look at the throughput corresponding to the best, average, and worst MT performance. Even if throughput is not one of the KPIs targeted by CAREM, it is clearly correlated with latency and packet loss, and it is one of the reference metrics considered for the analysis of wireless systems. As shown in Fig. 4.11, for all MTs the throughput matches the data traffic they are supposed to receive (i.e., 1 Mbps), thus confirming the effectiveness of CAREM.

Finally, Fig. 4.12 depicts the frequency with which CAREM selects the values of MCS (Fig. 4.12(a) and Fig. 4.12(b) for $N = 1$ and $N = 10$, resp.) and resource allocation (Fig. 4.12(c) and Fig. 4.12(d) for $N = 1$ and $N = 10$, resp.). We remark that, with the aim to combine results for different capacity-constrained links, the resource allocation

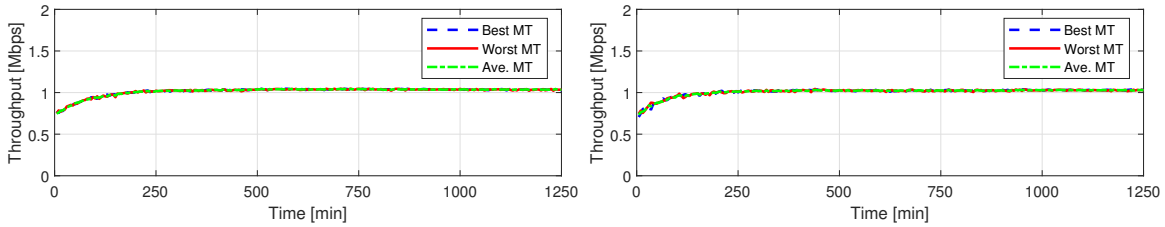


Fig. 4.11 2-link scenario: throughput in the case of best (blue), worst (red), and average (green) MT performance, for $N = 1$ (a) and $N = 10$ (b). For all the MTs, the measured throughput matches the offered traffic load.

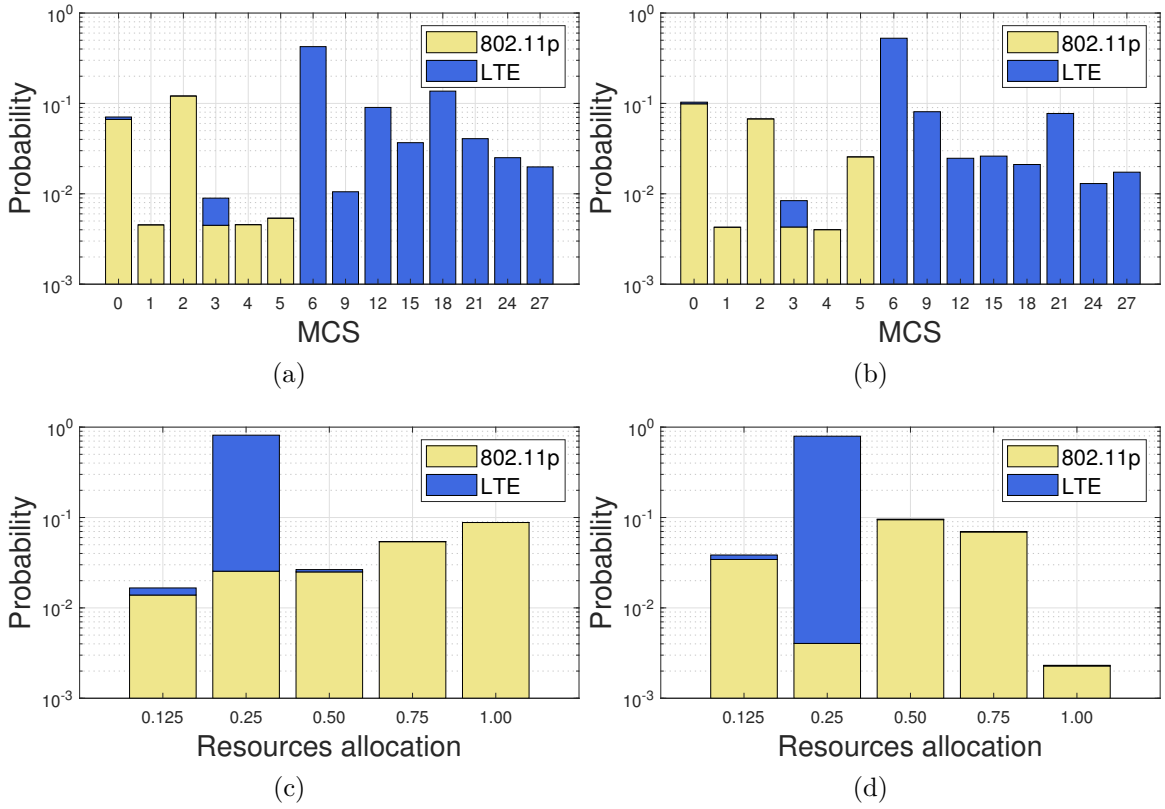


Fig. 4.12 2-link scenario: MCS selection for $N = 1$ (a) and $N = 10$ (b), and fraction of per-MT resource allocation for $N = 1$ (c) and $N = 10$ (d). Bars in yellow and blue refer to IEEE 802.11p and 10-MHz LTE, respectively. The distribution of the selected MCSs ((a), (b)) indicates that CAREM can deal well with the correlation between MCS and the MTs' state, while that of the link utilization ((c), (d)) highlights that LTE is the most selected link.

fraction on the plots x-axis is expressed as the ratio of resources allocated for each MT on a given link, to the number of available radio resources thereof.

By looking at the results obtained for $N = 1$ (Fig. 4.12(a) and (c)), one can observe that the choice of the MCS value varies depending on the experienced SNR as well as the link, as IEEE 802.11p supports only a subset of the MCS values that can instead be selected in LTE. On the latter link, it is quite evident a preference for relatively higher MCSs (greater or equal to 6), since, given a number of allocated RBs, such values allow for higher throughput.

With regard to link utilization in Fig. 4.12(c), the results reflect what the intuition suggests: LTE is the most used link. Indeed, since the 10-MHz LTE link offers a higher capacity than IEEE 802.11p, the latter likely accommodates the traffic for only one MT at the time, while LTE is used for multiple MTs. Also, while the allocated air time on IEEE 802.11p varies depending on the number of packets to be transmitted to the MT using that link and the adopted MCS, the most likely number of RBs allocated on LTE for each accommodated traffic flow is equal to 0.24 (i.e., 12 RBs). Indeed, with the aim to best meet the target KPIs, CAREM tries to allocate as many resources as possible for the served MTs.

At last, looking at Fig. 4.12(b) and (d), which refer to $N = 10$, we note that, although in this case the actions executed by CAREM may not be the optimum choice for all slots in a decision interval, its average resource utilization is almost at par with $N = 1$. This is further confirmed by the fact that the average resource utilization is found to be 14.91% and 19.86% for IEEE 802.11p and LTE (resp.) when $N = 1$, and 11% and 19.87% for IEEE 802.11p and LTE (resp.) when $N = 10$.

4.6.4 3-link scenario: KPIs and action selection

To further show the scalability of CAREM, we now focus on the 3-link scenario and present the results in Fig. 4.14 for $N = 1$. In particular, the plot in Fig. 4.13 shows the latency (in shades of blue) and packet loss (in shades of red) for the best, worst and average MT performance. After the initial learning, both KPIs meet the target value, except for the packet loss value of the MT experiencing the worst performance, which exceeds the threshold in very few time instants.

Fig. 4.14(a)–(b) depict instead the probability with which the different actions are selected, referring to the MCS values and the per-MT fraction of radio resource allocation (resp.) on the three available links. Most of the trends observed for the 2-link case are confirmed. For 5-MHz LTE, we notice that MCS values smaller than 27 are mostly selected, as the traffic load allocated on that link is lower than on 10-MHz LTE, hence less

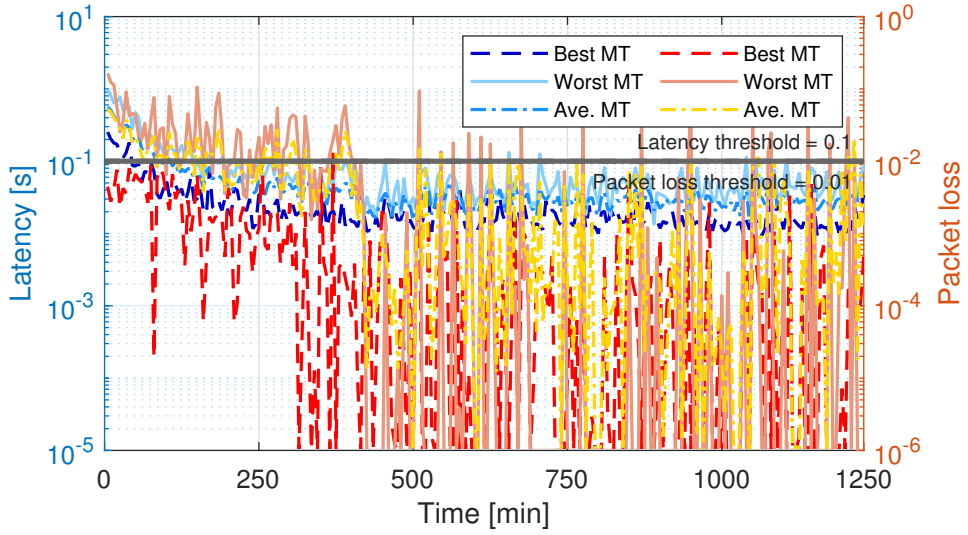
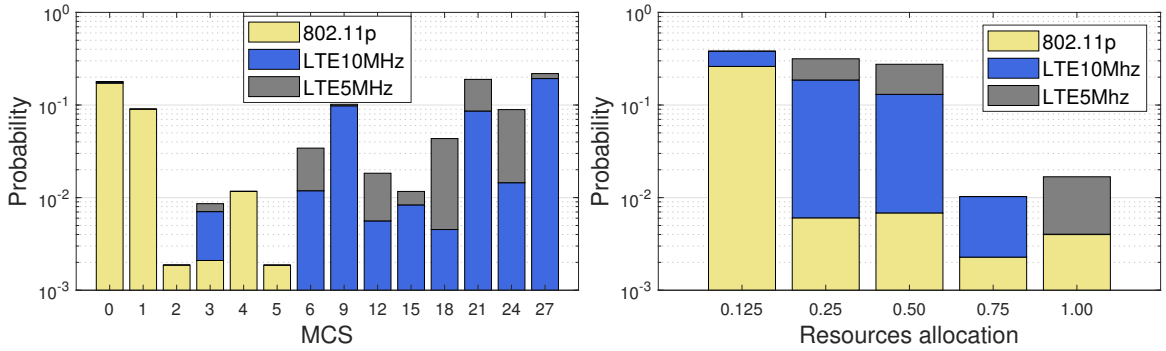


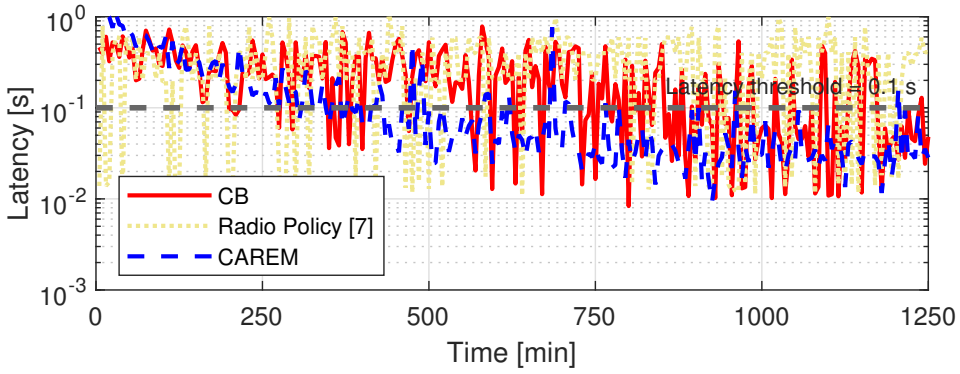
Fig. 4.13 3-link scenario for $N = 1$. KPIs time evolution (latency in shades of blue and packet loss in shades of red) for the best (solid line), worst (dashed line), and average (dashed dotted line) MT performance (KPI thresholds are in gray). As the learning converges, average KPI thresholds are always met.



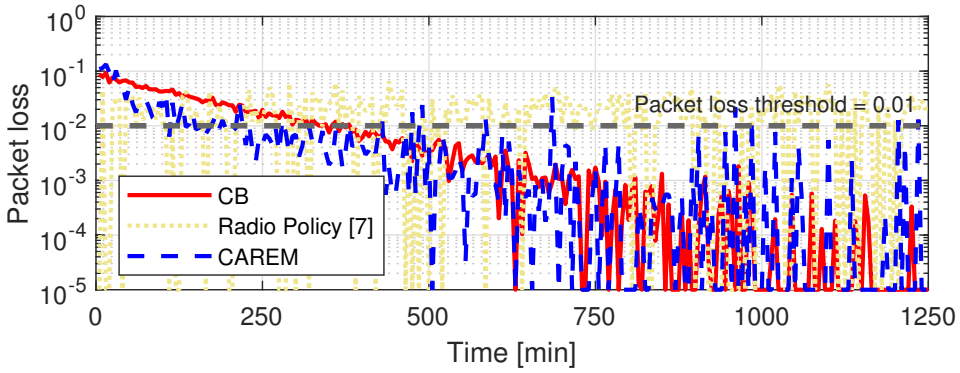
(a) (b)

Fig. 4.14 3-link scenario for $N = 1$. Probability of selecting MCS (a) and resource allocation fraction (b). In (a) and (b), bars in yellow, blue and gray refer to IEEE 802.11p, 10-MHz LTE, and 5-MHz LTE, respectively. Trends concur with 2-link scenario.

efficient, yet more robust, schemes are preferred. As for resource allocation on the 5-MHz LTE link, we notice that the unitary value (i.e., 25 RBs) is selected with significantly higher probability, given the smaller link capacity.



(a)

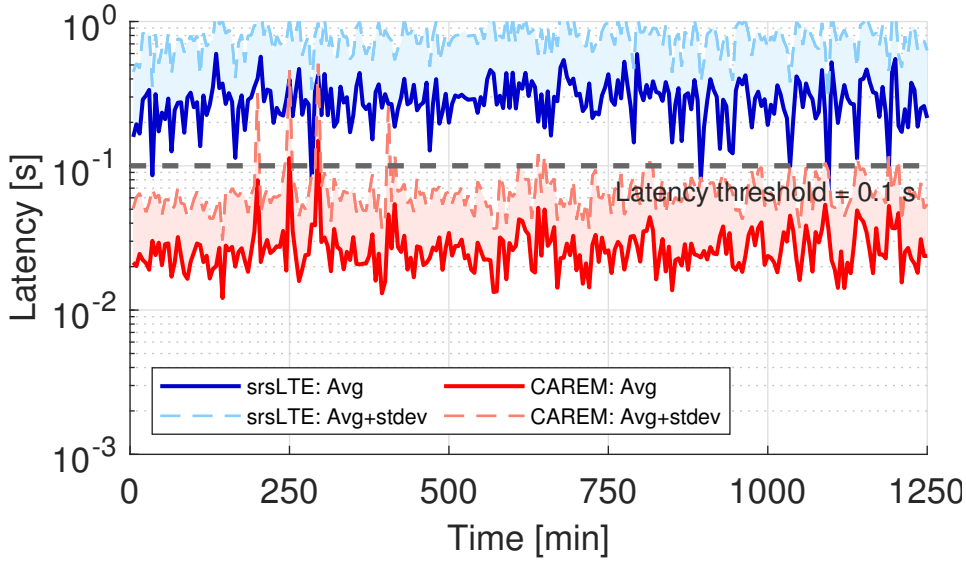


(b)

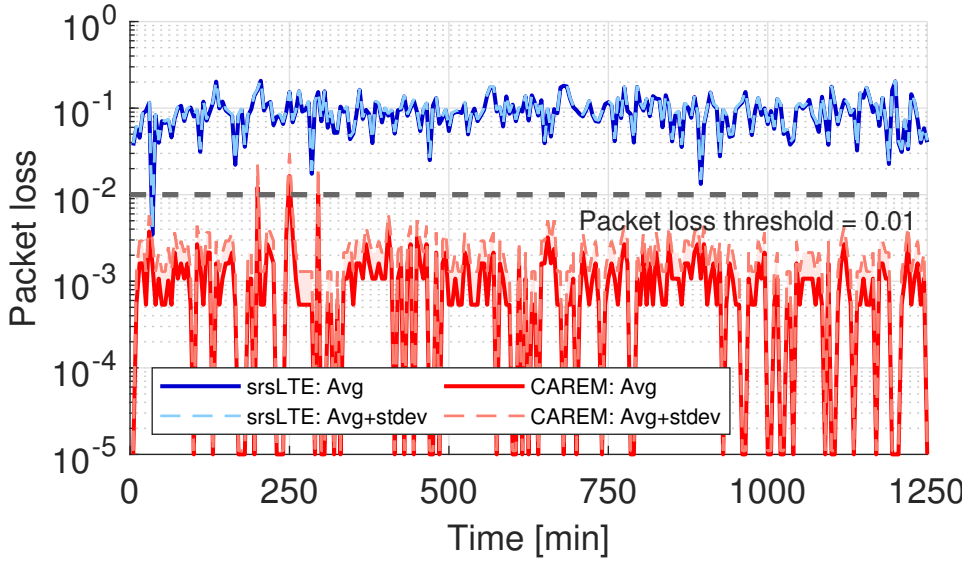
Fig. 4.15 KPIs time evolution for CAREM, radio policy [109], and contextual bandit (CB), in a one LTE-link scenario: latency (a) and packet loss (b). CAREM only can fulfill latency requirements, while target packet loss is met by CAREM and CB.

4.6.5 Comparative performance analysis

Here, we compare the performance of CAREM with the closest competitive approach in [109], a relatively simpler contextual bandit (CB) approach [150], and standard LTE. For the latter, we rely on the srsRAN implementation, which is compliant with LTE Rel. 9. As for [109] (see also Sec. 4.2), after relaxing the computation constraints, the radio policy targets the selection of suitable MCS for fast and reliable packet transmission over an LTE link. Note that heterogeneous links are considered neither in [109] nor in LTE. Hence, for the sake of fair comparison, we focus on the MCS allocation problem over a 10-MHz LTE link, and consider two MTs, each receiving a 3-Mbps traffic flow. In both [109] and CB, the classifier is trained using the dataset obtained from our testbed implementation of the LTE vRAN, as discussed in Sec. 4.5. Finally, in CB the same setting as in CAREM is used for the ϵ decay.



