Deep Learning solutions for next generation slicing-aware mobile networks

by

Dario Bega

A dissertation submitted by in partial fulfillment of the requirements for the degree of Doctor of Philosophy in

Telematics Engineering

Universidad Carlos III de Madrid

Advisor: Albert Banchs
Co-Advisor: Xavier Costa-Pérez

January 2020

Deep Learning solutions for next generation slicing-aware mobile networks

Prepared by:
Dario Bega, IMDEA Networks Institute, Universidad Carlos III de Madrid
contact: dario.bega@imdea.org

Under the advice of:
Albert Banchs, IMDEA Networks Institute
Telematics Engineering Department, Universidad Carlos III de Madrid
Xavier Costa-Pérez, NEC Laboratories Europe

This work has been supported by:

"If we knew what it was we were doing, it would not be called research, would it?"

Albert Einstein

"I have never met a man so ignorant that I couldn't learn something from him"

Galileo Galilei

"Che cosè il genio? È fantasia, intuizione, decisione e velocità d'esecuzione"

Amici Miei

Acknowledgements

Before presenting this work, I would like to start by thanking everyone who have helped me during this journey. First I would like to express my profound gratitude to my Ph.D supervisor Prof. Albert Banchs for his sincere guidance and support. He gave me the possibility to become his student when I was not expecting it anymore and thanks to his patience, suggestions and teachings helped me to gain a scientific thinking. Without him, all my modest research achievements wouldn't have been possible.

Next, I would like to thank Dr. Marco Gramaglia. Even though he support AC Milan (not the best period for that), he represented for me not only a great researcher with whom discuss every doubt, challenge or idea I had in mind, but also a true friend always ready to give me helpful advices. The works included in this thesis are as yours as mine.

I am very grateful to Dr. Marco Fiore and Dr. Xavier Costa-Peréz for their collaboration and valuable comments. You drove me to achievements I could not imagine at the beginning.

Moreover, I would like to thank Prof. Tommaso Melodia for hosting me for 6 months at Northeastern University giving me the opportunity to work in a top-level research lab and live a wonderful life experience. I cannot forget to thanks all the amazing people working at WiNESLab that welcomed me and made my stay in Boston unforgettable. Thanks to Lorenzo, Salvatore, Leonardo, Frank, Daniel, Jennie, Sara, Emrecan, Ludo and Norbert for the 6 months of fun we had.

Furthermore, I would like to thank all my colleagues at IMDEA Networks. You made my Ph.D an indelible experience. I would like to thank Roberto (yes I learned a lot from you but remember that you are becoming a number 4), Mauri (also for sharing the early morning gym), Chri (even if he hates Inter I always enjoy ours footbal discussions), Ev (I miss a lot your daily craziness), Noelia (my murcian/spanish teacher with all your stories and new trends to follow), Danilo (for our talks rigorously in italian), Hany (for our jokes), Giulia (always with your positive mood) and Francesco (seriously man, it has not been easy to watch your face all the mornings in the last months) for all the talks, coffees, dinners, beers, parties and all the breaks spent together. We met as colleagues but you quickly became part of my Spanish family. I would like also to thank Paolo, Ander, Dolores, Pablo, Adriana, Jesus and all the amazing people I met during these years at

IMDEA. You are definitely the soul of the institute. I would like to thank my all-life friends in Italy Vincenzo, Romiz, Mazzo, Trovaz, Andrea, Bedo, Dario, Luca, Claudio and Lisa for our light-hearted time.

Quindi voglio ringraziare mia madre e mio padre per aver fatto l'impossibile per darmi tutto quello che ho e per avermi sempre sostenuto e incoraggiato in ogni passo della mia vita. Siete il mio punto di riferimento e se oggi sono arrivato fino a qui è solo grazie a voi. Vorrei inoltre ringraziare mia sorella, che nonostante le nostre innumerevoli discussioni per i più futili motivi, è stata per me un esempio da seguire.

Finally, I would like to thank Alessia for being for half of my life at my side even when no-one would have stayed. We met long time ago and from that moment you brought your smiley, joyful and full of energy personality in my life. Thanks for your support and to foster my willing to continuously move around the world. You are a pillar of my life and without you I wouldn't neither have begun this journey.

Published and Submitted Content

This thesis is based on the following published papers:

[1] **Dario Bega**, Marco Gramaglia, Albert Banchs, Vincenzo Sciancalepore, Konstantinos Samdanis, Xavier Costa-Perez. "Optimising 5G infrastructure markets: The Business of Network Slicing". Published in *the 36th IEEE International Conference on Computer Communications (IEEE INFOCOM 2017)*, 1-4 May 2017, Atlanta, GA, USA. https://doi.org/10.1109/INFOCOM.2017.8057045

This work is fully included and its content is reported in Chapter 3.

The author's role in this work is focused on the design, implementation and experimentation of a reinforcement learning algorithm with regarding of the concepts proposed in the paper.

[2] Vincenzo Sciancalepore, Konstantinos Samdanis, Xavier Costa-Perez, **Dario Bega**, Marco Gramaglia, Albert Banchs. "Mobile Traffic Forecasting for Maximizing 5G Network Slicing Resource Utilization". Published in *the 36th IEEE International Conference on Computer Communications (IEEE INFOCOM 2017)*, 1-4 May 2017, Atlanta, GA, USA. https://doi.org/10.1109/INFOCOM.2017.8057230

This work is partially included and its content is reported in Chapter 3.

The author's role in this work is focused on participating in the discussion about the development of the concepts proposed in the paper.

[3] Peter Rost, Christian Mannweiler, Diomidis Michalopoulos, Cinzia Sartori, Vincenzo Sciancalepore, Nishanth Sastry, Oliver Holland, Shreya Tayade, Bin Han, **Dario Bega**, Danish Aziz, Hajo Bakker. "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks". Published in *IEEE Communications Magazine, 55(5), pp. 72-79. ISSN 0163-6804*, May 2017. https://doi.org/10.1109/MCOM.2017.1600920

This work is partially included in this thesis and its content is reported in Chapter 5.

The author's role in this work is focused on collaborating in the writing of network slicing approaches for 5G mobile networks.