(a)



(b)

Fig. 4.16 KPIs time evolution for one LTE link, under CAREM (red lines) and LTE (blue lines). Latency (a) and packet loss (b) averaged over two MTs (solid lines), and average plus standard deviation (dashed lines). CAREM outperforms LTE under high traffic load, owing to a more extensive exploration of different MCS values.

Fig. 4.15 presents the comparison of the KPI variation over time for $N = 1$ for CB, the radio policy in [109] and CAREM. We observe that, under CAREM and after the initial learning time, the variation of the latency in Fig. 4.15(a) is essentially always below threshold. The target latency value instead cannot be met using the MCS selection from the CB scheme or the radio policy in [109]: on average, CAREM provides a 65%

improvement with respect to CB and about one order of magnitude relatively to [109]. In case of CB, this is attributed to the fact that being a relatively simpler scheme, it is unable to capture the associative aspect of context-action mapping when the choice of current action influences the future values of the context. Further, the radio policy in [109] is trained using a loss function that minimizes the decoding error probability of the packet without accounting for latency as a criteria, consequently leading to larger latency values. This confirms that, thanks to the ability of SARSA to learn the best state-action trajectories, CAREM avoids taking high risk actions and incurring undesirable network states, which makes it a better match for dynamic scenarios than, e.g., a CB approach.

Further, from the packet loss variation in Fig. 4.15(b), we note that CAREM and CB provide similar performance, which is almost one order of magnitude better than that of the radio policy in [109]. Indeed, the latter aims at limiting the bit error rate at the physical layer, which may lead to different actions with respect to such policies as CAREM targeting instead a desired level of packet loss at the MAC layer.

At last, Fig. 4.16 compares the performance of CAREM against LTE, as implemented in srsRAN, again with two MTs and 3 Mbps traffic load. For clarity of the plots, we depict the value of the KPIs averaged over the MTs, along with the average value plus standard deviation. Note that, given the considered SNR pattern, the offered traffic load exceeds the maximum throughput that the standard LTE link can provide. Under such conditions, CAREM can instead support the considered traffic load with the target latency (Fig. 4.16(a)) and packet loss (Fig. 4.16(b)) values, although it has been designed primarily to manage heterogeneous links in vRANs. By looking at the physical layer metrics (omitted here for lack of room), we noticed that the better performance of CAREM is due to its ability of exploring various actions for a given context, hence more possible values of MCS. Some of them are never selected by standard LTE, but they can actually lead to a higher reward and are therefore chosen by CAREM.

4.7 Conclusions

We have proposed CAREM, a novel RL-based framework that efficiently allocates radio resources in terms of link, MCS, RBs and airtime for packet transmissions in heterogeneous vRANs. The choice of the RL algorithm, actions, and reward function has been made so that the resource utilization is optimized with respect to dynamic and non-stationary environments, with limited computation efforts. Importantly, we have provided a proof-of-concept of our solution, by developing a testbed that leverages an LTE and an IEEE

802.11p SDR implementation. We have evaluated CAREM under different operational settings, with different decision-making periodicity, number of links, number of MTs connected, and traffic load. The results show that, as the learning process of the model saturates, actions are chosen so that both the observed KPIs, latency and packet loss, always satisfy their target values. Further, it outperforms state-of-the-art solutions as well as standard LTE, as implemented in the srsRAN framework. In comparison to the closest competitive scheme in [109] and LTE, both the latency and packet loss observed with CAREM are of about one order of magnitude lower, while CAREM provides a 65% latency improvement relative to contextual bandit. Finally, we remark that CAREM is a promising starting point for the development of heterogeneous networks, where the advantages of different radio technologies can be fully exploited to maximize the performance and the robustness of the network. Additionally, it effectively addresses the need for a solution that can swiftly adapt to the underlying channel-network dynamics for context-aware radio resource allocation in heterogeneous vRANs.

In the next chapter, we will reuse and extend some of the concepts at the base of CAREM to develop a new RL framework for the management of not just only radio, but also compute resources. We will consider a scenario where a single LTE vRAN and a video transcoder application run on the same node and thus on shared resources, so a careful allocation of such resources will be required to avoid disrupting the services provided by the two applications. With VERA, we address the dynamic resource needs of the applications providing fair and efficient decisions through an additional framework block that performs Pareto analysis. Finally, we will prove the performance of our solution through numerical and testbed results.

Chapter 5

Fair and Scalable Orchestration of Edge Services Resources

The combination of service virtualization and edge computing allows for low latency services, while keeping data storage and processing local. However, given the limited resource availability at the edge, a conflict in resource usage arises when both virtualized user applications and network functions need to be supported. Further, the concurrent resource request by user applications and network functions is often entangled, since the data generated by the former has to be transferred by the latter, and vice versa. In this chapter, we first show through experimental tests the correlation between a video-based application and a vRAN. Then, owing to the complex involved dynamics, we develop a scalable reinforcement learning-based framework for resource orchestration at the edge, which leverages a Pareto analysis for provable fair and efficient decisions. We validate our framework, named VERA, through a real-time proof-of-concept implementation, which we also use to obtain datasets reporting real-world operational conditions and performance. Using such experimental datasets, we numerically demonstrate that VERA meets the KPI targets for over 96% of the observation period and performs similarly when executed in our real-time implementation, with KPI differences below 12.4%. Further, its scaling cost is 54% lower than a centralized framework based on deep-Q networks.

While VERA shares some low-level implementation details with CAREM, presented in the previous chapter, it is a substantially different framework. Here, the focus shifts from the management of only the heterogeneous RAN to the management of both radio and user applications. Moreover, to guarantee fairness between different applications, the Pareto block, which in CAREM takes care of adjusting the radio resource allocation to

satisfy the capacity constraint, has been improved to consider also the compute resource, namely the vCPU.

Part of the work described in this chapter has been already published in S. Tripathi, C. Puligheddu, S. Pramanik, A. Garcia-Saavedra and C. F. Chiasserini, "VERA: Resource Orchestration for Virtualized Services at the Edge," *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1641-1646, doi: 10.1109/ICC45855.2022.9838935,

Part of the work described in this chapter has been submitted for possible publication in *IEEE Transactions on Mobile Computing*.

5.1 Introduction

Network Function Virtualization (NFV) and edge computing are disrupting the way mobile services can be offered through mobile network infrastructure. Third parties such as vertical industries and over-the-top players can now partner up with mobile operators to reach directly their customers and deliver a plethora of services with substantially reduced latency and bandwidth consumption. Video streaming, gaming, virtual reality, safety services for connected vehicles, and IoT are all services that can benefit from the combination of NFV and edge computing: when implemented through virtual machines or containers in servers co-located with base stations (or nearby), they can enjoy low latency and jitter, while storing and processing data locally.

The combination of NFV, edge computing, and an efficient radio interface, e.g., O-RAN [151], is therefore a powerful means to offer mobile services with high quality of experience (QoE). However, some important aspects have been overlooked. On the one hand, user applications are not the only ones that can be virtualized: network services such as data radio transmission and reception are nowadays virtualized and implemented through Virtual Network Functions (VNFs) as well [152–156]; and both types of virtual services, user's and network's, may be highly computationally intensive. On the other hand, it is a fact that computational availability at the network edge is limited [157]. It follows that **in the edge ecosystem, user applications and network services compete for resources, hence designing automated and efficient resource orchestration mechanisms in the case of resource scarcity is critical.**

Further, looking more closely at the computational demand of virtualized user applications and at that of network service VNFs, one can notice that they certainly

depend on the amount of data each service has to process, but they are also entangled [158]. As an example, consider a user application at the edge and (de-)modulation and (de-)coding functions in a virtualized radio access network (vRAN). For downlink traffic, the application bitrate determines the amount of data to be processed by the vRAN; on the contrary, for uplink traffic, the data processed by the vRAN is the input to the application service. A negative correlation, however, may also exist: the more data compression is performed by a user application, the higher its computational demand, but the smaller the amount of data to be transmitted and the less the computing resources required by the vRAN. In a nutshell, **a correlation exists between the amount of data processed/generated by virtual applications at the edge and network services VNFs, and such correlation can be positive or negative depending on the type of involved VNFs.**

Related joint resource problems have been addressed before [158] albeit ignoring the complex relationship between all system parameters and context variables and, therefore, making simplifying assumptions that do not work in practice [159]. Our experimental analysis in Sec. 5.3 indeed unveils such complex couplings. For instance, contextual features of the wireless link, such as signal-to-noise-ratio (SNR), radio policies such as the modulation order and coding scheme (MCS) selected by the vRAN, and the computing resources allocated to the vRAN have non-linear effects on the resulting latency and, as a result, on the amount of buffering required by a video-based service. **These issues impair the use of modeling techniques traditionally used for optimal resource allocation in practical situations.**

To capture such trends and relations, and withstand the above challenges, we design a flexible and scalable framework, called VERA (Virtualized Edge for Radio and user Applications), leveraging a model-free reinforcement learning (RL) approach. Like us, other authors also use model-free approaches to address the aforementioned complex relationship across different aspects of a vRAN [160, 161]. However, extending such approaches to a multi-service scenario falls into serious scalability issues. To address this problem, we adopt a distributed multi-agent learning approach. Not surprisingly, designing a multi-agent learning framework where the actions of individual agents must collectively satisfy the hard capacity constraints characteristic of mobile edge platforms is inherently hard. Inspired by [162] and other literature on autonomous driving, we decompose the policy function into two stages; the first stage produces *greedy* actions based on the context collected from the environment while the latter refines these actions to enforce hard constraints. Unlike previous work in other settings, however, our use case

requires some notion of fairness when enforcing these constraints. To address this, we design a novel Pareto component that guarantees a fair Pareto-efficient solution.

In summary, we provide the following contributions:

- First, we present *experimental evidence* for the above observations, through a containerised edge and a software-defined-radio (SDR)-based vRAN testbed;
- Then, given the many interplaying factors and their complex interaction, *we introduce an RL model* for an effective, joint allocation of computing resources for user applications and vRAN at the edge;
- To aid scalability, we resort to *distributed learning agents*, which we complement with a Pareto analysis for a fair and efficient decision-making, whenever resource utilization is constrained to a given budget;
- We show the excellent performance of the VERA framework in terms of convergence as well as its ability to closely meet the target KPIs of all services in resource-constraint scenarios. Specifically, we show that, post convergence, VERA meets the KPI targets for more than 96% of the observation period, and that it performs similarly when executed in our real-time proof-of-concept implementation, with KPI differences below 12.4%. Further, we remark that the scaling cost of VERA is 54% lower compared to a competitive centralized framework using deep Q networks.
- Finally, we validate our approach by implementing VERA on our testbed and showing that its performance is preserved when interacts with a real system in real-time.

We remark that, to our knowledge, we are the first to address the allocation of a **common pool of edge resources to different**, competing, virtualized services through distributed learning, and to tackle the non-trivial correlations existing among the behaviors of such services *in a scalable manner*. Moreover, not only VERA can swiftly adapt to time-varying network conditions and application traffic, but it also controls the settings of both user applications and vRAN, selecting at each decision step a fair Pareto-efficient solution.

The rest of the chapter is organized as follows. Sec. 5.2 introduces the reference scenario and system architecture. Our experimental analysis is presented in Sec. 5.3, highlighting the relevant components of the environment contextual information, the target KPIs, and the driving factors determining the system behavior. Sec. 5.4 describes

the VERA framework, Sec. 5.5 presents our proof-of-concept and testbed. Sec. 5.6 shows VERA's performance, compares it against a state-of-the art alternative, and validates our approach by running the complete framework on our testbed. Finally, Sec. 5.7 discusses related work, and Sec. 5.8 draws our conclusions.

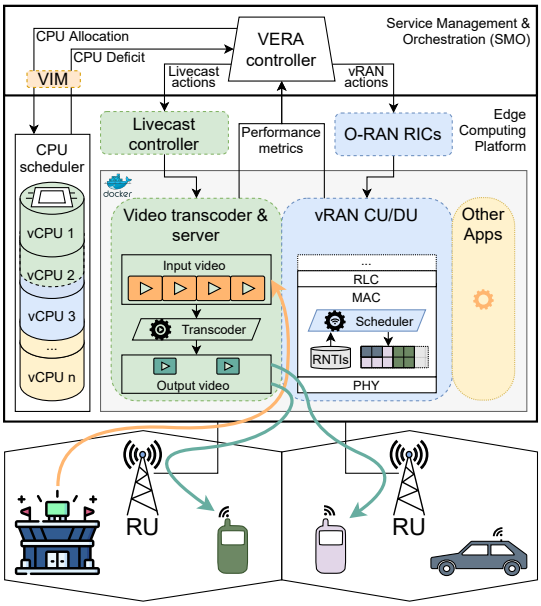


Fig. 5.1 Virtualized user application and vRAN at the edge: system scenario and reference use case

5.2 Reference Scenario and System Architecture

The system architecture and reference scenario under study are depicted in Fig. 5.1. For clarity, we focus on one type of vertical service and one virtual base station (vBS), implemented within an edge computing platform. Note that, as explained in Sec. 5.4, VERA is highly scalable and can effectively handle multiple coexisting services as well as multiple coexisting vBSs (and/or network slices therein).

As sample use case, we consider a livecast service representing a live video recording of an event occurring, e.g., at a stadium, that is broadcast to multiple mobile users located therein or in the nearby area. The high-quality source video is processed within an edge computing platform through a standard video transcoding service. We remark that deploying services like livecast at the edge entails that video traffic is produced and consumed locally, thus saving network bandwidth. Further, it can leverage a multi-access edge computing (MEC) platform and the associated Radio Network Information Service (RNIS) [163], which, through feedback-based multicast, allows the use of estimated radio

channel conditions for real-time tuning of the video coding parameters [164]. Finally, emerging interactive services for which video streaming is one of the essential components, e.g., crowdcast, augmented reality and mobile gaming, have such strict latency constraints that only an edge-based architecture can meet.

In addition to the livecast service (as well as, possibly, other user services running at the edge), the edge computing platform hosts vBS functions, central unit (CU) and/or distributed unit (DU), which are jointly controlled by the VERA controller. As depicted in Fig. 5.1, the VERA controller is deployed in the Service Management & Orchestration (SMO) platform, and interacts with both O-RAN intelligent controllers (RIC) to configure the vBS functions, the edge service controllers (in this case the livecast controller), and the NFV virtual infrastructure manager (VIM) to configure the CPU schedulers (see O-RAN specification [151]). In this way, VERA's workflows (data collection and decision making) are fully compliant with O-RAN's machine learning procedures [151]. Indeed, VERA continuously monitors the state of the vRAN and the livecast application (hereinafter also referred to as *services*), as well as the overall usage of computing resources in the edge platform. Then, it uses such observations to compute the values of the operating parameters for both livecast and vRAN, which, given the available computing and networking resources, meet both the application and vRAN KPI targets.

5.3 Experimental analysis

The system architecture described in Sec. 5.2 has been recreated in a smaller scale in our testbed for the development and testing of VERA. The main components are the edge computing platform, and the user equipments (UEs), which communicate by means of an LTE vRAN implemented using the srsRAN suite [165]. The edge platform runs two Docker containers implementing, respectively, the livecast and the vRAN service, which consume the edge resource pool.

Our experimental vRAN testbed includes one srseNB instance, i.e., the LTE vBS, and two srsUE instances, which represent the recipients of the video content livecast by the vBS. The livecast application consists of a live video streaming transcoder, implemented with ffmpeg[1], and a server, based on ffserver[2]. It receives the high-quality original video and transcodes it through the recent VP9 codec using the bit rate and frame rate settings provided by VERA. Then, the transcoded video is served to the UEs, each running a

[1] https://ffmpeg.org/
[2] https://trac.ffmpeg.org/wiki/ffserver

player that streams, decodes, and plays the video. The radio and livecast services are connected to VERA through a dedicated API, used to dynamically set radio and livecast operating parameters and retrieve performance measurements. VERA also interacts with the edge computing platform operating system and the Docker daemon to monitor and allocate computing resources to the services. More details about our testbed are provided in Sec. 5.5.2.

We then use our testbed to analyze empirically the trade-offs between different actions configurable in our system, given different contexts. To ease the analysis, we focus on a single user, but we note that VERA supports multiple users and we evaluate VERA with multiple users in later sections.

We start by defining a contextual feature that characterizes the videos being delivered by the livecast service:

- **Context 1:** Video input bit rate, input frame-per-second (FPS) rate, and input resolution;

and another feature that indicates the computational demand required by the livecast service:

- **Context 2:** Video CPU throttled time. This feature gives us an indication of the processing pressure associated with the requested video, which is a footprint distinguishable across videos, and hence impacts the overall performance and the choice of appropriate actions.

We also define three sets of actions for our livecast service, some of which re-encode each video accordingly:

- **Action 1:** Video output bit rate;
- **Action 2:** Video output FPS rate; and
- **Action 3:** CPU resources allocated to the livecast service;

and a KPI that we can use to estimate the quality of the video being delivered, as set forth below:

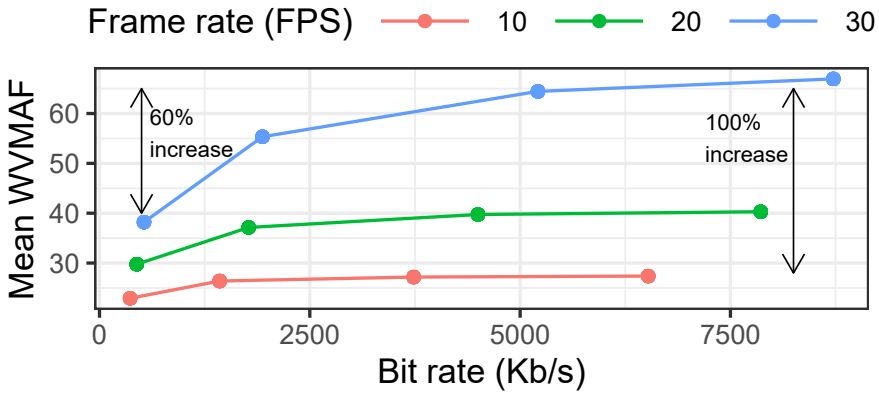


Fig. 5.2 Video quality (WVMAF) for different livecast service configurations (output FPS rate and bit rate)

- **KPI 1:** Weighted Video Multimethod Assessment Fusion (WVMAF). It is based on the VMAF, a widely used objective metric to assess video quality, which provides a score between 0 (worst) and 100 (best) *per video frame*. The score is computed by aggregating different components such as Visual Information Fidelity, Detail Loss Metric, or Mean Co-Located Pixel Difference (the interested reader can find further details in [166]). However, because VMAF assesses the quality of individual video frames only, it is not helpful to measure the *smoothness* of a video, which is well known to impact the perceived quality. To weight this in, we amplify or attenuate the measured VMAF of each frame by the ratio between the output frame rate and the input frame rate, and we refer to this metric as WVMAF.