[4] **Dario Bega**, Marco Gramaglia, Carlos Jesus Bernardos Cano, Albert Banchs, Xavier Costa-Perez. "Toward the network of the future: From enabling technologies to 5G concepts". Published in *Transactions on Emerging Telecommunications Technologies, 28(8), pp. 1-12.*, August 2017. https://doi.org/10.1002/ett.3205

This work is partially included in this thesis and its content is reported in Chapter 1.

The author's role in this work is focused on the collection and writing of the paper's concepts.

[5] Diomidis S. Michalopoulos, Mark Doll, Vincenzo Sciancalepore, **Dario Bega**, Peter Schneider, Peter Rost. "Network slicing via function decomposition and flexible network design". Published in *the 28th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 8-13 October 2017, Montreal, QC, Canada. https://doi.org/10.1109/PIMRC.2017.8292661

This work is partially included in Chapter 2.

The author's role in this work is focused on providing and writing contributions regarding mobile network architecture to support network slicing.

[6] Pablo Serrano, Marco Gramaglia, **Dario Bega**, David Gutierrez-Estevez, Gines Garcia-Aviles, Albert Banchs. "The path toward a cloud-aware mobile network protocol stack". Published in *Transactions on Emerging Telecommunications Technologies, 29(5), pp. 1-10.*, May 2018. https://doi.org/10.1002/ett.3312

This work is partially included in Chapter 1.

The author's role in this work is focused on contributing with performance evaluation results.

[7] **Dario Bega**, Albert Banchs, Marco Gramaglia, Xavier Costa-Perez, Peter Rost. "CARES: Computation-aware Scheduling in Virtualized Radio Access Networks". Published in *IEEE Transactions on Wireless Communications, 17(12), pp. 7993-8006. ISSN 1536-1276*, December 2018. https://doi.org/10.1109/TWC.2018.2873324

This work is partially included and its content is reported in Chapter 1.

The author's role in this work is focused on the design and experimentation of a computational-aware scheduling algorithm.

[8] **Dario Bega**, Marco Gramaglia, Albert Banchs, Vincenzo Sciancalepore, Xavier Costa-Perez. "A Machine Learning approach to 5G Infrastructure Market optimization". Published in *IEEE Transactions on Mobile Computing, pp. 1-15. ISSN 1536-1233*, 01 February 2019. https://doi.org/10.1109/TMC.2019.2896950

This work is fully included and its content is reported in Section 3.4.4.

The author's role in this work is focused on the design, implementation and experimentation of a deep learning framework with regarding of the concepts proposed in the paper.

[9] Dario Bega, Marco Gramaglia, Marco Fiore, Albert Banchs, Xavier Costa-Perez. "DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning". Published in *the 38th IEEE International Conference on Computer Communications (IEEE INFOCOM 2019)*, 29 April - 02 May 2019, Paris, France. https://doi.org/10.1109/INFOCOM.2019.8737488

This work is fully included and its content is reported in Chapter 4.

The author's role in this work is focused on the design, implementation and experimentation of a Deep Learning framework for the proposed methodology.

[10] Dario Bega, Marco Gramaglia, Marco Fiore, Albert Banchs, Xavier Costa-Perez. "α-OMC: Cost-Aware Deep Learning for Mobile Network Resource Orchestration". Published in *the 38th IEEE International Conference on Computer Communications Workshops (IEEE INFOCOM WKSHPS 2019)*, 29 April - 02 May 2019, Paris, France. https://doi.org/10.1109/INFCOMW.2019.8845178

This work is fully included and its content is reported in Section 4.4.

The author's role in this work is focused on the design, implementation and experimentation of the proposed loss function for Deep Learning framework.

[11] Dario Bega, Marco Gramaglia, Marco Fiore, Albert Banchs, Xavier Costa-Perez. "DeepCog: Optimizing Resource Provisioning in Network Slicing with AI-based Capacity Forecasting". Accepted for publication in *IEEE Journal on Selected Areas in Communications special issue on Leveraging Machine Learning in SDN/NFV-based Networks (IEEE JSAC SI-MLinSDNNFV). ISSN 0733-8716*, December 2019. https://doi.org/10.1109/JSAC.2019.2959245

This work is fully included and its content is reported in Section 4.3 and Section 4.5.

The author's role in this work is focused on the design, implementation and experimentation of a Deep Learning approach with regarding of the concepts proposed in the paper.

[12] Dario Bega, Marco Gramaglia, Marco Fiore, Albert Banchs, Xavier Costa-Perez. "AZTEC: Anticipatory Capacity Allocation for Zero-Touch Network Slicing". Accepted for publication in *the 39th IEEE International Conference on Computer Communications (IEEE INFOCOM 2020)*, 27 - 30 April 2020, Beijijng, China.

This work is fully included and its content is reported in Chapter 5.

The author's role in this work is focused on the design, implementation and experimentation of a novel Deep Learning - Optimization framework for capacity allocation.

[13] Dario Bega, Marco Gramaglia, Andres Garcia-Saavedra, Marco Fiore, Albert Banchs, Xavier Costa-Perez. "Network Slicing Meets Artificial Intelligence: an AI-based Framework for Slice Management". This work has been submitted for publication.

This work is fully included and its content is reported in Chapter 1.

The author's role in this work is focused on collaborating in the writing of the magazine and providing some of the described use-cases.

[14] Dario Bega, Albert Banchs, Marco Gramaglia, Marco Fiore, Ramon Perez, Xavier Costa-Perez. "AI-based Autonomous Control, Management, and Orchestration in 5G: from Standards to Algorithms". This work has been submitted for publication.

This work is not included in this thesis.

The author's role in this work is focused on the design, implementation and experimentation of the AI framework for 5G network management and orchestration.

Abstract

It is now commonly agreed that future 5G Networks will build upon the network slicing concept. Network slicing is an emerging paradigm in mobile networks that leverages Network Function Virtualization (NFV) to enable the instantiation of multiple logically independent copies -named *slices*- of a same physical network infrastructure. The operator can allocate to each slice dedicated resources and customized functions that allow meeting the highly heterogeneous and stringent requirements of modern mobile services. Managing functions and resources under network slicing is a challenging task that requires making efficient decisions at all network levels and in real-time, which can be achieved by integrating Artificial Intelligence (AI) in the network.