Fig. 5.2 shows the mean WVMAF score for a wide variety of VP9-encoded videos with different output FPS rates and bit rates, CPU throttle time, and (for simplicity) the same resolution. The results are intuitive: higher-bit-rate and higher-FPS videos have higher WVMAF. It is interesting to observe that the frame rate setting has a larger impact on WVMAF (100% increase for high video bit rate) than the target bit rate (60% increase for 30 FPS), which moreover shows diminishing returns. This is due to the weight that amplifies the measured VMAF, which increases the score when higher frame rates are used.

A second relevant KPI to assess the perceived QoE of the livecast service is:

- **KPI 2:** Video player buffering. Information about the client's buffer state, which stores video frames for playout, is a good estimator of the user's QoE [167]. Specifically, when the buffer size gets close to zero, the user's player may stutter, which resorts in a low level of QoE.

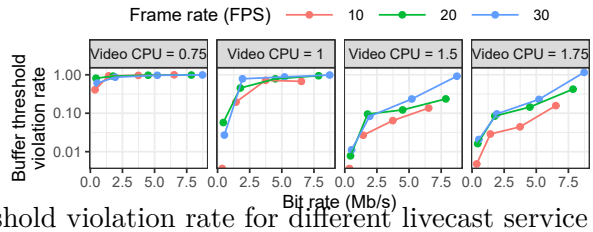


Fig. 5.3 Buffering threshold violation rate for different livecast service configurations and CPU allocations

To assess this KPI, we select a threshold equal to 0.5 seconds of video buffered at the client's video player, and report in Fig. 5.3 the frequency that such threshold is violated for the same set of videos used before and for different combinations of actions. In this case, the target bit rate and the amount of CPU resources, measured in units of virtual CPU (vCPU) assigned to the service, have a much larger impact on this KPI than before (ruling ranges of this KPI that span 2 orders of magnitude). Note the logarithmic scale in the y-axis, which indicates a non-linear behavior.

Next, we focus on the vRAN service, and define two more actions related to it:

- **Action 4:** CPU resources allocated to the radio; and
- **Action 5:** A Modulation and Codign Scheme (MCS) policy. This policy follows that used in [159] and imposes an upper bound on the MCS eligible by the base station, which helps to control the computational demand of the radio service;
- **Action 6:** Bandwidth allocated to each UE, measured as the aggregate amount of radio Resource Blocks (RBs);

and three relevant contextual features:

- **Context 3:** Radio CPU throttled time, which is the radio counterpart of Context 2;
- **Context 4:** Signal-to-noise-ratio (SNR), which is a common feature used to estimate the quality of a wireless channel and in turn bounds its capacity; and
- **Context 5:** Network load, which corresponds to the offered load that the vBS has to process, generated by the applications deployed at the edge (such as our livecast service) and background data related to the mobile network.

Concerning radio KPIs, we first define:

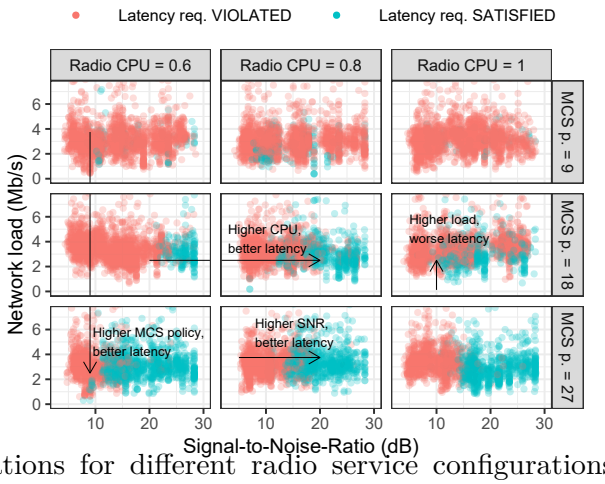


Fig. 5.4 Latency violations for different radio service configurations, contexts, and CPU allocations

- **KPI 3:** Radio latency. This is the latency associated with the data transmitted successfully over the air.

To analyze this KPI, we plot in Fig. 5.4 every data frame that violates/meets a latency threshold equal to 150 ms with red/blue colored dots when the vBS has to deliver randomly chosen videos from our set. We present these as functions of two of the radio contextual features (SNR and network load) and for different combinations of actions. Correlations between context, actions, and latency are evident. For instance, a higher MCS policy show a consistent improvement in latency performance, which however requires more computing resources. We observe a similar behavior when allocating a higher number of RBs (results omitted to reduce clutter).

We then define one last KPI associated with the radio service:

- **KPI 4:** Packet loss rate, which measures the number of unacknowledged TCP segments due to corruption on the radio link.

Fig. 5.5 shows this KPI for all the videos in our set as a function of the SNR (Context 4) and for different MCS policies (Action 5). In red, we mark those samples that exceed 1% packet loss. Obviously, the sample set is highly biased towards 0 packet loss rate. However, there are a number of video scenes that cause packet losses, and these are highly correlated with SNR and our MCS policy in a non-trivial manner as shown by the plot. For instance, we can observe that higher MCS policies yield considerably lower packet loss rate. Note that this gain comes from the extra wireless capacity granted by larger MCS policies and not from the reliability of a given MCS that is selected automatically

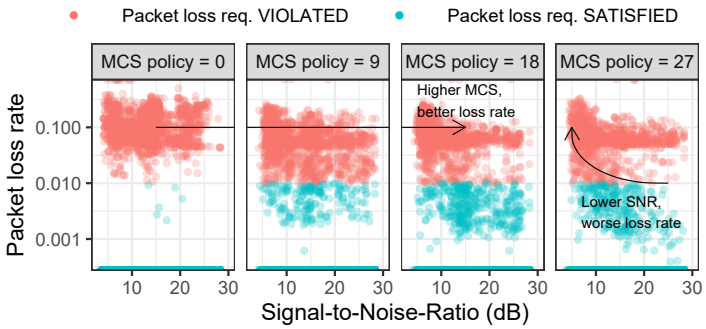


Fig. 5.5 Packet loss rate for different vRAN configurations and contexts

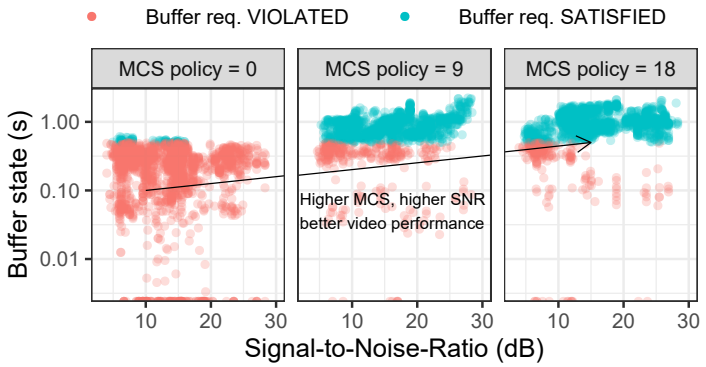


Fig. 5.6 Livecast performance (buffer state) for different vRAN configurations and contexts

by the radio scheduler: we simply impose a restriction on the set of eligible MCSs. Moreover, better SNR provides also better performance, which is intuitive. However, this relationship is non linear (note the logarithmic y axis). Likewise, the dependency between packet loss and the number of allocated RBs is also monotonic, i.e., high packet losses are observed with overly low RB allocations (like before, we omit these results to be concise).

To conclude our experimental analysis, we plot in Fig. 5.6 a livecast KPI (buffer state) as a function of SNR (radio context) and MCS policy (radio action). Evidently, the buffer dynamics of the client's video player are highly correlated with the context and the actions performed over the livecast service. This proves that the resource orchestration problem we endeavour into in this chapter is a coupled problem and all these edge services must be optimized jointly.

Conclusion: It becomes evident that the support of different applications in the edge platform leads to complex inter-dependencies between system parameters, context-action variables and KPIs, thereby making optimum resource allocation a challenging task. To this end, we propose the VERA framework, which is completely data-driven, and, hence,

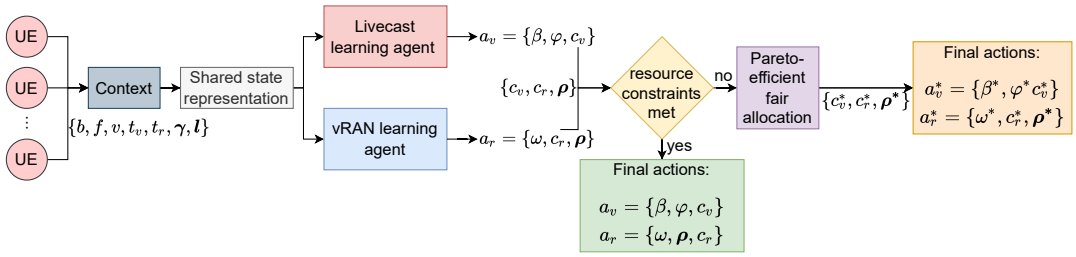


Fig. 5.7 Structure of the VERA framework

a good match for flexible and effective decision making in virtualized environments, despite the system complexity.

5.4 The VERA Framework

The VERA framework is designed using a model-free RL approach. It includes distributed learning agents, each corresponding to a service in the edge platform, which simultaneously make decisions for the allocation of radio and computing resources as well as tune service-specific operating parameters. This design choice is key to attain *a scalable solution.* These decisions are hereafter collectively referred to as a resource allocation policy, which consists of two development stages:

- In the first stage, each RL agent makes decisions based on the shared context representation to obtain a greedy resource allocation policy;
- In the second one, greedy policies from all RL agents are collated and further refined in view of the feasibility of the chosen actions to obtain a Pareto-efficient fair resource allocation policy.

The structure of the VERA framework is shown in Fig. 5.7. Decisions are made with periodicity equal to $N \geq 1$ monitoring slots, i.e., an action is selected at the end of every decision window of duration N slots, and it is applicable to the subsequent N monitoring slots. The individual stages are elaborated in the sequel.

5.4.1 Notation

$\mathbb{R}^n$ denotes the set of n-dimensional real vectors. Vectors (usually in column form) are written in bold font, matrices are in upper-case, bold font, and sets are in calligraphic font.

Table 5.1 Notation

Symbol	Description
Context notation	
b	Video input encoding bit rate
f	Video input FPS rate
v	Video input resolution
t_v, t_r	Normalized video and radio (resp.) throttled time
$\boldsymbol{\gamma}$	CQI value
$\boldsymbol{l}$	Livecast network load
Action notation	
β	Video output encoding bit rate
φ	Video output FPS rate
$\boldsymbol{c}$	CPU allocation
ω	vRAN MCS policy
$\boldsymbol{\rho}$	RB allocation
KPI notation	
ζ	WVMAF score
σ	Client's buffer state
λ	Latency
μ	Packet loss rate
$\{\zeta_o^m, \sigma_o^m, \lambda_o^m, \mu_o^m\}$	Observed KPI values by m-th UE
$\{\zeta_t, \sigma_t, \lambda_t, \mu_t\}$	Target KPI values

Subscripts and superscripts denote an element in a vector and elements in a sequence (resp.). E.g., $\langle \boldsymbol{x}^{(t)} \rangle$ is a sequence of vectors with $\boldsymbol{x}^{(t)} = [\boldsymbol{x}_1^{(t)}, \ldots, \boldsymbol{x}_n^{(t)}]^\top$ (superscript $\top$ is the transpose operator). $\boldsymbol{x}_i^{(t)}$ is the i-th component of the t-th vector in the sequence.

5.4.2 Greedy analysis

Since the edge platform may have several services consuming the resource pool, owing to resource sharing, the KPI satisfaction of each is interdependent. We therefore consider a context vector comprising variables pertinent to each service. The context vector is processed through an autoencoder to create a shared context representation that captures the correlation among context variables, as well as reduces the dimensionality of the context vector. Then, each RL agent devises a greedy resource allocation policy by using the same shared context representation and by mapping it onto an action vector such that its long-term cumulative reward from the environment is maximized. Notice that, although the decisions are based on shared context, greedy policies do not ensure that the sum of capacity-constrained resources among all the services does not exceed their maximum capacity. *To solve this issue, we design a Pareto algorithm that allows for feasible and fair resource sharing.* The elements composing the greedy resource allocation policy are introduced below, while the notation we use is summarized in Tab. 5.1.

Context space. As described in Sec. 5.3, the resource allocation for the livecast service is governed by the following contextual information: input bit rate (b), input video FPS (f), and input resolution (v) of the streaming video (i.e., Context 1 in Sec. 5.3).

Besides, to accommodate any backlog in video processing, the normalized CPU throttled time of the livecast application (t_v) in the previous monitoring slot is considered (Context 2).

Likewise, resource allocation for the vRAN is based on normalized CPU throttled time (t_r) (Context 3), the 3GPP-compliant Channel Quality Indicator (CQI) ($\boldsymbol{\gamma}$) reported from UEs to vBS, which is representative of the SNR (Context 4), and the traffic from the livecast application sent over the radio link to the UEs (Context 5), specified by the network load ($\boldsymbol{l}$). Thus, the context vector observed in monitoring slot n ($n = 1, \ldots, N$) can be written as $\boldsymbol{x}^{(n)} \in \mathcal{X},\ \boldsymbol{x}^{(n)} := \{b^{(n)}, f^{(n)}, v^{(n)}, {t_v}^{(n)}, {t_r}^{(n)}, \gamma_1^{(n)}, \cdots \gamma_M^{(n)}, l_1^{(n)}, \cdots l_M^{(n)}\}$.

Further, to extract the correlation between context variables, an autoencoder projects context vector $\boldsymbol{x}^{(n)} \in \mathcal{X}$ onto its latent representation $\boldsymbol{y}^{(n)} \in \mathbb{R}^D,\ \boldsymbol{y}^{(n)} := \{{\boldsymbol{y}_1}^{(n)}, \ldots, \boldsymbol{y}_D^{(n)}\}$ where $D < \dim(\mathcal{X})$. The latent representation $\boldsymbol{y}^{(n)}$ is shared with each RL agent so that its decision process for a given service is informed of the performance of others accessing the resource pool, thus representing a shared context representation. The autoencoder is implemented through a simple feed forward neural network that is activated using rectified linear units in the hidden layers. Note that dimensionality reduction is only one advantage of the autoencoder: indeed, it is primarily used to capture multimodal patterns among context variables, which may not otherwise be evident owing to the system complexity.

Action space. Since services are heterogeneous, we define action space $\mathcal{A} := \{\boldsymbol{a}_k\},\ \forall k \in (1, \ldots, K)$, comprising action vectors each having service-specific action variables. In our reference scenario, $K = 2$, and we associate $k = 1, 2$, respectively, to action vectors for livecast and vRAN. Consequently, $\boldsymbol{a}_1$ comprises the CPU allocated to the livecast application (c_v), i.e., Action 3 in Sec. 5.3, the video output encoding bitrate (β), i.e., Action 1, and the video output encoding FPS (φ), i.e., Action 2.

Conversely, $\boldsymbol{a}_2$ includes the CPU allocated to vRAN (c_r), i.e,. Action 4, the MCS value (ω) defined before as Action 5, and the bandwidth allocated to each UE, $\boldsymbol{\rho} = \{\rho_1, \rho_2, \cdots \rho_M\}$ as defined in Action 6, where M is the maximum number of users supported in the system. Here, the CPU and the radio (RB) resources are capacity constrained, i.e., $c_v + c_r \leq B_c$ and $\rho_1 + \rho_2 + \cdots \rho_M \leq B_\rho$, where B_c and B_ρ are, respectively, the total available CPU and number of RBs that can be allocated. To avoid clutter, we replace c_v and c_r with a generic c_k that denotes the CPU allocated to service k, and let

ρ_m be the number of RBs allocated to UE m. Mathematically,

$$\boldsymbol{a}_k = \begin{cases} (\beta, \varphi, c_k), & \text{if } \ k = 1, \\ (\omega, c_k, \boldsymbol{\rho}), & \text{if } k = 2\,. \end{cases} \tag{5.1}$$

Next, we discretize the quantity of capacity-constrained resources that can be allocated, and map each feasible combination of action variables during the n-th monitoring slot into an action index $\boldsymbol{a}_1^{(n)} := \{1, 2, \ldots, N_\beta \cdot N_\varphi \cdot N_{c_v}\}$ and $\boldsymbol{a}_2^{(n)} := \{1, 2, \ldots, N_\omega \cdot N_{c_r} \cdot N_{\boldsymbol{\rho}}\}$, where N_i is the number of elements in the discretized version of action variable $i = \{\beta, \varphi, \omega, c_v, c_r, \boldsymbol{\rho}\}$. The advantage of such action definition is that it limits the action space to a subset of discrete positive values with low cardinality, and it facilitates simultaneous selection of several resources with a single action.

Reward. For a given service, KPI satisfaction is achieved when the allocated resources make the observed KPIs to meet their respective target values. However, beside meeting the target KPIs, it is essential to keep the observed KPIs as close as possible to the target KPIs; failing that, the system may perform better than required at the cost of extra resource consumption. Consequently, the choice of a reward function should be such that it equally accounts for all the KPIs for a given service and its value increases as the observed KPIs approach the corresponding thresholds and vice versa.

Let the observed values of the livecast KPIs, i.e., WVMAF and client buffer state, and of the vRAN KPIs, i.e., latency and packet loss rate for the m-th UE, be denoted by $\zeta_o^m, \sigma_o^m, \lambda_o^m, \mu_o^m$, while the corresponding target values be $\zeta_t, \sigma_t, \lambda_t, \mu_t$, respectively. We define the reward value for m-th UE, r^m, as the sum of the reward components pertaining to each service-specific KPI k in the n-th monitoring slot within the same decision window, as:

$$r^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) = \begin{cases} r_\zeta^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) + r_\sigma^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}), & \text{if } \ k = 1\,, \\ r_\lambda^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) + r_\mu^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}), & \text{if } k = 2\,. \end{cases} \tag{5.2}$$

In the above expressions, $r_\zeta^m(\cdot), r_\sigma^m(\cdot)$ are the reward components from WVMAF and buffer state (resp.) for the m-th UE, given by:

$$r_{\text{KPI}}^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) = \begin{cases} 1 - \text{erf}(\text{KPI}_o^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) - \text{KPI}_t), \\ \qquad\qquad\qquad\qquad \text{if KPI is met} \\ \text{erf}(\text{KPI}_o^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) - \text{KPI}_t), \qquad \text{else}\,. \end{cases} \tag{5.3}$$

The terms $r_{\lambda}^{m}(\cdot)$ and $r_{\mu}^{m}(\cdot)$ are instead the reward components from latency and packet loss rate (resp.), which are given by similar expressions but with $(\mathrm{KPI}_t - \mathrm{KPI}_o^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}))$ as an argument of the erf function, since all values of latency and packet loss rate lower than their respective target values are acceptable. Since the minimum and maximum values of the erf function lie between -1 and $+1$, we have: $-2 \leq r^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) \leq 2$. For the individual reward components, in the positive region of operation, i.e., when the KPI threshold is met, the reward value is positive and it further increases to its maximum value $+1$ as the observed KPI approaches its target KPI value. Likewise, in the negative region of operation, i.e., when the KPI threshold is not met, the value of the individual reward components is negative, which further reduces and saturates to the minimum value -1 as the observed KPI moves away from the KPI threshold.

We recall that while devising the greedy resource allocation policy, the goal of the RL agent is to maximize the cumulative reward measured as the sum of immediate reward and future rewards over a long time horizon. To this end, we consider a generic decision window h and, extending the previous notation, we let $a_k^{(h-1)}$ denote the action for the k-th service selected in decision window $(h-1)$ and applied in decision window h. We then define the average reward over h, considering all the UEs, as

$$\overline{r}(\boldsymbol{y}^{(h)}, \boldsymbol{a}_k^{(h-1)}) := \frac{1}{MN} \sum_{n=1}^{N} \sum_{m=1}^{M} r^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(h-1)}), \tag{5.4}$$

where $\boldsymbol{y}^{(h)}$ is the vector of shared contexts observed in the N monitoring slots in decision window h, while $\boldsymbol{a}_k^{(h-1)}$ is the action for service k selected in decision window $h-1$ and applied in decision window h. Finally, we adopt the definition of cumulative reward for the k-th service, observed during decision window h, as the differential return $\boldsymbol{G}_k^{(h)}$ defined in [145] (see Appendix A in the Supplemental Material).

Estimation of user buffer states. A key challenge, however, is to estimate the actual buffer dynamics *without* explicit feedback from the users. Thus it is important to design an effective *online learning* mechanism. To this end, we keep track of the time status information provided by the video encoder and the sequence number of TCP acknowledgments, all information locally available. By monitoring the amount of bytes successfully delivered, the corresponding timestamp of the scenes been transmitted, and the encoded video's frame rate, we can estimate the buffer dynamics at the client's side using simple queuing theory. Fig. 5.8 shows the evolution over time of a video player's buffer state (ground truth) vs. the inferred value for 3 trivially chosen videos and system configuration parameters. In most of the cases, our inference method is remarkably

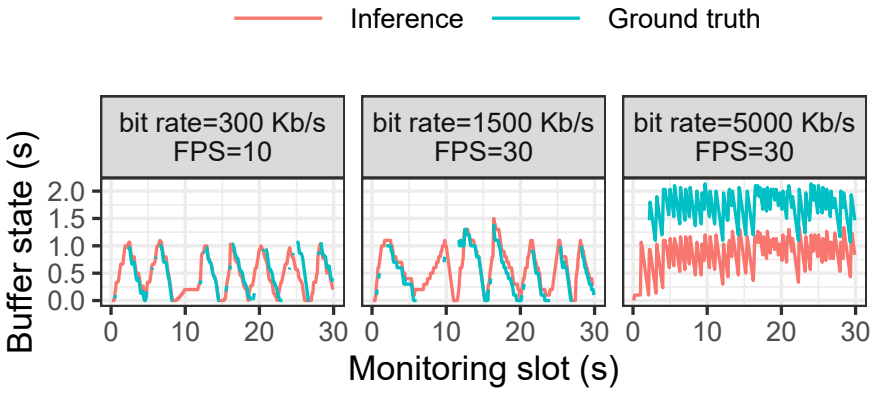


Fig. 5.8 Livecast client's buffer state inference

accurate. This is the case for the first two subplots in Fig. 5.8. However, we have found that there are some small number of cases where there is a non-negligible inference error. We show an example of this in the right-most plot of the figure, where we have an error of almost one second. Fortunately, these always occurs for non-critical cases, i.e., cases where the buffer state never approaches zero (the state we want to avoid). Since our estimator is pessimistic, the outcome are simply more conservative decisions. We hence conclude that our inference approach is valid to compute reward.

Action-value estimation and action selection. At the end of the generic decision window h, actions need to be evaluated and the best one has to be selected. To this end, we compute the mean shared context over the N monitoring slots in h as

$$\overline{\boldsymbol{y}}^{(h)} = \sum_{n=1}^{N} z_n \boldsymbol{y}^{(n)} / \sum_{n=1}^{N} z_n, \tag{5.5}$$

where $z_n > 0$ and $z_N > z_{N-1} > \cdots > z_1$ are the weights assigned so that the latest shared context has the highest weight.[3] We then quantify the goodness of taking an action in response to the mean shared context using action values. For service k, the value of $\boldsymbol{a}_k^{(h)}$ given policy π_k, which is $q_{\pi_k}(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)})$ (see Appendix A in the Supplemental Material), is defined as the expected differential return conditioned on $\overline{\boldsymbol{y}}^{(h)}$ and $\boldsymbol{a}_k^{(h)}$, following policy π_k, i.e.,

$$q_{\pi_k}(\boldsymbol{y}, a) = \mathbb{E}_{\pi_k}[\boldsymbol{G}_k^{(h)} | \overline{\boldsymbol{y}}^{(h)} = \boldsymbol{y}, \boldsymbol{a}_k^{(h)} = a]. \tag{5.6}$$

[3] Although they can be arbitrarily set, we fix them to $1, \ldots, N$, in accordance with the temporal sequence of the monitoring slots.

Since the context space $\mathcal{X}$ falls in the domain of real numbers, we use a practical method for action-value estimation using function approximation in an F-dimensional space, yielding the approximated function $\hat{q}_{\pi_k}(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)}, w) = \sum_{f=1}^{F} w_f s_f(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)})$, where $\mathbf{w}$ and $\mathbf{s}(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)})$ denote the F-size weight and feature vectors (resp.), with the latter being generated using tile coding [146] (see Appendix B in the Supplemental Material).

The estimation of the action values is followed by an ϵ-greedy action selection policy [145], which selects the best action for each service so as to maximize its cumulative reward over an infinite time horizon. We consider an ϵ-greedy action selection with $\epsilon = 0.5$ and ϵ-decay factor $= 0.999$. The ϵ parameter decays by a factor of 0.999 in the subsequent decision period. This favors higher exploration while the environment is still unfamiliar; with progression of time, instead, it allows for further exploitation of the environment knowledge gained during the exploration, so as to maximize the expected return.

Discussion. While single-agent RL approaches can easily solve the above capacity constraints, they suffer from the curse of dimensionality, even if implemented with deep neural networks (see DQNs) [168]. As a result, the key challenge is to provide safe (i.e., within a set of hard constraints) and fair resource allocation decisions with a distributed multi-agent RL model that is amenable to scalable orchestration.

In this way, our distributed approach allows us to handle two sets of capacity constraints:

1. Computing resources: VERA can handle multiple vBSs (or multiple radio slices within a vBS), and multiple edge services that are competing for the same computing resource budget;

2. Radio resources: VERA can handle multiple users sharing a common carrier bandwidth.

Although for the sake of clarity the above text and Fig. 5.7 refer to two service learning agents only, one of which is a single vBS serving M UEs, the scalable multi-agent design of VERA allows for as many learning agents as services, vBSs, or slices under the above capacity constraints.

Algorithm 5 Fair Pareto-efficient Resource allocation

$S = \{\tilde{a}_k\}_k$, $\{c_k, \rho_m(k)\ \forall m \in \mathcal{M}\} \leftarrow \tilde{a}_k(S), S' = \{c_k, \rho_m(k)\ \forall m \in \mathcal{M}\}, \forall k \in \mathcal{C}$ ▷ Extract capacity-constrained actions from greedy action set $\{\tilde{a}_k\}$
if $\sum_{k\in\mathcal{C}} c_k \leq B_c$ and $\sum_{m\in\mathcal{M}} \rho_m(k) \leq B_\rho, \forall k \in \mathcal{C}$ **then** ▷ Capacity-constraint check on the primary and secondary resource
    $S^\star = S'$ ▷ Output: Fair Pareto-efficient solution
else if $\sum_{k\in\mathcal{C}} c_k \leq B_c$ and $\sum_{m\in\mathcal{M}} \rho_m(k) > B_\rho$, for any $k \in \mathcal{C}$ **then** ▷ Primary resource budget constraint met, secondary resource budget constraint not met
    $\{\rho'_m(k)\ \forall m \in \mathcal{M}\} \leftarrow$ ParetoBlock($\{\rho_m(k)\ \forall m \in \mathcal{M}\}$), for the considered $k \in \mathcal{C}$ ▷ Revised Pareto-efficient fair secondary resource allocation adhered to budget constraint and allocated CPU
    $S'' = \{c_k, \rho'_k(m)\ \forall m \in \mathcal{M}\}, \forall k \in \mathcal{C}$
    $S^\star = S''$ ▷ Output: Fair Pareto-efficient solution
else if $\sum_{k\in\mathcal{C}} c_k > B_c$ **then** ▷ Primary resource budget constraint not met
    $\{c'_k\} \leftarrow$ ParetoBlock($\{c_k\}$), $\forall k \in \mathcal{C}$ ▷ Revised Pareto-efficient fair primary resource allocation adhered to its budget constraint
    $\{\rho'_m(k)\ \forall m \in \mathcal{M}\} \leftarrow$ ParetoBlock($\{\rho_m(k)\ \forall m \in \mathcal{M}\}$), $\forall k \in \mathcal{C}$ ▷ Revised Pareto-efficient fair secondary resource allocation adhering to revised Pareto efficient fair primary resource allocation
    $S'' = \{c'_k, \rho'_m(k)\ \forall m \in \mathcal{M}\}, \forall k \in \mathcal{C} S^\star = S''$ ▷ Output: Fair Pareto-efficient solution

Algorithm 6 ParetoBlock

Input: $\{c_k\}\ \forall k \in \mathcal{C}$ s.t. $\sum_{k\in\mathcal{C}} c_k > B_c$ or $\{\rho_m(k)\ \forall m \in \mathcal{M}\}$ s.t. $\sum_{m\in\mathcal{M}} \rho_m(k) > B_\rho\}$ ▷ Primary or secondary resource allocation violating the budget constraints
$S_1 = \{c_k\}\ \forall k \in \mathcal{C}$ or $S_1 = \{\rho_m(k)\}\ \forall m \in \mathcal{M}$, as applicable $\mathcal{S}_e = \{S_1, S_2, \ldots\}$ ▷ Build expanded solution set
$\mathcal{S}_s \leftarrow \{S_i/|\mathcal{C}|\}_{\mathcal{S}_e}$ or $\mathcal{S}_s \leftarrow \{S_i/|\mathcal{M}|\}_{\mathcal{S}_e}$, as applicable ▷ Rescale expanded solution set
for $S \in \mathcal{S}_s$ **do** $\widehat{\mathcal{S}}_s \leftarrow \{\widehat{a}_k(S)\}$ ▷ Define refined actions set wrt $\mathcal{S}_s$
Create $\mathcal{S}_d$ ▷ Pareto dominant solution set
Choose S'_1 ▷ Fair Pareto-efficient resource allocation
return $S'_1 = \{c'_k\}\ \forall k \in \mathcal{C}$ or $S'_1 = \{\rho'_m(k)\}\ \forall m \in \mathcal{M}$, as applicable

5.4.3 Pareto analysis

We recall that the CPU and RB allocation for (resp.) service k and UE m are capacity-constrained resources. Hence, it is essential that the sum of CPU (RB) allocated to different services (UEs) does not exceed the available resource budget and that the selected actions can be enacted. To this end, we introduce an algorithm that works on the multi-dimensional actions selected by the ϵ-greedy policy in the RL framework introduced above, and it further refines them so that the resulting actions (i) meet the budget constraint and (ii) entail fair Pareto-efficient resource sharing.

It is important to note here that not only the CPU allocation across the services and RB allocation across the UEs are capacity constrained, the RB allocation is also dependent on the CPU allocated to vRAN. Thus, CPU and RB (resp.) act as the primary and secondary capacity constrained resources for vRAN. Unlike vRAN, the livecast service has no associated secondary capacity-constrained resource. However, for the sake of mathematical proofs, this observation can be generalized as follows: the primary capacity constrained resource (here CPU) is distributed among K services, and each service may in turn serve M units (UEs for vRAN, none for livecast) using the primary resource. The secondary capacity constrained resource is distributed among

M units of the service (if applicable). Further, the QoS satisfaction of each service in entirety comprises QoS satisfaction of individual units, and depends both on primary and secondary capacity constrained resource allocation.

To model such interdependence, we formulate the fair Pareto-efficient allocation of CPU across the services and RBs across the UEs in a given decision window as a constrained joint multi-criteria optimization problem. Further, for notational simplicity, we assume that each decision window comprises of just one monitoring slot, i.e., $N = 1$. Let $\mathcal{C}, \mathcal{M}$ denote the set of services and UEs; given a set of coefficients $u_k \geq 0, k \in \mathcal{C}, m \in \mathcal{M}$, with $\sum_{k\in\mathcal{C}} u_k = 1$, it is required to find a solution $S^\star = \{c_k^\star, \rho_m^\star(k) \ \forall m \in \mathcal{M}\}, \forall k \in \mathcal{C}$, that maximizes $\sum_{k\in\mathcal{C}} u_k \mathbf{\Gamma}_k(S)$ such that $S \in \mathcal{S}_c$, $\sum_{k\in\mathcal{C}} c_k \leq B_c$ and $\sum_{m\in\mathcal{M}} \rho_m(k) \leq B_\rho$. Here, c_k is the primary capacity constrained resource allocated to k-th service, $\rho_m(k)$ denotes the secondary capacity constrained resource allocated to m-th unit of the k-th service, $\mathcal{S}_c$ is the set of feasible capacity constrained resource allocations and $\mathbf{\Gamma}_k(S)$ is the criteria function denoting the reward of the k-th service in a decision period following the CPU allocation strategy S.

The flow of fair Pareto-efficient resource allocation is summarized in Alg. 5. The key component of Alg. 5 is ParetoBlock (Alg. 6) that solves the joint multi-criteria optimization problem. It is invoked whenever the sum of allocated primary (secondary) capacity constrained resource exceeds its specified budget. It initially considers the CPU (RB) allocation to the services (UEs) provided by the greedy resource allocation policy, and creates the expanded CPU (RB) allocation solution set by considering all possible values for the c_k's ($\rho_m(k)$'s) that are greater than those output by the greedy policy and whose sum does not exceed $|\mathcal{C}|$ ($|\mathcal{M}|$) times the available budget. Such values are then scaled by $|\mathcal{C}|$ ($|\mathcal{M}|$), to get candidate allocation values that meet the CPU (RB) budget. The corresponding action set, $\widehat{\mathcal{S}}_s$, is built starting from such c_k's ($\rho_m(k)$'s) and possibly refining the actions so that their components take feasible values considering the dependence of primary and secondary resource. Such actions, $\{\widehat{\boldsymbol{a}}_k(S)\}$, $\mathcal{S} \in \widehat{\mathcal{S}}_s$, are then used to compute the values of $\mathbf{\Gamma}_k(S)$ to identify the Pareto-dominant solution set through iterative search and update, $\mathcal{S}_d \leftarrow \{S\}$, s.t. $\forall S \in \mathcal{S}_d$, $\forall S' \in \widehat{\mathcal{S}}_s, \mathbf{\Gamma}_i(S) > \mathbf{\Gamma}_i(S'), \mathbf{\Gamma}_j(S) \geq \mathbf{\Gamma}_j(S'), \forall i, j \in \mathcal{C}(\mathcal{M}), i \neq j$. Finally, for the primary capacity constrained resource, the Pareto-dominant solution that maximizes the minimum value of criterion function, i.e., the reward value over all the services is chosen as the fair Pareto-efficient solution. For the secondary capacity constrained resource, we define $g(m) := l_t(m) - l_i(m) \ \forall m \in \mathcal{M}$, where $l_t(m), l_i(m)$ denote (resp.) the target traffic load and the instantaneous rate achieved by m-th UE. Further, we choose the secondary

resource allocation $\{\rho'_m(k)\ \forall m \in \mathcal{M}\}$ for a given fair primary resource c'_k as the solution that minimizes $\max_{m\in\mathcal{M}} g(m)$ over all $S \in \mathcal{S}_d$.

We finally prove the following results:

- The solution $S^\star$ introduced above is Pareto-efficient with respect to the primary (see Proposition 1), as well as jointly Pareto-efficient with respect to primary as well as secondary capacity constrained resources (see Proposition 2);
- Alg. 6 converges to a Pareto-efficient solution set, at a sub-linear rate (see Proposition 3);
- Alg. 6 converges to a solution that is fair with respect to the primary capacity constrained resource (see Proposition 4), as well as the secondary capacity constrained resource for a given primary capacity constrained resource allocation (see Proposition 5), thus leading to a fair Pareto-efficient solution.

Proposition 1. *Pareto-efficient allocation of the primary resource*: Given a set of coefficients $u_k \geq 0, k \in \mathcal{C}$, such that, $\sum_{k\in\mathcal{C}} u_k = 1$, then the solution $S^\star = \{c_k^\star\}$, $k \in \mathcal{C}$, that maximizes the multi-criteria optimization problem $\sum_{k\in\mathcal{C}} u_k \mathbf{\Gamma}_k(S)$, is Pareto-efficient.

Proof. See Appendix C in the Supplemental Material. □

Proposition 2. *Pareto-efficient joint allocation of primary and secondary resource*: Given a set of coefficients $u_k \geq 0, k \in \mathcal{C}$, such that, $\sum_{k\in\mathcal{C}} u_k = 1$, then the solution $S^\star = \{c_k^\star, \rho_m^\star(k)\ \forall m \in \mathcal{M}\}, \forall k \in \mathcal{C}$, that maximizes the multi-criteria optimization problem $\sum_{k\in\mathcal{C}} u_k \mathbf{\Gamma}_k(S)$, is Pareto-efficient.

Proof. See Appendix C. □

Proposition 3. Alg. 6 converges to a Pareto-efficient solution set at a sub-linear rate.

Proof. See Appendix C in the Supplemental Material. □

Proposition 4. *Fairness of Pareto-efficient primary resource allocation*: The solution $S^\star = \{c_k^\star, \rho_m^\star(k)\ \forall m \in \mathcal{M}\}, \forall k \in \mathcal{C}$ obtained using Algorithm 2 is fair with respect to primary resource allocation $c_k^\star, \forall k \in \mathcal{C}$.