This thesis investigates the potential of AI for sliced mobile networks. In particular it focuses on resource allocation and orchestration for network slices. This involves two steps: (i)*Admission Control* that is responsible to decide which slices can be admitted to the network, and (ii) *Network resource orchestration* that dynamically allots to the admitted slices the necessary resources for their operation.

Network Slicing will have an impact on the models that sustain the business ecosystem opening the door to new players: the Infrastructure Provider (InP), which is the owner of the infrastructure, and the tenants, which may acquire a network slice from the InP to deliver specific service to their customers. In this new context, how to correctly handle resource allocation among tenants and how to maximize the monetization of the infrastructure become fundamental problems that need to be solved. In this thesis we address this issue by designing a network slice admission control algorithm that (i) autonomously learns the best acceptance policy while (ii) it ensures that the service guarantees provided to tenants are always satisfied. This includes (i) an analytical model for the admissibility region of a network slicing-capable 5G Network, (ii) the analysis of the system (modeled as a Semi-Markov Decision Process) and the optimization of the infrastructure provider's revenue, and (iii) the design of a machine learning algorithm that can be deployed in practical settings and achieves close to optimal performance.

Dynamically orchestrate network resources is both a critical and challenging task in upcoming multi-tenant mobile networks, which requires allocating capacity to individual network slices so as to accommodate future time-varying service demands. Such an

anticipatory resource configuration process must be driven by suitable predictors that take into account all the sources of monetary cost associated to network capacity orchestration. Legacy models that aim at forecasting traffic demands fail to capture these key economic aspects of network operation. To close this gap in the second part of this thesis, we first present DeepCog, a first generation deep neural network architecture inspired by advances in image processing and trained via a dedicated loss function in order to deal with monetary cost due to overprovisioning or underprovisioning of networking capacity. Unlike traditional traffic volume predictors, DeepCog returns a cost-aware *capacity forecast*, which can be directly used by operators to take short- and long-term reallocation decisions that maximize their revenues. Extensive performance evaluations with real-world measurement data collected in a metropolitan-scale operational mobile network demonstrate the effectiveness of our proposed solution, which can reduce resource management costs by over 50% in practical case studies. Then we introduce `AZTEC`, a second generation data-driven framework that effectively allocates capacity to individual slices by adopting an original multi-timescale forecasting model. Hinging on a combination of Deep Learning architectures and a traditional optimization algorithm, `AZTEC` anticipates resource assignments that minimize the comprehensive management costs induced by resource overprovisioning, instantiation and reconfiguration, as well as by denied traffic demands. Experiments with real-world mobile data traffic show that `AZTEC` dynamically adapts to traffic fluctuations, and largely outperforms state-of-the-art solutions for network resource orchestration.

At the time of writing DeepCog and `AZTEC` are, to the best of our knowledge, the only works where a deep learning architecture is explicitly tailored to the problem of anticipatory resource orchestration in mobile networks.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AI Artificial Intelligence

BBU baseband unit

CN Core Network

CNN Convolutional Neural Network

C-RAN Cloud Radio Access Networks

CSC Communication Service Client

CSP Communication Service Provider

DNN Deep Neural Network

DRL Deep Reinforcement Learning

DPI Deep Packet Inspection

eMBB Enhanced Mobile Broadband

EPC Evolved Packet Core

FC Fully Connected

GTP GPRS Tunneling Protocol

IaaS Infrastructure as a Service

InP Infrastructure Provider

KPI Key Performance Indicator

MAC Medium Access Control

MAE Mean Absolute Error

MANO management and orchestration

mMTC massive Machine Type Communication

MNO Mobile Network Operator

MCS Modulation and Coding Scheme

MDS Multi-Dimensional Scaling

ML Machine Learning

MEC Mobile Edge Computing

MLP Multi-Layer Perceptron

MSE Mean Squared Error

N3AC Network-slicing Neural Network Admission Control

NN Neural Network

NFV Network Function Virtualization

NFVI Network Functions Virtualization Infrastructure

NF Network Function

OPEX operating expenses

OSS/BSS Operations and Business Support System

PHY physical layer

PRB Physical Resource Block

QoE Quality of Experience

QOS Quality of Service

RAN Radio Access Network

RAT Radio Access Technology

ReLU Rectified Linear Unit

RL Reinforcement Learning

RNN Recurrent Neural Network

RWP Random Waypoint

SDN Software Defined Networking

SMDP Semi-Markov Decision Process

SLA Service Level Agreement

SGD Stochastic Gradient Descent

UE User Equipment

URLLC Ultra Reliable Low Latency Communication

UPF User Plane Function

VIM Virtual Infrastructure Manager

VM Virtual Machine

VNF Virtual Network Function

JCR Journal Citation Reports

1 Introduction

The expectations that build around future 5G Networks are very high, as the envisioned Key Performance Indicators (KPIs) represent a giant leap when compared to legacy 4G/LTE networks. Very high data rates, extensive coverage, sub-ms delays are just few of the performance metrics that 5G networks are expected to boost when deployed.

This game changer relies on new technical enablers such as Software Defined Networking (SDN) or Network Function Virtualization (NFV) that will bring the network architecture from a purely *hardbox* based paradigm (*e.g.*, a eNodeB or a Packet Gateway) to a completely *cloudified* approach, in which network functions that formerly were hardware-based (*e.g.*, baseband processing, mobility management) are implemented as software NFVs running on a, possibly hierarichical, general purpose *telco-cloud*.

Building on these enablers, several novel key concepts have been proposed for next generation 5G networks [4] (*e.g.*, *virtualized Radio Access Network (RAN)* or *Network Slicing*) requiring a re-design of current network functionalities. Among them, *Network Slicing* [14] is probably the most important one. Indeed, there is a wide consensus in that accommodating the very diverse requirements demanded by 5G services using the same infrastructure will not be possible with the current, relatively monolithic architecture in a cost efficient way. In contrast, with network slicing the infrastructure can be divided in different slices, each of which can be tailored to meet specific service requirements.

A network slice consists of a set of Virtual Network Functions (VNFs) that run on a virtual network infrastructure and provide a specific telecommunication service. The services provided are usually typified in macro-categories, depending on the most important KPIs they target. Enhanced Mobile Broadband (eMBB), massive Machine Type Communication (mMTC) or Ultra Reliable Low Latency Communication (URLLC) are the type of services currently envisioned by, *e.g.*, ITU [15]. Each of these services is instantiated in a specific network slice, which has especially tailored management and orchestration algorithms to perform the lifecycle management within the slice.