Proof. See Appendix C in the Supplemental Material. □

Proposition 5. *Fairness of Pareto-efficient secondary resource allocation in the vRAN*: For a given fair primary resource allocation $c_k^\star$ in the solution $S^\star = \{c_k^\star, \rho_m^\star(k) \ \forall m \in \mathcal{M}\}$, for $k = 2$ (denoting the vRAN service), $S^\star$ is fair with respect to secondary resource allocation $\{\rho_k^\star(m) \ \forall m \in \mathcal{M}\}$.

Proof. See Appendix C in the Supplemental Material. □

5.4.4 Learning algorithm

We exploit the concept of experience-based learning using sample sequences of shared context, actions, and rewards observed from the actual interaction of the RL agent with the environment. SARSA, an acronym for quintuple $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$, is an on-line policy algorithm where learning of the RL agent at time t is governed by its current state S_t, choice of action A_t, reward R_t received on taking action A_t, state S_{t+1} that the RL agent enters after taking action A_t, and finally the next action A_{t+1} that the agent chooses in new state S_{t+1} [145].

For clarity and without loss of generality, here we focus on the learning of a single RL agent that corresponds to one of the services, over successive decision windows. Given the mean shared context and possible actions, the primary steps in the learning algorithm are: (i) obtain greedy resource allocation policy for service k through estimation of action values $q_{\pi_k}(y, a)$, (ii) obtain a fair Pareto-efficient resource allocation policy by collating and returning the greedy policies of all the services, and (iii) update of the action-value estimates for service k using differential semi-gradient SARSA [145].

5.5 Proof-of-concept Implementation

In this section, we first introduce our proof-of-concept implementation (Sec. 5.5.1), and then we present the parameters and settings we use to collect our datasets and run our experimental tests (Sec. 5.5.2).

5.5.1 VERA implementation

As depicted in Fig. 5.9, we have integrated VERA in a testbed based on `srsRAN` (to emulate a vRAN service), `ffserver` (to emulate a livecast video service), and `mpv` (livecast video client deployed at each UE video player).

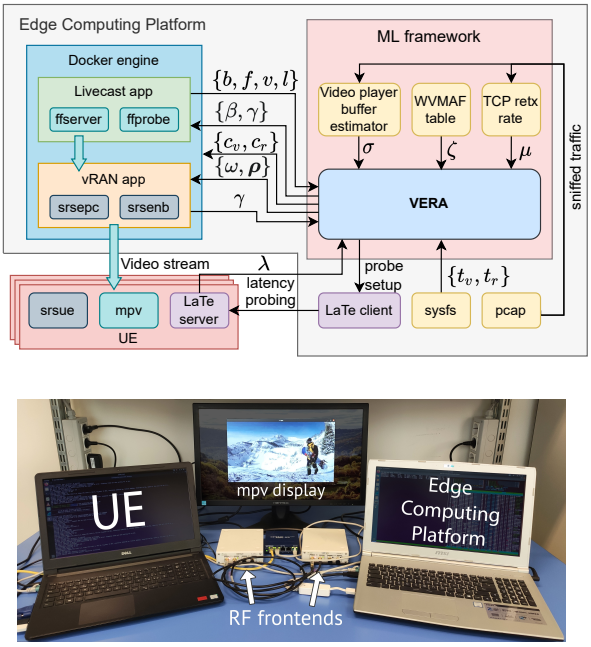


Fig. 5.9 (top) VERA system representation (details in Sec. 5.5.1); (bottom) picture of the testbed (details in Sec. 5.5.2)

VERA's learning agents receive real-time context information directly from the vRAN, the livecast service, and the edge computing platform, using custom-made TCP-based interfaces. More specifically, the CQI information is retrieved from srseNB using its `enb_metrics_interface` class. The input video features (i.e., input video FPS, bitrate, and resolution) are instead probed using `ffprobe`, which is part of `ffmpeg`, and then sent to VERA whenever a new video source is selected. The CPU throttling times of the livecast and vRAN services are collected by interfacing with Linux `cgroups`, using Linux' pseudo file-system `sysfs`. Finally, the network load, which corresponds to the offered load that the vBS has to process, is derived by summing up the bytes produced each second by the video encoder, as reported in its output log.

Concerning the reward signal, the observed unidirectional latency, the packet loss rate, the estimated video player buffer state, and the WVMAF are sent to VERA in real time, using additional custom TCP-based interfaces. In more detail, the latency measurements are obtained using LaTe[4], a flexible client-server multi-protocol Latency Tester that sends probes to the network under test and measures the delay that the probe experiences. To this end, clock synchronization between the Edge Platform and the UEs is performed using Precision Time Protocol daemon (PTPd).

[4] https://github.com/francescoraves483/LaMP_LaTe

Table 5.2 Input and output video characteristics

Video characteristics	**Input**	**Output**
Resolution [pixels]	1920 × 1080	1280 × 720
Bit rate [Mbps]	18	0.3, 1.5, 5, 10
Frame rate [FPS]	30	10, 20, 30
Codec (container)	VP9 – WebM	VP9 – WebM

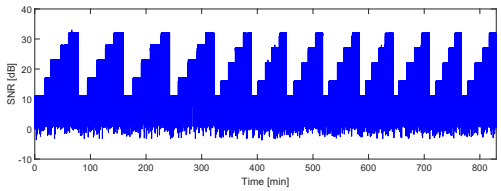


Fig. 5.10 Time evolution of SNR in our experiments

The packet loss rate, computed through the TCP segment retransmission rate, and the buffer occupancy are estimated at the edge platform by leveraging, respectively, the TCP sequence and the acknowledgment numbers obtained using `libpcap`. To model the player buffer and, hence, infer its status, VERA exploits the TCP acknowledgment numbers, the offered load, and the output frame numbers of the video encoder, as explained in Sec. 5.4. Finally, to compute the WVMAF, we use a lookup table that maps every choice of encoding parameters to the expected VMAF score. Calculating the VMAF score is indeed a computing-intensive task that cannot be performed in real-time without a noticeable performance impact. The table has been built by considering a collection of 1080p video samples[5], and by encoding each source video using every encoding parameters combination available to VERA. The VMAF score is calculated by comparing the frames of each encoded video to the original frames. The WVMAF score of each video sample is obtained by multiplying its VMAF score by the ratio of the output frame rate and the input frame rate. Then, for every combination of the encoding parameters, the corresponding WVMAF score is computed averaging the WVMAF scores of all video samples encoded with the same combination of parameters.

At last, to enforce decisions made by VERA, the per-UE RB allocation and the MCS policy are sent to the vRAN through a custom interface, the CPU allocation is set through Docker API (which, in turn, enforces it using Linux cgroups), and the video encoding FPS and bit rate policies are updated overriding the parameter settings in the livecast service, thus allowing VERA to work in real time.

[5] https://media.xiph.org/video/derf/

Table 5.3 Video encoder, server and client parameters

Encoder/server param.	Description	Value
StartSendOnKey	If set, the video is streamed starting from the first I-frame generated by the encoder, i.e., P-frames not preceded by an I-frame are discarded	Enabled
Preroll N	The video is streamed starting not from the most recent frame but from N seconds in the past; if set, it increases buffer occupancy at the expense of the end-to-end latency	Disabled
VP9 Threads	No. of threads that decoder & encoder can use: high values increase speed if multiple vCPUs are allocated, at the cost of a small overhead.	No. of allocated vCPUs
VP9 Quality	Possible settings: realtime, good, or best. It controls the time that the encoder can take to encode frames beyond their presentation time	Realtime (no additional time beyond presentation timestamp)
VP9 Speed	It controls the trade-off between computational lightness and picture quality. Possible values in [0,16], with the higher values prioritizing encoding speed (i.e., lower CPU consumption) over picture quality	16
Client CachePauseInitial	The client pauses the playback at the beginning to wait for the buffer to fill, so as to avoid pauses while the video is playing	Enabled
Client CachePauseWait	Video time that the client requires before resuming the playback when paused. It affects the end-to-end latency, but a bigger size can better cope with oscillations in the data transfer and encoding delays	1 s

5.5.2 Testbed configuration

The testbed configuration used to collect the necessary dataset and run the VERA framework in real time is based on the architecture presented in Sec. 5.2.

The edge computing platform and the UEs are hosted on GNU/Linux machines; they accommodate, respectively, an Intel i7-7700HQ and an Intel i7-8550U CPU, with 16 GB of DDR4 memory. The LTE network uses a 10-MHz channel in band 7, which provides a capacity of 50 RBs. The dynamic SNR pattern, considered in our experiments to be experienced by the UEs, is depicted in Fig. 5.10; the values of SNR are then mapped into CQI by the vRAN system. We assume a network slice dedicated to the livecast service, with a capacity that may vary between 12 and 36 RBs. As RF frontend, Ettus USRP B210 boards are used to perform up/down-conversion, filtering, amplification, and AD/DA conversion of the UEs and eNB LTE signals.

As mentioned above, we use `ffmpeg`, as this is compatible with a wide set of video codecs, picture formats, containers, besides offering a number of filters to modify video characteristics. Tab. 5.2 reports the characteristics of the input and output videos in our experiments. The characteristics of the input videos are representative of a livecast content, as they ensure that the video can be properly played by the client player in a wide range of network conditions and client configurations. The output parameters have been set so as to allow for the best video quality retention, hence a good level of user Quality of Experience, while requiring a reasonable consumption of network and computing resources. Tab. 5.3 includes additional parameters settings that we have used at the video encoder, server, and client, which are typical of a livecast service.

Finally, we consider that decisions are made every monitoring slot ($N = 1$), and, unless otherwise specified, we set the available CPU budget to 3 vCPUs.

5.6 Evaluation and experimental validation

In this section, we first present the numerical results (Sec. 5.6.1) derived using the data sets obtained through extensive experiments on the testbed described in Sec. 5.5; and then, we present the performance of a real-time implementation of VERA on the testbed (Sec. 5.6.2).

5.6.1 Numerical results

The baseline scenario we consider in our numerical performance evaluation includes 1 vBS, 2 UEs, and a livecast service streaming a single video to both UEs. The CPU and RB budgets are fixed to 2 vCPUs and 60 RBs, respectively.

Convergence evaluation. Fig. 5.11 depicts the time evolution of reward values for the vRAN and livecast services in the baseline scenario. From the plots, we observe that, despite the large heterogeneous action set and the diverse context vector, the reward corresponding to each of the KPIs, and hence the total reward, saturates close to the maximum value for both the UEs, thereby highlighting the efficient learning capability of the VERA framework. Also, the convergence of the livecast service is relatively slower with respect to vRAN owing to its slowly varying dynamics.

KPI performance. Next, Fig. 5.12 presents the evolution of the KPIs across iterations during the learning process. Notice that the KPI satisfaction for vRAN is

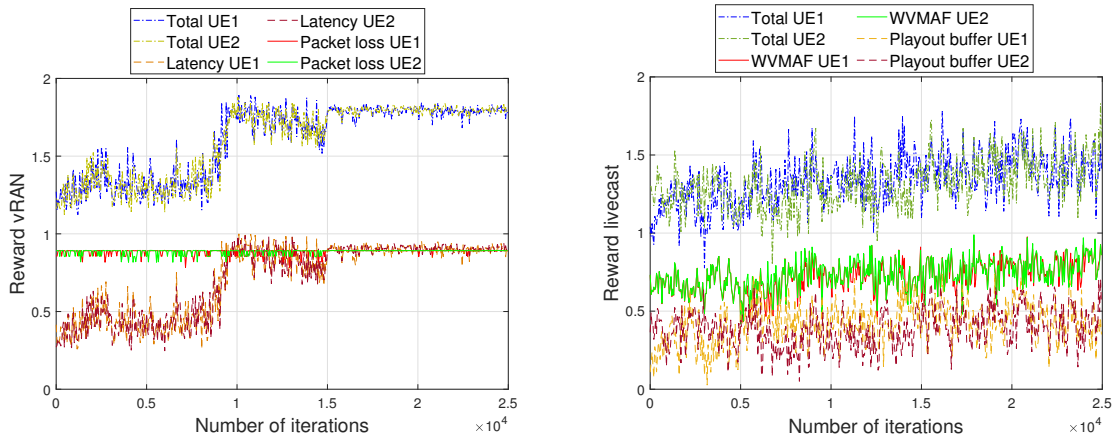


Fig. 5.11 Convergence of reward values: vRAN (a) and livecast (b) services

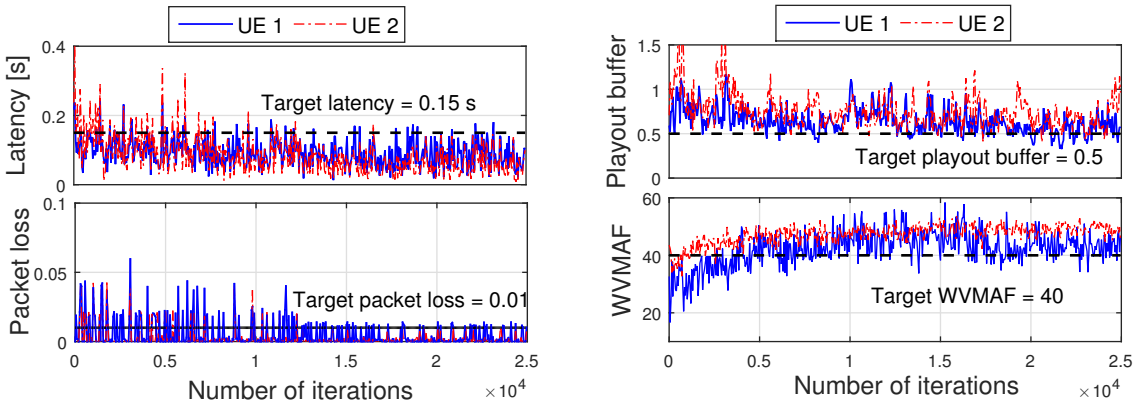


Fig. 5.12 KPI evolution with respect to iterations: (a) vRAN and (b) livecast. Dashed dark line shows target KPI values.

achieved when its latency and packet loss do not exceed their respective targets. On the contrary, for the livecast service, the playout buffer and WVMAF should not fall below their target values, while keeping the KPIs observed for both the services as close as possible to their target values. According to the 3GPP 5G specifications and acceptable QoE, the target KPI values are set at 150 ms, 0.01, 0.5 s and 40 (resp.) for latency, packet loss, playout buffer, and WVMAF. From the plots, we observe that barring a few initial iterations during which the algorithm is still learning, the choice of actions by the VERA framework leads to KPI satisfaction for both vRAN and livecast services. To quantify VERA's suboptimality, the mean KPI target violation for VERA post convergence of the algorithm is 3.7%.

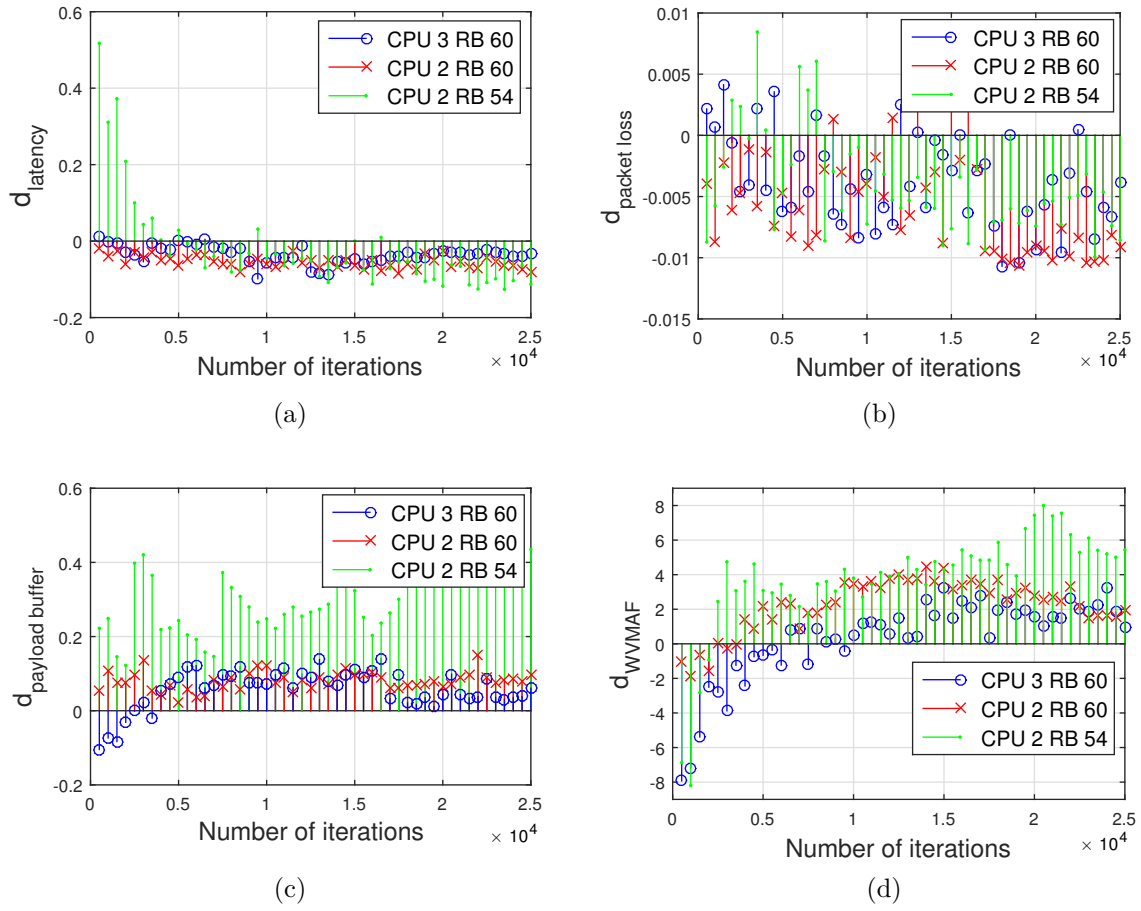


Fig. 5.13 Performance of VERA under varying CPU and RB budget constraints: (a) latency, (b) packet loss, (c) playout buffer, and (d) WVMAF

Performance under different constraints. We now evaluate the impact that different CPU and RB capacity constraints have on the performance of VERA. To this end, we consider two additional scenarios having budgets 3 vCPUs - 60 RBs and 2 vCPUs - 54 RBs, along with the baseline scenario. The performance is characterized using a distance parameter $d_{KPI} = \text{KPI}_o - \text{KPI}_t$, where KPI_o is the KPI target threshold and KPI_t is the KPI experienced at time t, and which basically quantifies how far away the observed KPI value is from its target. Fig. 5.13 compares the average value of d_{KPI} computed over both UEs for the said scenarios, for both vRAN and livecast services. It may be noted that a negative (positive) value of d_{KPI} for vRAN (livecast) denotes KPI satisfaction, and, irrespective of the service type, it is desirable that d_{KPI} is as close as possible to 0. From the plots, we observe that when the budget constraints are stringent, the choice of actions in order to meet the KPI target values is limited. Consequently, the resource allocation efficiency is slightly compromised as shown by higher d_{KPI} values.