In this way, heterogeneous services may be provided using the same infrastructure, as

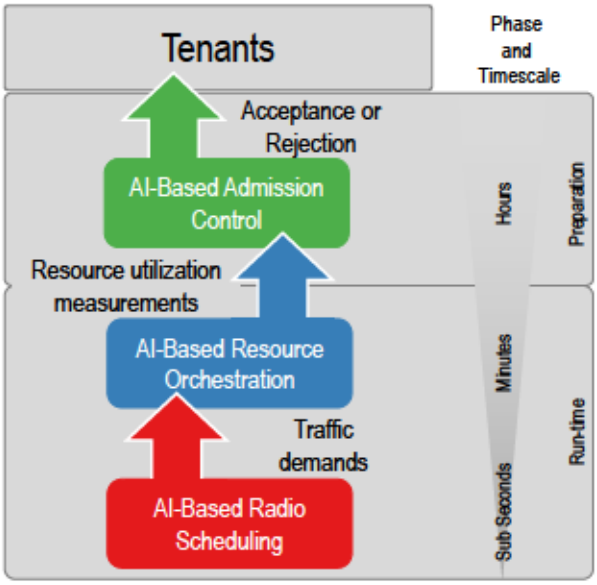


Figure 1.1: Comprehensive network slicing framework. The diagram outlines the timescales and composition of the key slice management functions.

different telecommunication services (that are mapped to a specific slice) can be configured independently according to their specific requirements. Additionally, the *cloudification* of the network allows for the cost-efficient customization of network slices, as the slices run on a shared infrastructure.

1.1. AI-based Slice Management Framework

Much of the complexity in re-designing mobile networks for slicing relates to decision-making towards an efficient, dynamic management of resources in real-time. There, Artificial Intelligence (AI) appears as a natural approach to design the various algorithms employerd by different network functions [16]. As a matter of fact, AI provides a powerful tool to address highly complex problems that involve large amounts of data. This is indeed the case with network slicing, where the presence of a large number of slices each independently operated by a different tenant, drastically increases the complexity of the system with respect to legacy non-sliced networks controlled by a single entity, *i.e.*, the network operator. Also, the sheer amount of data flowing through the network and potentially relevant to resource allocation decision-making, and the difficulties in forecasting the overall behavior of a system involving many different players, make traditional tools for network management insufficient.

Devising a network slicing framework requires novel algorithms to manage the infrastructure resources, sharing them between the different slices while guaranteeing

the requirements of each slice will be met [13]. This applies throughout the following network functions:

- *Admission Control* is in charge of deciding whether upcoming network slice requests can be admitted or not in the system, and is enacted so as to ensure that the requirements of the admitted slices are satisfied.

- *Network (re-)orchestration* is central to both slice instantiation and run-time operation, since it allocates the available network resources to the admitted slices in the most efficient way possible, and then dynamically updates such an allocation at run-time in order to fulfill the time-varying demands of each slice while avoiding capacity outages.

- *Radio resource sharing* is paramount at run-time, as it manages the sharing of radio access resources among the network slices, ensuring that potentially stringent requirements of all the slices (*e.g.*, in terms of latency and throughput) are met over the air interface.

Figure 1.1 shows the framework that we envisage in this thesis to support network slicing. This framework gathers components to deal with each of the above functions. These components involve different phases of the "Network Slice Lifecycle Management" [17], consisting of four main steps that have to be addressed: (*i*) preparation; (*ii*) instantiation, configuration and activation; (*iii*) run-time and (*iv*) de-commissioning. Each of these phases involves different timescales: (*i*) admission control runs at frequencies that match those of arrivals of new network slices requests, which may be in the order of hours; (*ii*) the orchestration of resources in softwarized networks occurs at frequencies that depend on the time required to re-sizing virtual machines resources, typically in the order of minutes; and, (*iii*) scheduling of radio resources applies at a finer granularity, down to millisecond intervals in extreme cases. It is also worth highlighting that the different functions (and their associated algorithms) may benefit from mutual interactions. The information on resource utilization gathered at the network orchestration level can be leveraged for admission control, where it allows understanding whether admitting a new slice may lead to problems in provisioning enough capacity for all admitted services. Similarly, data collected by the resource management function can help an orchestrator to produce more accurate forecasts of future slice demands for anticipatory resource allocation.

All functions above need to make decisions to meet the requirements of the individual slices while maximizing the overall system performance. To this end, they need to learn the dynamics of per-slice data traffic, and automatically react to their impact on the network architecture, towards their respective management goals. Self-adapting network function configurations were introduced over a decade ago [18], however the solutions

designed so far typically apply control on limited sets of parameters that change slowly in time (*e.g.*, eNB transmission power). Also, current approaches produce outputs that then need human intervention to be translated into modifications of the network configuration (*e.g.*, updating the transport network so as to optimize handovers in a given region).

These characteristics are not compatible with the novel requirements introduced by network slicing. The parameters that may need reconfiguration are much more numerous, as each virtual network functions may expose several of them in a programmatic way. The timescale at which decisions must be made is drastically reduced, as one must be ideally capable of acting at radio level timings or even at wire-speed. Decisions often need to take into account metrics that go beyond pure network performance, such as energy efficiency or infrastructure monetization, which may hide complex cross-relationships.

This context provides a fertile ground for AI to become instrumental in mobile network operation. All classes of AI may be useful to this end, including (*i*) *supervised* solutions that require ground truth data for training, (*ii*) *unsupervised* techniques that work in absence of ground truth, and (*iii*) *reinforcement* learning approaches where different forms of interaction with the system that has to be controlled are possible [19]. The most appropriate AI tools must be selected case by case, depending on the involved algorithmic requirements and operation timescales.

For instance, reinforcement learning is particularly well suited when the time dynamics of the problem can accommodate a learning curve, and the objective is to define a sequence of actions that maximizes a certain reward: this is the case of admission control algorithm as demonstrated by the practical implementations presented in Section 3.4.4. Conversely, when the target is to provide decisions that are independent of those previously taken and whose quality can be assessed during systems training, supervised learning solutions are a strong option: this is precisely the settings where network resource orchestration takes place, as illustrated by the applied solution in Section 4.3 and Section 5.3.

Before proceeding further, we remark that those presented next are examples of successful integration of AI across the framework in Figure 1.1. They do not exhaust the application space of AI for network operations; rather, they realize important components in the comprehensive design of self-organizing sliced mobile networks.

1.2. Challenges

As described in the section above, the integration of the key novel concepts envisioned for 5G networks will affect the current mobile network functions. In this thesis we focus on Network Slicing and in particular on the re-design of two fundamental network functions, *i.e.*, Admission Control and Resource Orchestration. Each of them conveys several challenges that need to be taken account of while designing viable implementations of slice management functions. Following we provides more details about the research

challenges of these works.

1.2.1. Admission Control

Network infrastructure resources are limited and network slice demand quality guarantees, which calls for admission control on new slice requests. According to 3GPP standardization on network slicing, the Communication Service Client (CSC) [17], *i.e.*, the tenant, will request specific services to the Communication Service Provider (CSP), *i.e.*, the network provider, among those available in the offered portfolio. Then, it will pay for the service according to metrics like, *e.g.*, the number of served users, the service coverage area, or the duration of the slice instance. Such admission control decisions have profound business implications: the choice of how many network slices to run simultaneously, and how to share the network infrastructure among slices have an impact on the revenues of the network provider.

The complexity and heterogeneity of the slice admission decision process deprecates manual configuration, which is the *de-facto* legacy approach in 4G networks. To identify the best operating point, slice admission control must learn the arrival dynamics of slices and make decisions that maximize the revenue, based on the current system occupation and its expected long-term evolution. This problem is highly dimensional (growing linearly with the number of network slices) with a potential huge number of states (increasing exponentially with the number of classes) and many variables (one for each state). Furthermore, in many cases the behavior of the tenants that request slices is not known *a priori* and may vary with time. For these reasons, traditional solutions building on optimization techniques are not affordable (because of complexity reasons) or simply impossible (when slice behavior is not known). Instead, AI provides a means to cope with such complex problems.

1.2.2. Network Orchestration

Once admitted, slices must be allocated sufficient resources. Due to the prevailing softwarization of mobile networks, such resources are increasingly of computational nature. This holds both at the RAN where they map to, *e.g.*, CPU time for containers running baseband units (BBUs) in Cloud Radio Access Networks (C-RAN) datacenters, and in the Core Network (CN) where, *e.g.*, virtual machines run softwarized Evolved Packet Core (EPC) entities in datacenters. In these case, ensuring strong KPI guarantees often requires that computational resources are exclusively allocated to specific slices, and cannot be shared across others. The dynamic allocation of network resources to the different admitted slices becomes then a chief management task in network slicing.

In this context, the network operator needs to decide in advance the amount of resources that should be dedicated to the different slices, so as to ensure that the

available capacity is used in the most efficient way possible and thus minimize operating expenses (OPEX). Finding the correct operational point requires (*i*) predicting the future demand in each slice, and (*ii*) deciding what amount of resources is needed to serve such demand. These two problems are complex per-se: forecasting future demands at service level requires designing dedicated, accurate predictors; instead, allocating resources in a way that minimizes the OPEX of the operator requires estimating the expected error of the prediction. Moreover, addressing (*i*) and (*ii*) above as separate problems risks to lead to largely suboptimal solutions, since legacy predictors do not provide reliable information about the expected error they will incur into. **W**hile the complexity of the complete solution may be daunting with traditional techniques, AI can be leveraged to address both aspects at once.

1.3. Contributions

This thesis investigates the application of deep learning solutions for next generation sliced mobile networks. The main contributions of this doctoral thesis have been published in 8 publications, of which 1 has been published in *IEEE Communication Magazine* (indexed in Journal Citation Reports (JCR)), 1 in *IEEE Transactions on Mobile Computing* (indexed in JCR), 1 in *IEEE Transactions on Wireless Communications* (indexed in JCR), 1 in *IEEE JSAC special issue on Leveraging Machine Learning in SDN/NFV-based Networks* (indexed in JCR), 2 in *Transactions on Emerging Telecommunications Technologies* (indexed in JCR), and 2 submitted to *IEEE Communication Magazine* and *IEEE Network Magazine* currently under revision. Other 4 publications have been published in tier-1 conference *IEEE INFOCOM* according to CORE2014[1] or ERA2010[2] datasets, 1 publication have been published in *IEEE INFOCOM* workshop. In details,

1. **Admissibility region analytical formulation.** An analytical model for the admissibility region of a network slicing-capable 5G Network has been devised. It provides to the Infrastructure Provider (InP) the information about the maximum number of network slices can be admitted in the system to maximize his revenue while guaranteeing that the Service Level Agreements (SLAs) are met for all tenants. This is a fundamental information for the InP, indeed if admitting a new network slice in the system would lead to violating the SLA of already admitted slices, then such a request should be rejected.

2. **Decision-making optimal and adaptive algorithms design.** The decision-making process on slice requests has been modeled as Semi-Markov Decision Process

[1] http://portal.core.edu.au/conf-ranks/
[2] http://www.conferenceranks.com/

(SMDP), including the definition of the state space of the system, along with the decisions that can be taken at each state and the resulting revenue. This is used to derive the optimal admission control policy that maximizes the revenue of the infrastructure provider, which serves as a benchmark for the performance evaluation, and an adaptive algorithm that provides close to optimal performance.

3. **Machine Learning based admission control algorithm.** Admission control represents a very complex task. This problem is highly dimensional with a potential huge number of states and many variables. Optimal methods require that all variables are known, and as adaptive algorithms do not scale to huge space states. Machine Learning provides a mean to cope with such complex problems, consequently we have designed a practicable Neural Networks (NNs) solution based on deep reinforcement learning that provides close to optimal performance.

4. **Capacity Forecast.** Legacy techniques for the prediction of mobile network traffic aim at perfectly matching the temporal behavior of traffic, independently of whether the anticipated demand is above or below the target. As a result, they incur in substantial SLA violations. Hence, we introduce the notion of capacity forecast, *i.e.*, the minimum provisioned capacity needed to cut down SLA violations. This closes the gap between simple traffic prediction and practical orchestration.