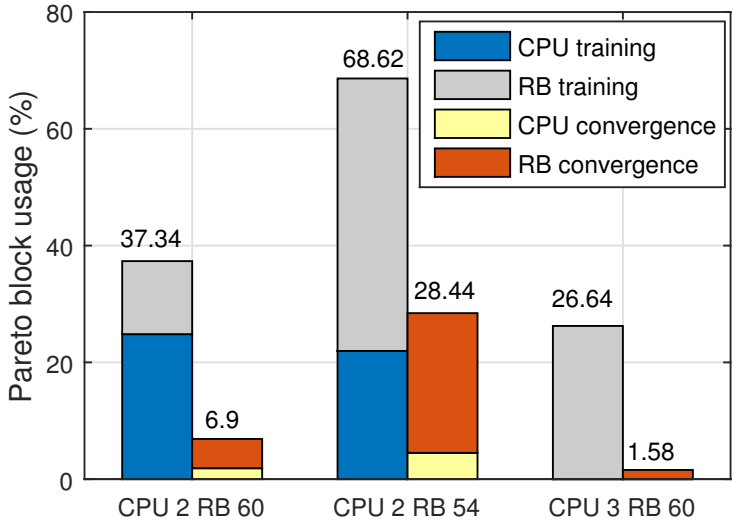
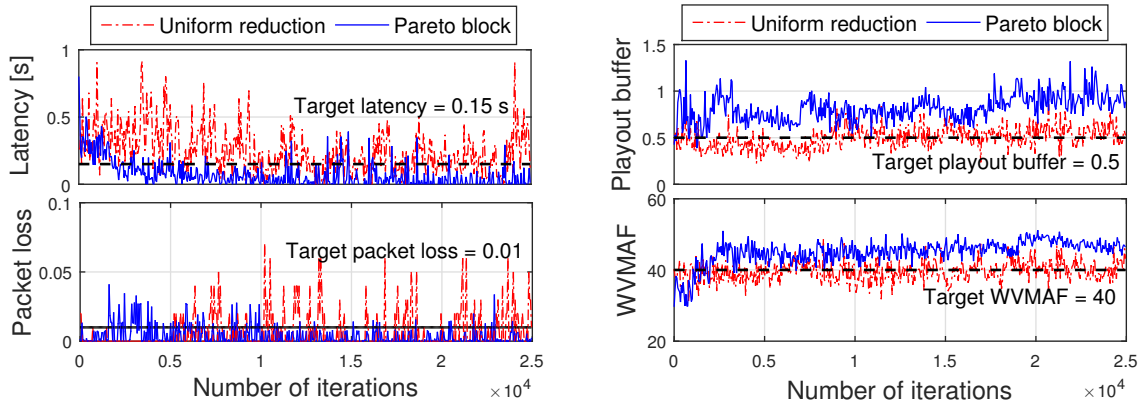


Fig. 5.14 Utilization of Pareto block in VERA during training and after convergence has been attained



(a) (b)

Fig. 5.15 KPI comparison of VERA with Pareto block and uniform reduction approach: (a) vRAN, (b) livecast

Nevertheless, the KPI satisfaction is still achieved. As more resources are made available in terms of CPU and RBs, d_{KPI} values cling closer to 0, thereby minimizing the wastage in resource allocation. Thus, VERA can successfully attain KPI satisfaction for both the services and UEs under varying CPU and RB capacity constraints, however, stringent constraints may lead to a marginal loss in efficiency.

Pareto block statistics. Next, we investigate the significance of the Pareto block in the VERA framework. The bar plot in Fig. 5.14 shows the statistics of the Pareto block usage under different CPU and RB capacity constraints. We observe that the Pareto

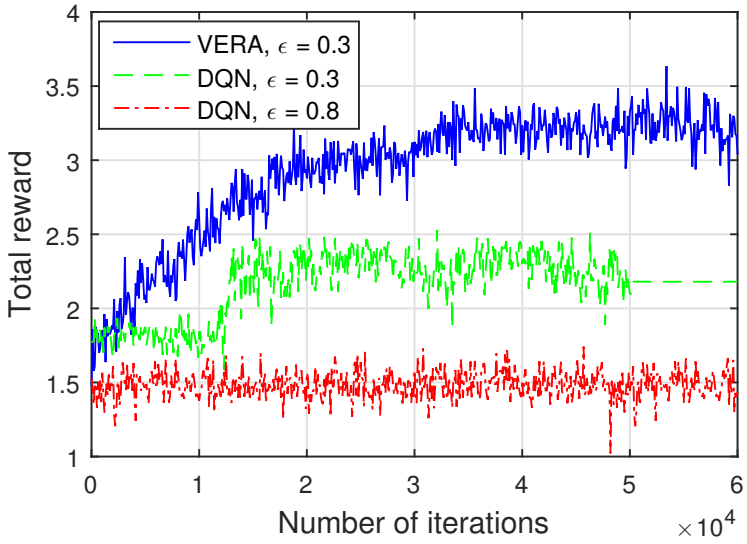


Fig. 5.16 Performance of VERA in comparison to DQN

block usage for CPU as well as RB is the highest when budget is the most stringent, i.e., 2 vCPUs - 54 RBs. However, on a positive note, the Pareto block invocation substantially reduces once VERA has attained convergence compared to its training phase. This in turn suggests that, when a pre-trained VERA model is used, the Pareto block will not add to the system runtime complexity.

To further emphasize this aspect, we replace the Pareto analysis in the VERA framework by a more intuitive and simpler uniform reduction approach, wherein an equal proportion of any excess CPU (RB) allocated beyond the budget is subtracted from the allocated CPU (RB) values across the services (UEs) such that the budget constraint is met. Fig. 5.15 presents the KPI (averaged over both UEs) comparison using the Pareto block and uniform reduction in the worst of our considered scenarios, i.e., 2 vCPUs - 54 RBs for vRAN and livecast services. From the plots we observe that unlike the Pareto block, uniform reduction fails to meet the target KPI values. This confirms that the Pareto block has a crucial role in optimal resource orchestration, especially when resources are constrained.

Comparison with other approaches. Finally, we address the scalability of the VERA framework. To emphasize the distributed decision making used by VERA, we compare its performance to a data-driven centralized framework using deep Q-network (DQN). Since a generic DQN has no provision to enforce capacity constraints, to be fair, we consider a scenario wherein full CPU is available to the hosted services and there is no RB capacity constraint. Fig. 5.16 shows the convergence of total reward from vRAN as well as livecast services observed in consequence to actions chosen by VERA and DQN.

We observe that, even if the same number of actions per service is considered both for VERA and DQN, being a centralized framework, DQN has to deal with a much larger action space that increases exponentially with the number of services. This is also evident from the poor convergence of DQN plots in Fig. 5.16. With the same ϵ-greedy action selection policy as VERA (i.e., $\epsilon = 0.3$, ϵ-decay $= 0.9999$), DQN is unable to explore all the actions while ϵ decays down to a negligibly small value and the learning agent gets caught up in a local optimum. Even if the action selection parameters are improved ($\epsilon = 0.8$, ϵ-decay $= 0.9999$), DQN still explores the action space until $\sim$ 60k iterations in our experiments, whereas VERA has shown a clean and early convergence with smaller ϵ value. To this end, it is worth noting that VERA exhibits a much higher scalability in terms of cardinality of action space compared to the state-of-the-art DQN-based centralized framework. To quantify the scalability performance, we define the scaling cost as the sum of reward deficit with respect to maximum reward value at convergence and the fraction of total iterations used for convergence. In our experiments, we found that the scaling cost of VERA is 45% and 60% lower compared to DQN, (resp.) for $\epsilon = 0.3$ and $\epsilon = 0.8$. In more complex scenarios, considering the CPU and RB budget constraints, or more services or more UEs in the system, further increases the cardinality of the action set, however, our observations on scalability of VERA with respect to DQN still hold.

5.6.2 Proof-of-concept results

A pre-trained version of the VERA RL agents has been generated offline using the dataset collected from our testbed, then it has been evaluated online and in real time in two scenarios with different RBs availability, i.e., 24 and 36 RBs, and with CPU budget equal to 3 vCPUs. In both scenarios, a single UE connects to the vRAN and receives the livecast, using the configuration described in Sec. 5.5.2. For this case, the USRPs' transmission gain is set to a high value, so as to ensure that the SNR on uplink and downlink does not drop below 29 dB.

The radio latency, playout buffer and WMAF KPIs, collected from both testbed and numerical experiments, are compared in Figure 5.17 (packet loss is omitted as it is equal to 0 in all cases). All KPIs are always satisfied for both the vRAN and the livecast service. As expected, the latency is higher in the case of 24 RBs than for 36 RBs, on the contrary, the playout buffer and WVMAF are less for 24 RBs than 36 RBs, owing to the lower number of radio resources being available in the first scenario: a higher number

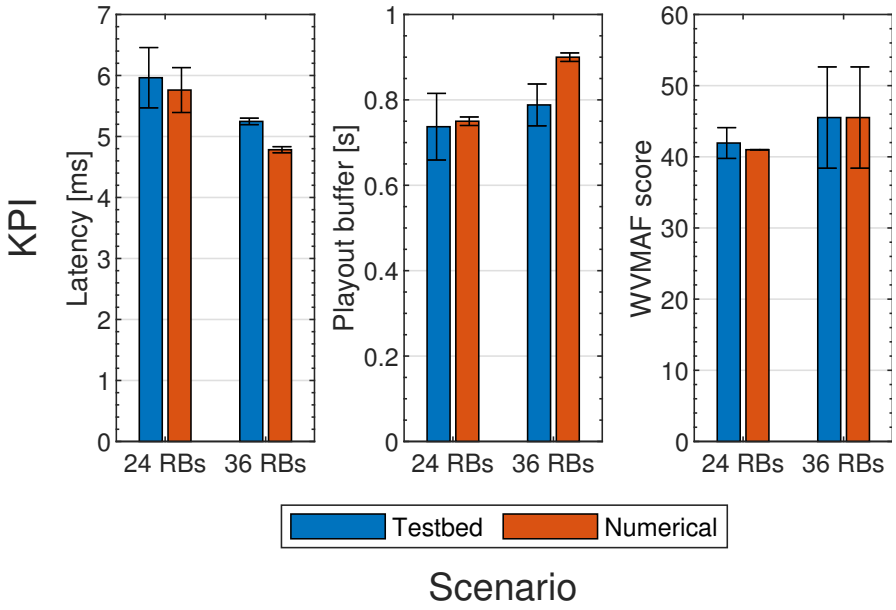


Fig. 5.17 Comparison of testbed and numerical KPI values for the livecast and vRAN services on the 24 RBs and 36 RBs scenarios with CPU budget equal to 3 vCPUs. Error bars indicate the confidence interval at 95% confidence level.

of allocated RBs leads to higher user throughput, which results in less latency, higher playout buffer and higher WVMAF score.

Importantly, the relative difference between testbed and numerical KPI values never exceeds 12.4%, with such a value being observed in the case of the playout buffer in the 36 RBs scenario. The similarity between testbed and numerical results validates VERA performance in a real time, hardware-in-the-loop implementation, and it demonstrates the effectiveness of our solution in a real-world environment.

5.7 Related Work

Several works have addressed the VNF placement problem at the network edge, which is related but orthogonal to the problem we face. Recent examples include: [169], which minimizes latency and system cost; [170], which optimizes both service placement and traffic routing under different resource constraints; and [171], which uses cooperation among edge nodes for service caching and workload scheduling.

Other studies have focused on QoE provisioning to mobile users through edge-assisted solutions. In particular, [172] presents an RL framework for crowdcasting services at the

edge meeting bit rate as well as streaming and channel switching latency requirements, while minimizing the overall computing and bandwidth cost. [173], instead, designs and implements an edge network orchestrator, and a server assignment and frame resolution selection algorithm for best latency-accuracy trade-off in mobile augmented reality.

Relevant to our work are also existing studies on resource consumption by edge user applications. In particular, [174] investigates the impact of real-time video analytics on computing and energy resources, while [175] focuses on image processing through CNNs and maximizes the learning accuracy given the limited resources at the edge.

The above literature does not consider the inherent resource contention between edge services and virtualized RANs. Related with this, [158] jointly optimizes the allocated resources to edge services and the placement of RAN functions, but uses an over-simplistic linear optimization model that cannot adapt to quick system dynamics. Like us, some other authors have used machine learning for practical resource allocation, radio parameter settings, and service KPI support in cellular networks. Among these, [119–123] aim at maximizing throughput through channel or link-rate selection, using multi-armed bandit techniques. Similarly, but leveraging the contextual information from the environment, [124] proposes an RL approach for rate selection and resource allocation, and [161] to maximize throughput subject to power consumption constraints. Finally, [160] extends the latter to accommodate service KPI constraints. However, [160] does not consider individual user dynamics nor fairness in resource allocation, as we do. RL-based schemes can also be found in [130, 131, 176], to minimize latency and packet drop rate in 5G systems. The work in [159], instead, tackles computing resource allocation in a virtualized radio access, and introduces a deep RL approach for resource management. Deep RL is also used to determine the suitable MCS and transmit power level in cognitive radio networks in [136] and [137], respectively, and to maximize the network sum-rate in [138]. More recently, [177] proposed a data-driven O-RAN-compliant framework that configures DUs/RUs according to a specified spectrum access policy.

We underline that, unlike previous work, we address the allocation of edge resources *constrained to a limited budget across different, competing, virtual services.* Through our testbed, we identify the non-trivial correlations existing among the actions related to the different services, making the VERA learning objectives very different from those of existing works. To derive a scalable solution that can accommodate multiple services and vBSs (or slices of vBSs), we resort to a multi-agent RL mode. And, inspired by [162] and other literature on autonomous driving, we accommodate such hard constraints as a non-learnable building block. Different from previous work, however, we design a

Pareto-efficient block for this task, which provides fair resource allocation across agents (vBSs and edge applications).

Finally, a preliminary version of our solution with only one user and considering only CPU as a constrained resource was presented in our conference paper [178].

5.8 Conclusions

We considered an edge computing platform hosting virtualized user applications and network services (namely, vRAN) competing for the same resources. We first investigated the correlations existing between the dynamics of such services through an experimental testbed that leverages a containerized livecast application and a containerized LTE base station. Then we developed a distributed learning framework, called VERA, that sets the configuration of both types of services so that the target KPIs can be met in spite of the limited availability of computing resources at the edge. Importantly, VERA also exploits a Pareto analysis that leads to fair Pareto-efficient decisions, and it can scale well with the number of virtualized services that are hosted at the edge platform. Our experimental results demonstrate the feasibility of the VERA approach and the important role of the Pareto analysis. Also, they show the excellent performance we can obtain in the presence of capacity-constrained resources with the KPI target violation limited to just 3.7%. Further, we show that VERA performs similarly when executed in our real-time proof-of-concept implementation, with KPI differences below 12.4%, thus confirming the effectiveness of VERA also in a real-world environment. Finally, we compare the performance of VERA to the centralized DQN framework and found it to be 54% more scalable, thereby establishing the efficacy of distributed over centralized learning in such complex resource limited scenarios.

This chapter concludes the second part of the thesis. In the next one, a summary of the obtained results will be presented, along with the conclusive comments.

Chapter 6

Conclusions

In this thesis, we have seen how concepts such as Network Function Virtualization, Edge Computing and Machine Learning can be combined to improve network intelligence, flexibility and performance, paving the road ahead for vertical industries to reach mobile users with their innovative yet demanding services.

To answer the research questions listed in Sec 1.1, we investigated the potential of network automation taking separately two complementary approaches. First, we considered centralized automated service provisioning in 5G networks, where management of 5G services is performed at the core of the network. We examined the assurance of Service Level Agreements both with a simple threshold algorithm and leveraging ML-based approaches, showing excellent performance and the possibility of reducing the OPEX incurred by network operators and vertical industries by up to 30%. Then we explored the capabilities of managing network and compute resources at the edge using reinforcement learning, with the objective of meeting predefined KPIs. CAREM has been designed to manage heterogeneous vRANs, selecting in real-time the most appropriate radio link and radio parameters to use for each data packet. We demonstrated CAREM effectiveness and showed that it provides a 65% latency improvement relative to a competitive contextual bandit approach. VERA uses a similar learning framework to dynamically assign edge resources to network services and user applications, whose resource needs are often entangled. Considering as a network service an LTE vRAN and, as a user application, an adaptive video transcoder, we showed how the ML framework is able to learn the relationship between the radio and video actions. Moreover, to assure fairness when accounting for competing resource requests by the applications, a pareto

analysis has been integrated into VERA to avoid resource hogging. We observed that VERA is able to reach the target KPIs for over 96% of the observation periods.

As briefly mentioned in the Sec. 1.1, centralized edge service orchestration (Chapters 2 and 3) and distributed vRAN and user application management (Chapters 4 and 5), even though in this work they have been considered separately, can and should be utilized together to obtain the best possible results, since they perform different kind operations at different network levels. Specifically, in the first case, edge applications are managed on a coarse level to satisfy SLAs while avoiding resource waste, in the second case fine-grained actions are taken to optimize the real-time performance of the vRAN and edge applications. Notably, from several mobile network operators and research teams, there has already been a joint effort to design a unified RAN architecture able to accommodate the two approaches together, leveraging the synergy that their concurrent utilization can provide. The O-RAN alliance designed an architecture where non-real-time centralized intelligence can be provided by special purpose applications called rApps, which can directly operate on the RAN or coordinate with distributed near real-time applications, called xApps, to perform more fine-grained operations. Not only the O-RAN architecture envisions the dichotomy between centralized non-real-time and distributed near-real-time decisions, but it also directly integrates Machine Learning as a core component of the architecture specifications, paving the way for the pivotal role of this technology in the future of mobile networks.

6.1 Future Work and Open Challenges

Moving towards an architecture that unifies interfaces and procedures for centralized and distributed network management and control poses new challenges that the research community must address. The research effort focused on the O-RAN architecture is recently rising because of the clear advantages in managing and optimizing the RAN. In fact, relying on disaggregated, virtualized and software-based components would enable increased flexibility in deploying the network infrastructure, for improved reconfigurability and resiliency. Moreover, the network operators would be allowed to rely on interoperable network equipment of different vendors and avoid lock-ins, thus driving down CAPEX. However, this new paradigm, although it answers current mobile operators' needs, introduces many new research challenges that call for further investigation. Without any pretensions of being exhaustive, some interesting directions for future work within this scope are presented below.

Coexistence of virtualized services and vRAN The VERA framework presented in Chapter 5 has been designed to optimize the coexistence of the video-streaming user service and the vRAN at the edge. We showed that the resource demands of these two are tightly entangled, which is a behavior that could apply, albeit with different relationships, also to other user services. Further research is needed to better examine the coexistence of user applications and network services in the edge ecosystem, in order to investigate whether margins for better optimization exist.