5. **A Deep Learning Framework for Resource Orchestration.** A new mobile traffic data analytics tool explicitly tailored to solve capacity forecast problem has been designed. It hinges on a deep learning architecture that leverages a customized loss function that targets capacity forecast rather than plain mobile traffic prediction. It also provides long-term forecasts over configurable prediction horizons operating on a per-service base.

6. **Two-time scale anticipatory capacity allocation for network slicing with hard guarantees.** An original model for the anticipatory allocation of capacity to network slices, which is mindful of all operating costs is proposed. The new model takes into account not only the orchestration costs associated with over- and under-provisioning but also the ones linked with resource instantiation and reconfiguration. `AZTEC`, a complete framework for capacity allocation to network slices has been designed. It relies on a combination of deep learning architectures and traditional optimizer.

1.4. Outline of the thesis

The rest of the thesis is organized as follows. Within each chapter, a list of relevant state-of-the-art works are provided. We first describe in Chapter 2 how Network Slice

will modify the current mobile network architecture in order to fully exploit its potential benefits. Then, we investigate the impact of Network Slicing over Admission Control, presenting in Section 3.2 our system model and the analytical formulation for the network slice admissibility region. Building on this model, in Section 3.3 we address the problem of designing an admission control algorithm that maximizes the InP revenue while satisfying the desired service guarantees; to this end, we first analyze the revenue resulting from a given admission control policy and then obtain the optimal policy that maximizes the resulting revenue. Building on this analysis, we design a practical adaptive algorithm that provides close to optimal performance. In Section 3.4.4 we present a NNs approach based on deep reinforcement learning, which provides a practical and scalable solution with close to optimal performance. Finally, in Section 3.5 we evaluate the proposed algorithms in a number of scenario to assess their performance in terms of optimality, scalability and adaptability to different conditions.

Then, we study the impact of Network Slicing over Resource Orchestration. First, the concept of capacity forecast is introduced in Section 4.1. Then, we outline the overall framework of DeepCog and detail the design of its most critical component, *i.e.*, the loss function, in Section 4.3. The quality of the solution is then assessed in realistic scenario in Section 4.5. To complete our analysis over Resource Orchestration, in Section 5.1 we motivate the need of including also the costs derived by resource instantiation and reconfiguration. We present the new orchestration model, formalizing the different costs and trade-offs in the resource management of sliced networks in Section 5.2. Building on such a new model, we develop a complete framework for the anticipatory allocation of capacity to network slices, named `AZTEC`; the framework relies on a combination of deep learning architecture and a traditional optimizer, as detailed in Section 5.3. We demonstrated the quality of the solution with real-world measurements data collected in a metropolitan-scale mobile network in Section 5.6.

Finally, Chapter 6 draws the most important conclusions of this research work.

2 Trends and challenges in network slicing

Network Slicing represents an efficient solution that addresses the diverse requirements of 5G mobile networks, providing the necessary flexibility and scalability associated with future network implementations.

To fully exploit the benefits provided by network slicing, for each slice should be possible to design customized network operation or flexible Network Functions (NFs) each optimized for the particular usage. Thus, employing network slicing in 5G networks results into a number of challenges, in part due to difficulties in virtualizing and apportioning the Radio Access Network (RAN) into different slices.

In the following we first describe the realization options of a flexible mobile network architecture with focus on network slicing and their impact on the design of 5G mobile networks. Then the potential design challenges associated with the implementation of network slicing in future networks are analyzed.

2.1. Mobile Network Slicing Architecture

To fully exploit the potential of network slicing, there is the need to design a new more flexible mobile network architecture. Indeed, the current relatively monolithic and static architecture, where network functions rely on dedicated hardware and are placed in independent entities (*e.g.*, eNB, S-GW or MME), is not able to accommodate in a cost efficient way such heterogeneous services with very diverse requirements as the ones demanded by 5G.

There is a wide consensum [3, 5, 20, 21] that the new 5G architecture must support softwarization for allowing the sharing of the network among multiple tenants. In this new vision, a logical network would be implemented as a set of individual softwarized network functions customized for each network slice. A slice then becomes a composition of function blocks into a chain or more generally into a network of function blocks. Decomposition into function blocks enables sharing of network functions among slices for reuse and consistency, or where common resources must be shared. A slice may be

partly composed of a set of common function blocks to be shared across slices and a set of dedicated function blocks that implement customized and optimized functionality of a slice. Furthermore, decomposition enables the function blocks of a slice to be placed according to its service needs and the concrete deployment scenario, *i.e.*, the available execution environments such as distributed (edge) or centralized resources.

Network slices will then operate on top of a partially shared infrastructure, which is composed of generic hardware resources such as Network Functions Virtualization Infrastructure (NFVI) resources, as well as dedicated hardware such as network elements in the RAN.

Generally, three solution groups are discussed with varying levels of common functionality in 3GPP standards [22]:

- **Group A** is characterized by a common RAN and completely dedicated Core Network (CN) slices, that is, independent subscription, session, and mobility management for each network slice handling the User Equipment (UE).

- **Group B** also assumes a common RAN, where identity, subscription, and mobility management are common across all network slices, while other functions such as session management reside in individual network slices.

- **Group C** assumes a completely shared RAN and a common CN control plane, while CN user planes belong to dedicated slices.

In line with the above grouping considered by 3GPP [22], in [3] we introduce new dedicated network functions, which together form a dedicated sub-slice, to cope with NFs sharing among slices. This represent a possible solution to manage shared NFs reflecting the fact that these functions have to coordinate and, if necessary, prioritize the Quality of Service (QOS) requirements of multiple slices.

Realize the above vision entails multiple challenges that are further discussed in the following section.

2.2. Design Challenges

Network slicing allows multiple Mobile Network Operators (MNOs) to share the same network infrastructure. To enable network slicing, the design of a new mobile network architecture is required as detailed above. This entails facing multiple challenges:

- In the concept envisioned by [14] no distinctions are made regarding the type of resources shared. From the various type of resources, spectrum represents typically the most limiting factor: despite of cloud resources which can be easily scaled up if needed, spectrum is limited and increasing its capacity is more complex and more expensive. Thus, RAN sharing quickly runs into a *physical* constraint. For this

reason, 5G sliced network calls for new mechanisms for managing slices admission control in an optimized way as the one proposed in Chapter 3.

- It is expected that 5G will incorporate several kinds of Radio Access Technologys (RATs) and air interfaces, each with different capabilities and needs. General-purpose infrastructure providers will need to carefully plan and apply different technologies to serve diverse tenant needs. However, it may be infeasible to satisfy the needs of each application at any location. For instance, tactile Internet may require careful positioning of resources to minimize latency. In another example, an industrial control network might have to use a certain computational resource in a given location for security reasons. There is clearly the need of automated tools, drove by network data analytic, that anticipate to the Infrastructure Provider (InP) the information regarding the resources needed by any slice at any location in the network as the ones proposed in Chapter 4 and Chapter 5.

- Depending on the extent of network elements that are shared among tenants, different flavors of network slices can be defined ranging from slices that share only the physical layer (PHY), or even the Medium Access Control (MAC) or the complete RAN. The more information that can be provided by the infrastructure about the shared parts to the network slice, the more efficiently the slice can be operated. However, exposing information also creates new potential security vulnerabilities between InP and tenants as well as between tenants themselves. Security requirements of specific tenant applications, such as traffic associated with emergency services or machine control (e.g., remote surgery or vehicular control), could put constraints on how the slices are partitioned, or even prevent network slices coexisting and thus share the same hardware at all.

- A major question is whether a slice can be extended all the way to the UE; that is, whether the definition of the slice will be transparent to the UE, or whether the UE will be aware of the network slice. A slicing-aware UE may open up new possibilities (*e.g.*, simplification of multi-slice connectivity). However, it also creates new challenges for network slices; for example, UE mobility may need to be handled by the slice provider as part of the slice setup and maintenance.

- Network slicing in 5G networks enables a new ecosystem in which different tenants issue requests to an InP for acquiring network slices. Since spectrum is a scarce resource for which overprovisioning is not possible, applying an "always accept" strategy for all incoming requests is not feasible. This calls for novel algorithms and solutions to allocate network resources among different tenants, allowing an InP to accept or reject network slice requests with the objective of

maximizing the overall utility. We further investigate this problem in Chapter 3.

- When a network supports multiple tenants by creating tenant specific network slice instances, it is necessary to isolate these slice instances in a way that one tenant is not aware of the other tenants and has no means to access or even modify information in the other tenants' slices. In Network Function Virtualization (NFV) environments, this type of isolation is a basic feature that also includes the capability to limit the resource usage of each tenant slice instance in a well-defined way. This prevents a tenant from using so many resources that other tenants cannot get resources anymore and thus experience a Denial of Service (DoS).

- A likely scenario leveraged by network slicing is that a MNO provides individual network slice instances for verticals. The MNO can provide isolation between the tenant slices as described above. However, a vertical as a tenant typically has no means to verify the effectiveness of the isolation, but must trust the MNO to ensure it. Moreover, if the MNO controls the infrastructure, is also able to access everything that is processed on this infrastructure. Consequently, the MNO must be trusted to not illegally access a tenant's traffic. Trust in the MNO is also required concerning the correct resource assignment, since a tenant has no practical means to monitor the correct assignment of edge cloud infrastructure resources or radio interface resources to the tenant's slice.

- Common functions that are operated by a MNO and used by several tenants must be protected. Any internal interfaces of such functions must not be accessible for tenant functions. Only dedicated, carefully secured interfaces must be available to tenants. Such interfaces may need to be subject to access control, which includes authenticating the slices accessing the common functions, as well as authorizing their requests. An example could be a tenant requesting certain QOS parameters for a radio bearer. In this case, a common function may check whether the request is covered by the tenant's Service Level Agreement (SLA). Moreover, tenants may inadvertently misuse or even deliberately try to abuse interfaces exposed by common functions. To mitigate this threat, such interfaces must be designed and implemented with high care to minimize their vulnerability.

In the rest of this thesis, we further analyze the impact of network slicing over the current mobile network functionalities and design new approaches to enable slicing for future 5G networks.

3 Network Slice Admission Control

As discussed in Chapter 1, Network Slicing is probably one of the most important key concept for next generation 5G networks [4]. This novel approach does not just provide better performing and more efficient networks, but enables a new business model around mobile networks, involving new entities and opening up new business opportunities. Network slices allow for a role separation between *Infrastructure Providers (InPs)*, who provide computational and network resources used by different network slice, and *network slice tenants*, the ones acquiring a slice to orchestrate and run network functions within that slice to provide a certain service to their customer. In this new context, how to correctly decide which slices can be admitted to the network and how to maximize the monetization of the infrastructure become fundamental problems that need to be solved.

The above model is currently being successfully applied by Infrastructure as a Service (IaaS) providers such as Amazon Web Services or Microsoft Azure, which sell their computational resources such as CPU, disk or memory for Virtual Network Function (VNF) purposes. While such an IaaS approach follows a very similar business model to the network slicing one, providing network resources is an intrinsically different problem, since (*i*) spectrum is a scarce resource for which over-provisioning is not possible, (*ii*) the actual capacity of the systems (i.e., the resources that can actually be sold) heavily depends on the mobility patterns of the users, and (*iii*) the Service Level Agreements (SLAs) with network slices tenants usually impose stringent requirements on the Quality of Experience (QoE) perceived by their users. Therefore, in contrast to IaaS, in this case applying a strategy where all the requests coming to the InP are admitted is simply not possible.

While there is a body of work on the literature on spectrum sharing [23–26], these proposal are not tailored to the specific requirements of the 5G ecosystem. Conversely, most of the work has focused on architectural aspects [27,28] with only a limited focus on resource allocation algorithms. In [2], we provide an analysis of network slicing admission control and propose a learning algorithm; however, the proposed algorithm relies on an offline approach, which is not suitable for a continuously varying environment such as

the 5G Infrastructure market. Moreover, the aim is to maximize the overall network utilization, in contrast to our goal here which is focused on maximizing InP revenues.

The need for new algorithms that specifically targets the monetization of the network has been identified in [29]. However, there are still very few works on this topic. The work in [30] analyzes the problem from an economical perspective, proposing a revenue model for the InP. The authors of [31] build an economic model that describes the Mobile Network Operator (MNO) profit when dealing with the network slice admission control problem, and propose a decision strategy to maximize the expected overall network profit. The proposed approach, however, is not on demand and requires the full knowledge of arriving requests statistics, thus making it impracticable in real scenarios. Another work in this field is the one of [30], with similar limitations.