O-RAN new use cases In Chapter 4 we proposed a reinforcement learning framework to perform radio resource management in heterogeneous RAN, then, in Chapter 5, we considered video transcoding and streaming at the edge, colocated with a vRAN, to build a learning framework able to optimize performance and resource consumption. The O-RAN intelligent, data-driven closed-loop control architecture, which allows for the radio parameter tuning according to the feedback from the mobile terminals, well suits the mentioned frameworks. Indeed, O-RAN specifications already consider these two use cases, as well as more advanced ones[179]. The identification of new use cases is currently ongoing and very relevant. As new use cases emerge, it will be necessary to investigate whether they are implementable with the current revision of the O-RAN specifications and, if they are not, propose the required additional features to make sure that the O-RAN architecture can properly allow for their implementation.

Application deployment optimization In a large-scale scenario, it is not unrealistic to imagine tens or hundreds of control applications running concurrently to optimize the different blocks that constitute the vRAN to achieve high-level intents set by the network operator. These applications are going to perform similar operations: collect and process RAN metrics, run optimization algorithms and then output an optimized action to be applied on the RAN. Many applications will execute a data processing pipeline common to other applications running at the same time, which could be shared to avoid resource waste. Some applications may also output partially colliding actions, thus requiring an arbitrator to avoid undesirable behaviors. To exploit synergies and avoid conflicts between applications, it is needed to design and develop dedicated orchestration frameworks, which can also ensure that each application run within its time budget. A preliminary step in this direction has been taken in [180], where the authors propose a framework to map network operators' intents to required control algorithms.

Real-time applications: dApps We have seen how the control algorithms and procedures presented in this work operate at different timescales. Centralized control algorithms, such as those demonstrated in Chapters 2 and 3, take actions that are applied in a matter of seconds or tens of seconds. Instead, distributed algorithms, such as CAREM and VERA (Chapter 4 and 5, respectively), work in timescales in the order of fractions of seconds. Though, it is still an open question whether a tighter time scale below 10 ms could be beneficial in reaching further optimization in the RAN operation. This real-time time scale is not mentioned yet in the O-RAN specification, but it is interesting because it would allow for fine-grained TTI-level optimization (e.g., FEC tuning, channel equalization, beamforming). In [181], the authors propose the concept of dApps, which are distributed applications to perform real-time inference and control in O-RAN.

Multi-time-scale integration Network management applications operate at different timescales according to the specific network component they intervene on. In the literature, many examples of rApps and xApps have been produced, however, there are not many examples of RAN services constituted by multiple applications running at different time scales. It would be interesting to investigate if a holistic approach to network management, in which applications at different time scales operate in synergy, would be able to achieve a deeper level of optimization and performance. The O-RAN specifications provide clear examples of the integration of different time scales with the Machine Learning models' lifecycle [182]. One of the most interesting presented scenarios foresees the training of an ML model in non-real-time since that is a usually long and demanding operation, followed by its deployment and execution close to the user where it can perform near-real-time inference.

AI/ML algorithms integration The integration of AI/ML algorithms in RAN management is a well-investigated topic with still many open challenges. Among them, it is worth mentioning (i) how to test or refine data-driven ML models without affecting production RAN performance and (ii) how to collect training datasets that are representative of all possible nuances of production large-scale deployments.

References

[1] Ericsson. Ericsson Mobility Report. https://www.ericsson.com/49d3a0/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-june-2022.pdf, June 2022. [Online; accessed 21-September-2022].

[2] ETSI. Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges & Call for Action. http://portal.etsi.org/NFV/NFV_White_Paper.pdf, October 2012. [Online; accessed 21-September-2022].

[3] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[4] EU H2020 5G-PPP 5G-TRANSFORMER project. 5G Mobile Transport Platform for Verticals. Available at: http://5g-transformer.eu/. [Online; Accessed: 21-September-2022].

[5] A. Oliva et al. 5G-TRANSFORMER: Slicing and Orchestrating Transport Networks for Industry Verticals. *IEEE Communications Magazine*, 56(8):78 – 84, 2018.

[6] ETSI. 3GPP TS 28.541, 5G Network Resource Model (NRM); Stage 2 and state 3 (Release 16), v16.4.1. 2020.

[7] Oscar Adamuz-Hinojosa, Pablo Munoz, Pablo Ameigeiras, and Juan M. Lopez-Soler. Sharing gNB components in RAN slicing: A perspective from 3GPP/NFV standards. *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct 2019.

[8] R. Ferrus, O. Sallent, and J. et al. Pérez-Romero. On the automation of RAN slicing provisioning: solution framework and applicability examples. *EURASIP Journal on Wireless Communications and Networking volume 2019*, Jun 2019.

[9] S. E. Elayoubi, S. B. Jemaa, Z. Altman, and A. Galindo-Serrano. 5G RAN Slicing for Verticals: Enablers and Challenges. *IEEE Communications Magazine*, 57(1):28–34, 2019.

[10] 3GPP. Technical Specification Group Services and System Aspects; Management and Orchestration; Concepts, use cases and requirements (Release 17), TS 28.530, v. 17.0.0, December 2020.

[11] J. Mangues-Bafalluy et al. 5G-TRANSFORMER Service Orchestrator: Design Implementation and Evaluation. In *Procs of the 28th European Conf. on Networks and Communications (EuCNC)*, pages 31 –36, June 2019.

[12] J. Baranda et al. Realizing the Network Service Federation Vision: Enabling Automated Multidomain Orchestration of Network Services. *IEEE Vehicular Technology Magazine*, 15(2):48–57, 2020.

[13] F. Paganelli, M. Ulema, and B. Martini. Context-aware Service Composition and Delivery in NGSONs over SDN. *IEEE Communications Magazine*, 52(8):97–105, 2014.

[14] S. Fichera et al. Latency-aware resource orchestration in sdn-based packet over optical flexi-grid transport networks. *IEEE/OSA Journal of Optical Communications and Networking*, 11(4):B83–B96, 2019.

[15] Wenfeng Xia, Peng Zhao, Yonggang Wen, and Haiyong Xie. A Survey on Data Center Networking (DCN): Infrastructure and Operations. *IEEE Comm. surveys & tutorials*, 19(1):640 – 656, 2017.

[16] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. Near optimal placement of virtual network functions. In *Procs of the IEEE Int. Conf. on Computer Communications (INFOCOM)*, pages 1346–1354, April 2015.

[17] Satyam Agarwal, Francesco Malandrino, Carla Fabiana Chiasserini, and Swades De. VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks. *IEEE/ACM Transactions on Networking*, 27(1):433 – 446, 2019.

[18] Leonard Kleinrock. *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[19] J. Mangues-Bafalluy et al. Experimental framework and evaluation of the 5G-Crosshaul Control Infrastructure. *Elsevier Computer Standards and Interfaces*, 64:96–105, 2019.

[20] G. Avino et al. A MEC-based Extended Virtual Sensing for Automotive Services. *IEEE Transactions on Network and Service Management*, 16(4):1450–1463, 2019.

[21] J.Baranda et al. Automated deployment and scaling of automotive safety services in 5G-Transformer. In *IEEE Conf. on Network Function Virtualization and Software Defined Networking, (NFV-SDN)*, pages 1–2, Nov. 2019.

[22] G. Avino et al. Support of Safety Services through Vehicular Communications: The Intersection Collision Avoidance Use Case. In *Procs of the IEEE International Conf. of Electrical and Electronic Technologies for Automotive*, pages 1–6, July 2018.

[23] M. Malinverno, G. Avino, C. Casetti, C. F. Chiasserini, F. Malandrino, and S. Scarpina. Edge-Based Collision Avoidance for Vehicles and Vulnerable Users: An Architecture Based on MECs. *IEEE Vehicular Technology Magazine*, 15(1):27 –35, 2019.

[24] P. A. Frangoudis, F. Giannone, A. Ksentini, and L. Valcarenghi. Orchestrating Heterogeneous MEC-based Applications for Connected Vehicles. *submitted to Elsevier Computer Networks*, 2020.

[25] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.

[26] W. Cerroni et al. Cross-layer Resource Orchestration for cloud service delivery: A seamless SDN approach. *Elsevier Computer Networks*, 87:16 – 32, 2015.

[27] M. Irfan et al. SLA (Service Level Agreement) Driven Orchestration Based New Methodology for Cloud Computing Services. In *Future Optical Materials and Circuit Design*, volume 660 of *Advanced Materials Research*, pages 196–201. Trans Tech Publications Ltd, 4 2013.

[28] Pankesh Patel, Ajith Ranabahu, and Amit P. Sheth. Service Level Agreement in Cloud Computing. In *Wright State University report*, 2009.

[29] Jennings R. Verma D., Beigi M. Policy Based SLA Management in Enterprise Networks. In *Lecture Notes in Computer Science*, 1995.

[30] F. Zulkernine, P. Martin, C. Craddock and K. Wilson. A Policy-Based Middleware for Web Services SLA Negotiation. In *IEEE International Conf. on Web Services (ICWS)*, pages 1043–1050, July 2009.

[31] P Bhoj, S Singhal, and S Chutani. SLA Management in Federated Environments. *Elsevier Computer Networks*, 35(1):5 – 24, 2001. Selected Topics in Network and Systems Management.

[32] M. A. Rahim, I. U. Haq, H. Durad and E. Schikuta. Generalized SLA Enforcement Framework Using Feedback Control System. In *Procs of the IEEE International Conf. on High-capacity Optical Networks and Enabling/Emerging Technologies (HONET)*, Dec. 2015.

[33] O. Adamuz-Hinojosa, J. Ordonez-Lucena, P. Ameigeiras, J. J. Ramos-Munoz, D. Lopez, and J. Folgueira. Automated Network Service Scaling in NFV: Concepts, Mechanisms and Scaling Workflow. In *IEEE Communications Magazine*, volume 56, pages 162–169, 2018.

[34] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, R. Boutaba. Topology-aware prediction of virtual network function resource requirements. *IEEE Transactions on Network and Service Management*, 14(1):106–120, 2017.

[35] J. G. Herrera and J. F. Botero. Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management*, 13(3):518 – 532, 2016.

[36] A. Boubendir, F. Guillemin, S. Kerboeuf, B. Orlandi, F. Faucheux and J. Lafragette. Network Slice Life-Cycle Management Towards Automation. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 709–711, April 2019.

[37] G. Xilouris et al. Towards autonomic policy-based network service deployment with sla and monitoring. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–2, 2018.

[38] M. Bouzid, D. H. Luong, D. Kostadinov, Y. Jin, L. Maggi, A. Outtagarts, and A. Aghasaryan. Cooperative ai-based e2e network slice scaling. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 959–960, 2019.

[39] J. Zhou, W. Zhao, and S. Chen. Dynamic network slice scaling assisted by prediction in 5g network. *IEEE Access*, 8:133700–133712, 2020.

[40] H. Khalili et al. Network slicing-aware NFV orchestration for 5G service platforms. In *European Conference on Networks and Communications (EuCNC)*, pages 25–30, 2019.

[41] V. Q. Rodriguez, F. Guillemin, and A. Boubendir. 5G E2E Network Slicing Management with ONAP. In *2020 23rd IEEE Conf. on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 87–94, Feb. 2020.

[42] H. Chergui and C. Verikoukis. Big data for 5g intelligent network slicing management. *IEEE Network*, 34(4):56–61, 2020.

[43] M. Iannelli, M. R. Rahman, N. Choi, and L. Wang. Applying machine learning to end-to-end slice sla decomposition. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 92–99, 2020.

[44] H. Chergui and C. Verikoukis. Offline sla-constrained deep learning for 5g networks reliable and dynamic end-to-end slicing. *IEEE Journal on Selected Areas in Communications*, 38(2):350–360, 2020.

[45] B. Khodapanah, A. Awada, I. Viering, D. Oehmann, M. Simsek, and G. P. Fettweis. Fulfillment of service level agreements via slice-aware radio resource management in 5g networks. In *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pages 1–6, 2018.

[46] D. De Vleeschauwer, C. Papagianni, and A. Walid. Decomposing slas for network slicing. *IEEE Communications Letters*, pages 1–1, 2020.

[47] A. Papageorgiou, A. Fernández-Fernández, L. Ochoa-Aday, M. S. Peláez, and M. Shuaib Siddiqui. Sla management procedures in 5g slicing-based systems. In *2020 European Conference on Networks and Communications (EuCNC)*, pages 7–11, 2020.

[48] F. Fossati, S. Moretti, and S. Secci. Multi-resource allocation for network slicing under service level agreements. In *2019 10th International Conference on Networks of the Future (NoF)*, pages 48–53, 2019.

[49] P. Trakadas et. al. Comparison of Management and Orchestration Solutions for the 5G Era. *MDPI, J. Sens. Actuator Networks*, 9(1):4, 2020.

[50] M. Touloupou, E. Kapassa, C. Symvoulidis, P. Stavrianos, and D. Kyriazis. An Integrated SLA Management Framework in a 5G Environment. In *2019 22nd IEEE Conf. on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 233–235, Feb. 2019.

[51] M. Fidler and V. Sander. A parameter based admission control for differentiated services networks. *Elsevier Computer Networks*, 44:463—479, 2004.

[52] M. H. Ahmed. Call admission control in wireless networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 7(1):49–68, 2005.

[53] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. Chen, and L. Hanzo. Machine learning paradigms for next-generation wireless networks. *IEEE Wireless Communications*, 24(2):98–105, 2017.

[54] T. Miyazawa, M. Jibiki, V. P. Kafle, and H. Harai. Autonomic resource arbitration and service-continuable network function migration along service function chains. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2018.

[55] C. Casetti et al. Arbitration among vertical services. In *Procs. of the IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 153–157, Sep. 2018.

[56] J. Baranda et al. On the Integration of AI/ML-based scaling operations in the 5Growth platform. In *Procs of the 6th IEEE Conference on Network Functions Virtualization and Software Defined Networking (IEEE NFV-SDN 2020), 10-12 November 2020*. IEEE, 2020.

[57] X. Li et al. 5Growth: An End-to-End Service Platform for Automated Deployment and Management of Vertical Services over 5G Networks. *IEEE Communications Magazine*, 59(3):84–90, March 2021.

[58] EU H2020 5G-PPP 5GROWTH project. 5G-enabled Growth in Vertical Industries. Available at: http://5growth.eu/. [Online; Accessed: 21-September-2022].

[59] 5G-CLARITY project: Initial design of the SDN/NFV platform and identification of target 5G-CLARITY ML algorithms. *D4.1*, 2020.

[60] ETSI. Zero-touch network and Service Management (ZSM); Reference Architecture. Technical report, 2019.

[61] 3GPP. O-RAN Working Group 2: AI/ML workflow description and requirements. Technical Report O-RAN.WG2.AIML, 2020.

[62] ETSI. GR NGP 011: Next Generation Protocols (NGP); E2E Network Slicing Reference Framework and Information Model. 2018.

[63] 3GPP. 3GPP TS 28.530 v17.11.0 - 5G; Management and orchestration; Concepts, use cases and requirements (Release 17). 2021.

[64] X. Li, F. Chiasserini, C. J. Mangues-Bafalluy, J. Baranda, G. Landi, B. Martini, X. Costa-Perez, P. Puligheddu, and L. Valcarenghi. Automated service provisioning and hierarchical SLA management in 5G systems. *IEEE Trans. on Network and System Management*, 18(4):4669–4684, 2021.

[65] 3GPP. 3GPP TS 28.533 v16.7.0 - Technical Specification Group Services and System Aspects; Management and orchestration; Architecture framework (Release 16). 2021.

[66] ETSI. NFV Release 2 Description. https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFV(21)000023_NFV_Release_2_Description_v1_12_0.pdf, [Accessed in December 2021].

[67] 3GPP. TS 28.536 v16.3.0 - Technical Specification Group Services and System Aspects; Management and orchestration; Management services for communication service assurance (Release 16). 2021.

[68] 3GPP. TS 23.288 v17.0.0 - Technical Specification Group Services and System Aspects; Architecture enhancements for 5G System (5GS) to support network data analytics services (Release 17). 2021.

[69] Michael J Neely, Eytan Modiano, and Chih-Ping Li. Fairness and optimal stochastic control for heterogeneous networks. *IEEE/ACM Trans. on Networking*, 16(2):396–409, 2008.

[70] Jing Bi, Zhiliang Zhu, Ruixiong Tian, and Qingbo Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In *IEEE CLOUD*, pages 370–377, 2010.

[71] Satyam Agarwal, Francesco Malandrino, Carla Fabiana Chiasserini, and Swades De. VNF placement and resource allocation for the support of vertical services in 5G networks. *IEEE/ACM Trans. on Networking*, 27(1):433–446, 2019.

[72] Jonathan Prados, Pablo Ameigeiras, Juan Jose Ramos-Munoz, Jorge Navarro-Ortiz, Pilar Andres-Maldonado, and Juan M Lopez-Soler. Performance modeling of softwarized network services based on queuing theory with experimental validation. *IEEE Trans. on Mobile Computing*, 20(4):1558–1573, 2021.

[73] Jonathan Prados-Garzon, Juan J Ramos-Munoz, Pablo Ameigeiras, Pilar Andres-Maldonado, and Juan M Lopez-Soler. Modeling and dimensioning of a virtualized MME for 5G mobile networks. *IEEE Trans. on Vehicular Technology*, 66(5):4383–4395, 2017.

[74] J. Baranda and et al. On the integration of AI/ML-based scaling operations in the 5Growth platform. In *IEEE NFV-SDN*, pages 105–109, 2020.

[75] 5G-Transformer. D1.1, Report on Vertical Requirements and Use Cases. Technical Report, 2018.

[76] M. Malinverno, J. Mangues, C. Casetti, C.F. Chiasserini, M. Requena, and J. Baranda. An Edge-based Framework for Enhanced Road Safety of Connected Cars. *IEEE Access, March 2020*, 8:58018–58031, March 2020.

[77] François Rameau, Hyowon Ha, Kyungdon Joo, Jinsoo Choi, Kibaek Park, and In So Kweon. A real-time augmented reality system to see-through cars. *IEEE Trans. on Visualization and Computer Graphics*, 22(11):2395–2404, 2016.

[78] Marco Rapelli, Claudio Casetti, and Giandomenico Gagliardi. Vehicular traffic simulation in the city of turin from raw data. *IEEE Trans. on Mobile Computing*, pages 1–1, 2021.

[79] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009.

[80] 5GROWTH. D2.3, Final Design and Evaluation of the innovations of the 5G End-to-End Service Platform. Technical Report, 2021.

[81] Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K. Marina. Network slicing in 5G: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100, 2017.

[82] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hinesd. 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167, February 2020.

[83] Fadoua Debbabi, Rihab Jmal, Lamia Chaari Fourati, and Adlen Ksentini. Algorithmics and modeling aspects of network slicing in 5G and beyonds network: Survey. *IEEE Access*, 8:162748–162762, 2020.

[84] Mohammed Chahbar, Gladys Diaz, Abdulhalim Dandoush, Christophe Cérin, and Kamal Ghoumid. A comprehensive survey on the E2E 5G network slicing model. *IEEE Trans. on Network and Service Management*, 18(1):49–62, 2021.

[85] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Trans. on Network and Service Management*, 13(3):518–532, 2016.

[86] Hamzeh Khalili and et al. Network slicing-aware NFV orchestration for 5G service platforms. In *EuCNC*, pages 25–30, 2019.

[87] David M. Gutierrez-Estevez, N. Dipietro, A. Dedomenico, M. Gramaglia, U. Elzur, and Y. Wang. 5G-MoNArch use case for ETSI ENI: elastic resource management and orchestration. In *IEEE CSCN*, pages 1–5, 2018.

[88] MATILDA project: Intelligent orchestration mechanisms. *D3.2*, 2020.

[89] MATILDA project: 5G-ready vertical applications orchestration. *White paper*, Dec. 2019.

[90] R. Bruschi, F. Davoli, C. Lombardo, and J. F. Pajo. Managing 5G network slicing and edge computing with the MATILDA telecom layer platform. *Computer Networks*, 194, July 2021.

[91] David M. Gutierrez-Estevez and et al. Artificial intelligence for elastic management and orchestration of 5G networks. *IEEE Wireless Communications*, 26(5):134–141, 2019.

[92] Qiang Liu and Tao Han. VirtualEdge: Multi-domain resource orchestration and virtualization in cellular edge computing. In *IEEE ICDCS*, pages 1051–1060, 2019.

[93] Fengsheng Wei, Gang Feng, Yao Sun, Yatong Wang, Shuang Qin, and Ying-Chang Liang. Network slice reconfiguration by exploiting deep reinforcement learning with large action space. *IEEE Trans. on Network and Service Management*, 17(4):2197–2211, 2020.

[94] Ibrahim Afolabi, Jonathan Prados-Garzon, Miloud Bagaa, Tarik Taleb, and Pablo Ameigeiras. Dynamic resource provisioning of a scalable E2E network slicing orchestration system. *IEEE Trans. on Mobile Computing*, 19(11):2594–2608, 2020.

[95] Sabidur Rahman, Tanjila Ahmed, Minh Huynh, Massimo Tornatore, and Biswanath Mukherjee. Auto-scaling network service chains using machine learning and negotiation game. *IEEE Trans. on Network and Service Management*, 17(3):1322–1336, 2020.

[96] László Toka, Gergely Dobreff, Balázs Fodor, and Balázs Sonkoly. Machine learning-based scaling management for Kubernetes edge clusters. *IEEE Trans. on Network and Service Management*, 18(1):958–972, 2021.

[97] Hui Yu, Jiahai Yang, and Carol Fung. Fine-grained cloud resource provisioning for virtual network function. *IEEE Trans. on Network and Service Management*, 17(3):1363–1376, 2020.

[98] Lianjie Cao, Puneet Sharma, Sonia Fahmy, and Vinay Saxena. ENVI: Elastic resource flexing for network function virtualization. In *USENIX HotCloud*, Santa Clara, CA, July 2017.

[99] H. Chergui and C. Verikoukis. Offline SLA-constrained deep learning for 5G networks reliable and dynamic end-to-end slicing. *IEEE Journal on Selected Areas in Communications*, 38(2):350–360, 2020.

[100] Dario Bega, Marco Gramaglia, Albert Banchs, Vincenzo Sciancalepore, Konstantinos Samdanis, and Xavier Costa-Perez. Optimising 5G infrastructure markets: The business of network slicing. In *IEEE INFOCOM*, pages 1–9, 2017.

[101] Bin Han, Vincenzo Sciancalepore, Xavier Costa-Pérez, Di Feng, and Hans D. Schotten. Multiservice-based network slicing orchestration with impatient tenants. *IEEE Trans. on Wireless Communications*, 19(7):5010–5024, 2020.

[102] Tulja Vamshi Kiran Buyakar, Harsh Agarwal, Bheemarjuna Reddy Tamma, and A Antony Franklin. Resource allocation with admission control for GBR and delay QoS in 5G network slices. In *COMSNETS*, pages 213–220, 2020.

[103] J. Baranda, J. Mangues-Bafalluy, E. Zeydan, C. Casetti, C. F. Chiasserini, M. Malinverno, C. Puligheddu, M. Groshev, C. Guimaraes, K. Tomakh, and O. Kolodiazhnyi. Demo: AIML-as-a-Service for SLA management of a Digital Twin virtual network service. In *IEEE INFOCOM - Demo Session*, 2021.

[104] J. G. Andrews and et al. What will 5G be? *IEEE J. Sel. Areas Commun.*, 32(6):1065–1082, June 2014.

[105] C. Wang and et al. Cellular architecture and key technologies for 5G wireless communication networks. *IEEE Commun. Mag.*, 52(2):122–130, Feb. 2014.

[106] T. O. Olwal, K. Djouani, and A. M. Kurien. A survey of resource management toward 5G radio access networks. *IEEE Commun. Surveys Tuts.*, 18(3):1656–1686, 2016.

[107] C. Liang and F. R. Yu. Wireless network virtualization: A survey, some research issues and challenges. *IEEE Commun. Surveys Tuts.*, 17(1):358–380, 2015.

[108] K. Tsagkaris, G. Poulios, P. Demestichas, A. Tall, Z. Altman, and C. Destré. An open framework for programmable, self-managed radio access networks. *IEEE Commun. Mag.*, 53(7):154–161, 2015.

[109] Jose A. Ayala-Romero, Andres Garcia-Saavedra, Marco Gramaglia, Xavier Costa-Perez, Albert Banchs, and Juan J. Alcaraz. VrAIn: a deep learning approach tailoring computing and radio resources in virtualized RANs. In *ACM MobiCom*, New York, NY, USA, 2019.

[110] Ericsson. 5G Radio Access. *Ericsson Rev.*, 6:1–8, June 2014.

[111] Y. Fu, S. Wang, C. Wang, X. Hong, and S. McLaughlin. Artificial intelligence to manage network traffic of 5G wireless networks. *IEEE Netw.*, 32(6):58–64, 2018.

[112] F. Hussain, S. A. Hassan, R. Hussain, and E. Hossain. Machine learning for resource management in cellular and IoT networks: Potentials, current solutions, and open challenges. *IEEE Commun. Surveys Tuts.*, 22(2):1251–1275, 2020.

[113] F. Tang, Y. Kawamoto, N. Kato, and J. Liu. Future intelligent and secure vehicular network toward 6G: Machine-learning approaches. *Proc. IEEE*, 108(2):292–307, 2020.

[114] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang. Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges. *IEEE Veh. Technol. Mag.*, 14(2):44–52, 2019.

[115] R. Zheng and C. Hua. Sequential learning and decision-making in wireless resource management. *Wireless Networks*, 2016.

[116] 3GPP TS 23.501 V16.3.0 Technical Specification Group Services and System Aspects; System Architecture for the 5G System (5GS); Stage 2, (Release 16) , 12 2019.

[117] A. Slivkins. Introduction to multi-armed bandits. *Foundations and Trends in Machine Learning*, 12(1-2):1–286, 2019.

[118] B. Jang, M. Kim, G. Harerimana, and J. W. Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019.

[119] R. Combes and A. Proutiere. Dynamic Rate and Channel Selection in Cognitive Radio Systems. *IEEE J. Sel. Areas Commun.*, 33(5):910–921, May 2015.

[120] R. Combes, J. Ok, A. Proutiere, D. Yun, and Y. Yi. Optimal Rate Sampling in 802.11 Systems: Theory, Design, and Implementation. *IEEE Trans. Mobile Comput.*, 18(5):1145–1158, May 2019.

[121] H. Gupta, A. Eryilmaz, and R. Srikant. Low-complexity, Low-regret Link Rate Selection in Rapidly-varying Wireless Channels. In *IEEE Conf. Comput. Commun.*, pages 540–548, 2018.

[122] J. Ma, T. Nagatsuma, S. Kim, and M. Hasegawa. A machine-learning-based channel assignment algorithm for IoT. In *ICAIIC*, pages 1–6, 2019.

[123] S. Hasegawa, S. Kim, Y. Shoji, and M. Hasegawa. Performance evaluation of machine learning based channel selection algorithm implemented on IoT sensor devices in coexisting IoT networks. In *IEEE CCNC*, pages 1–5, 2020.

[124] M. A. Qureshi and C. Tekin. Fast learning for dynamic resource allocation in AI-enabled radio networks. *IEEE Trans. Cogn. Commun. Netw.*, 6(1):95–110, 2020.

[125] M. El Helou and et al. A network-assisted approach for RAT selection in heterogeneous cellular networks. *IEEE J. Sel. Areas Commun.*, 33(6):1055–1067, 2015.

[126] D. D. Nguyen, H. X. Nguyen, and L. B. White. Reinforcement learning with network-assisted feedback for heterogeneous RAT selection. *IEEE Trans. Wireless Commun.*, 16(9):6062–6076, 2017.

[127] Y. Wei, R. Yu F. M. Song, and Z. Han. User scheduling and resource allocation in HetNets with hybrid energy supply: An actor-critic reinforcement learning approach. *IEEE Trans. Wireless Commun.*, 17(1):680–692, 2018.

[128] N. Morozs, T. Clarke, and D. Grace. Heuristically accelerated reinforcement learning for dynamic secondary spectrum sharing. *IEEE Access*, 3:2771–2783, 2015.

[129] V. Raj, I. Dias, T. Tholeti, and S. Kalyani. Spectrum access in cognitive radio using a two-stage reinforcement learning approach. *IEEE J. Sel. Topics Signal Process.*, 12(1):20–34, 2018.

[130] I. Comşa and et al. Towards 5G: A reinforcement learning-based scheduling solution for data traffic management. *IEEE Trans. Netw. Service Manag.*, 15(4):1661–1675, 2018.

[131] I. Comșa, R. Trestian, G. Muntean, and G. Ghinea. 5MART: A 5G SMART scheduling framework for optimizing QoS through reinforcement learning. *IEEE Trans. Netw. Service Manag.*, 17(2):1110–1124, 2020.

[132] Y. Zhou, F. Tang, Y. Kawamoto, and N. Kato. Reinforcement learning-based radio resource control in 5G vehicular network. *IEEE Wireless Commun. Lett.*, 9(5):611–614, 2020.

[133] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari. Deep reinforcement learning for dynamic multichannel access in wireless networks. *IEEE Trans. Cogn. Commun. Netw.*, 4(2):257–265, 2018.

[134] O. Naparstek and K. Cohen. Deep multi-user reinforcement learning for distributed dynamic spectrum access. *IEEE Trans. Wireless Commun.*, 18(1):310–323, 2019.

[135] C. Zhong, Z. Lu, M. C. Gursoy, and S. Velipasalar. A deep actor-critic reinforcement learning framework for dynamic multichannel access. *IEEE Trans. Cogn. Commun. Netw.*, 5(4):1125–1139, Nov. 2019.

[136] L. Zhang, J. Tan, Y. Liang, G. Feng, and D. Niyato. Deep reinforcement learning-based modulation and coding scheme selection in cognitive heterogeneous networks. *IEEE Trans. Wireless Commun.*, 18(6):3281–3294, 2019.

[137] X. Li, J. Fang, W. Cheng, H. Duan, Z. Chen, and H. Li. Intelligent power control for spectrum sharing in cognitive radios: A deep reinforcement learning approach. *IEEE Access*, 6:25463–25473, 2018.

[138] Y. S. Nasir and D. Guo. Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks. *IEEE J. Sel. Areas Commun.*, 37(10):2239–2250, 2019.

[139] S. Gyawali, Y. Qian, and R. Q. Hu. Resource allocation in vehicular communications using graph and deep reinforcement learning. In *IEEE GLOBECOM*, pages 1–6, Dec. 2019.

[140] X. Chen and et al. Age of information aware radio resource management in vehicular networks: A proactive deep reinforcement learning perspective. *IEEE Trans. Wireless Commun.*, 19(4):2268–2281, 2020.

[141] H. Ye, G. Y. Li, and B. F. Juang. Deep reinforcement learning based resource allocation for V2V communications. *IEEE Trans. Veh. Technol.*, 68(4):3163–3173, 2019.

[142] X. Zhang, M. Peng, S. Yan, and Y. Sun. Deep reinforcement learning based mode selection and resource allocation for cellular V2X communications. *IEEE Internet Things J.*, pages 1–1, Dec. 2019.

[143] S. Tripath, C. Puligheddu, and C.F. Chiasserini. An RL approach to radio resource management in heterogeneous virtual RANs. In *IEEE/IFIP WONS*, 2021.

[144] 5G mobile transport platform for verticals. *5G PPP H2020 5G-TRANSFORMER Project*, 2019.

[145] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[146] Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In Jean-Daniel Zucker and Lorenza Saitta, editors, *Abstraction, Reformulation and Approximation*, pages 194–205, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[147] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[148] Ismael Gomez-Miguelez and et al. SrsLTE: an open-source platform for LTE evolution and experimentation. In *ACM WiNTECH*, page 25–32, 2016.

[149] Bastian Bloessl, Michele Segata, Christoph Sommer, and Falko Dressler. Performance assessment of IEEE 802.11p with an open source SDR-based prototype. *IEEE Trans. Mobile Comput.*, 17(5):1162–1175, May 2018.

[150] Lihong Li, Wei Chu, J. Langford, and R. Schapire. A Contextual-bandit Approach to Personalized News Article Recommendation. In *ACM WWW*, 2010.

[151] Andres Garcia-Saavedra and Xavier Costa-Pérez. O-RAN: Disrupting the virtualized RAN ecosystem. *IEEE Communications Standards Magazine*, 5(4):96–103, 2021.

[152] Open RAN Alliance. O-RAN: Towards an Open and Smart RAN. *White Paper*, 2018.

[153] Cisco, Rakuten, Altiostar. Reimagining the End-to-End Mobile Network in the 5G Era. *White Paper*, 2019.

[154] Samsung. Virtualized Radio Access Network: Architecture, Key technologies and Benefits. *Technical Report*, 2019.

[155] Intel. vRAN: The Next Step in Network Transformation. *White Paper*, 2017.

[156] Gines Garcia-Aviles et al. Nuberu: Reliable ran virtualization in shared platforms. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 749–761, New York, NY, USA, 2021. Association for Computing Machinery.

[157] Nokia. The edge cloud: An agile foundation to support advanced new services. *White Paper*, 2018.

[158] Andres Garcia-Saavedra, George Iosifidis, Xavier Costa-Perez, and Douglas J Leith. Joint optimization of edge computing architectures and radio access networks. *IEEE Journal on Selected Areas in Communications*, 36(11):2433–2443, 2018.

[159] Jose A. Ayala-Romero, Andres Garcia-Saavedra, Marco Gramaglia, Xavier Costa-Perez, Albert Banchs, and Juan J. Alcaraz. vrAIn: Deep learning based orchestration for computing and radio resources in vRANs. *IEEE Transactions on Mobile Computing*, pages 1–1, 2020.

[160] Jose A. Ayala-Romero et al. EdgeBOL: automating energy-savings for mobile edge AI. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pages 397–410, 2021.

[161] Jose A. Ayala-Romero et al. Bayesian online learning for energy-aware resource orchestration in virtualized RANs. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[162] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[163] S. Arora, P. A. Frangoudis, and A. Ksentini. Exposing radio network information in a mec-in-nfv environment: the rnisaas concept. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 306–310, 2019.

[164] G. Avino, P. Bande, P. A. Frangoudis, C. Vitale, C. Casetti, C. F. Chiasserini, K. Gebru, A. Ksentini, and G. Zennaro. A MEC-based extended virtual sensing for automotive services. *IEEE Transactions on Network and Service Management*, 16(4):1450–1463, 2019.

[165] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. srsLTE: An open-source platform for LTE evolution and experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, pages 25–32, 2016.

[166] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward a practical perceptual video quality metric. *The Netflix Tech Blog*, 6(2), 2016.

[167] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[168] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[169] S. Pasteris, S. Wang, M. Herbster, and T. He. Service placement with provable guarantees in heterogeneous edge computing systems. In *IEEE INFOCOM*, pages 514–522, 2019.

[170] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas. Joint service placement and request routing in multi-cell mobile edge computing networks. In *IEEE INFOCOM*, pages 10–18, 2019.

[171] X. Ma, A. Zhou, S. Zhang, and S. Wang. Cooperative service caching and workload scheduling in mobile edge computing. In *IEEE INFOCOM*, pages 2076–2085, 2020.

[172] F. Wang, C. Zhang, F. wang, J. Liu, Y. Zhu, H. Pang, and L. Sun. Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe. In *IEEE INFOCOM*, pages 910–918, 2019.

[173] Q. Liu, S. Huang, J. Opadere, and T. Han. An edge network orchestrator for mobile augmented reality. In *IEEE INFOCOM*, pages 756–764, 2018.

[174] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics. In *IEEE INFOCOM*, pages 257–266, 2020.

[175] Y. He, J. Ren, G. Yu, and Y. Cai. Optimizing the learning performance in mobile augmented reality systems with CNN. *IEEE Transactions on Wireless Communications*, 19(8):5333–5344, 2020.

[176] Sharda Tripathi, Corrado Puligheddu, Carla Fabiana Chiasserini, and Federico Mungari. A context-aware radio resource management in heterogeneous virtual rans. volume 8, pages 321–334, 2022.

[177] Luca Baldesi, Francesco Restuccia, and Tommaso Melodia. Charm: Nextg spectrum sharing through data-driven real-time o-ran dynamic control. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 2022.

[178] S. Tripathi, C. Puligheddu, S. Pramanik, A. Garcia-Saavedra, and C. F. Chiasserini. VERA: Resource orchestration for virtualized services at the edge. In *2022 IEEE International Conference on Communications (ICC)*, 2022.

[179] O-RAN Working Group 1. Use cases analysis report. Technical report, O-RAN Alliance, October 2022.

[180] Salvatore D'Oro, Leonardo Bonati, Michele Polese, and Tommaso Melodia. Orchestran: Network automation through orchestrated intelligence in the open ran. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 270–279, 2022.

[181] Salvatore D'Oro, Michele Polese, Leonardo Bonati, Hai Cheng, and Tommaso Melodia. dapps: Distributed applications for real-time inference and control in o-ran. *IEEE Communications Magazine*, 60(11):52–58, 2022.

[182] O-RAN Working Group 2. Ai/ml workflow description and requirements. Technical report, O-RAN Alliance, October 2021.