In the above context, the new 5G ecosystem calls for novel algorithms and solutions for the allocation of the (scarce) network resources among tenants; this is the so-called spectrum market. In this chapter, a network capacity brokering solution is introduced. In Section 3.2, we first introduce our system model and the analytical formulation for the network slice admissibility region. In Section 3.3, we address the problem of designing an admission control algorithm that maximizes the InP revenue while satisfying the desired service guarantees. To this end, we first model the decision-making process by means of a Markovian analysis, and derive the optimal policy and then design an adaptive algorithm that provides close to optimal performance. Finally, in Section 3.4.4 we present a Neural Networks (NNs) approach based on deep reinforcement learning, which provides a practical and scalable solution with close to optimal performance.

3.1. System Model

The high customizability that 5G Networks introduce will enable a richer ecosystem on both the portfolio of available services and the possible business relationships. New players are expected to join the 5G market, leading to an ecosystem that is composed of (*i*) users that are subscribed to a given service provided by a (*ii*) tenant that, in turn, uses the resources (*i.e.*, cloud, spectrum) provided by an (*iii*) Infrastructure Provider (InP). The design of a network slice admission control policy that maximizes InPs revenues in this spectrum market is still an open problem. The 5G Network Slice Broker [2] is a novel element introduced in the network management system of the InP for advanced Radio Access Network (RAN) sharing. It exploits 3GPP conventional monitoring procedures for gathering global network load measurements, to map incoming Service Level Agreement (SLA) requirements associated to network slice requests into physical resources. More specifically, the network capacity broker algorithm has to decide on whether to admit or reject new network slice requests simultaneously satisfying two different goals:

- meeting the service guarantees requested by the network slices admitted

- maximizing the revenue of a network InP.

The goal of meeting the desired service guarantees needs to consider radio related aspects, as a congested network will likely not be able to meet the service required by a network slice. Conversely, the goal of maximizing the revenue obtained by the admission control should be met by applying an on-demand algorithm that updates the policies as long as new requests arrive.

In the rest of the section, we describe the various aspects related to our system model, while the analytical formulation for the network admissibility region is provided in Section 3.2. In Section 3.3, we model the decision-making process by means of a Markovian analysis, and derive the optimal policy which we use as a benchmark. Building on this analysis, we first design an adaptive algorithm in Section 3.4.2, and then present a deep reinforcement learning approach in Section 3.4.4 which provides a practical and scalable solution with close to optimal performance.

Players. In our system model, there are the following players: (i) the *InP*, who is the owner of the network and provides *network slices* corresponding to a certain fraction of network resources to the tenants, (ii) the *tenants*, which issue requests to the InP to acquire network resources, and use these resources to serve their users, providing them a specific telecommunication service, and finally (iii) the *end-users*, which are subscribers of the service provided by a tenant which uses the resources of the InP.

Network model. The ecosystem described above does not make any distinction on the kind of resources an InP may provide to the tenants. From the various types of resources, spectrum will typically be the most important factor when taking a decision on whether to accept a request from a tenant. Indeed, cloud resources are easier to provision, while increasing the spectrum capacity is more complex and more expensive (involving an increase on antenna densification). Based on this, we focus on the wireless access network as the most limiting factor. In our model of the wireless access, the network has a set of base stations $\mathcal{B}$ owned by an InP. For each base station $b \in \mathcal{B}$, we let C_b denote the base station capacity. We further refer to the system capacity as the sum of the capacity of all base stations, $C = \sum_{\mathcal{B}} C_b$. We let $\mathcal{U}$ denote the set of end-users in the network, each of them being served by one of the tenants. We consider that each user $u \in \mathcal{U}$ in the system is associated to one base station $b \in \mathcal{B}$. We denote by f_{ub} the fraction of the resources of base station b assigned to user u, leading to a throughput for user u of $r_u = f_{ub}C_b$. We also assume that users are distributed among base stations according to a given probability distribution; we denote by $P_{u,b}$ the probability that user u is associated with base station b. We assume that these are independent probabilities, i.e., each user behaves independently from the others.

Traffic model. 5G Networks provide diverse services which are mapped to three different usage scenarios or slice categories: Enhanced Mobile Broadband (eMBB), massive Machine Type Communication (mMTC) and Ultra Reliable Low Latency Communication

(URLLC) [15]. As the main bottleneck from a resource infrastructure market point of view is spectrum, different slice categories need to be matched based to their requirements in terms of the spectrum usage. For instance eMBB-alike slices have a higher flexibility with respect to resource usage, and can use the leftover capacity of URLLC services which have more stringent requirements on the needed capacity.

Following the above, we focus on elastic and inelastic traffic as it is the main distinguishing factor for spectrum usage and thus provides a fairly large level of generality. In line with previous work in the literature [32], we consider that inelastic end-users require a certain fixed throughput demand which needs to be satisfied *at all times*, in contrast to elastic users which only need guarantees on the *average* throughput, requiring that the expected average throughput over long time scales is above a certain threshold. That is, for inelastic users throughput needs to be always (or with a very high probability) above the guaranteed rate, while the throughput for elastic users is allowed to fall below the guaranteed rate during some periods as long as the average stays above this value.

We let $\mathcal{I}$ denote the set of classes of inelastic users; each class $i \in \mathcal{I}$ has a different rate guarantee R_i which needs to be satisfied with a very high probability; we refer the probability that this rate is not met as the outage probability, and impose that it cannot exceed P_{out}, which is set to a very small value. We further let N_i denote the number of inelastic users of class $i \in \mathcal{I}$, and $P_{i,b}$ be the probability that a user of class i is at base station b. Finally, we let N_e be the number of elastic users in the network and R_e their average rate guarantee.

At any given point in time, the resources of each base stations are distributed among associated users as follows: inelastic users $u \in \mathcal{I}$ are provided sufficient resources to guarantee $r_u = R_i$, while the remaining resources are equally shared among the elastic users. In case there are not sufficient resources to satisfy the requirements of inelastic users, even when leaving elastic users with no throughput, we reject as many inelastic users as needed to satisfy the required throughput guarantees of the remaining ones.

Note that the above traffic types are well aligned with the slice categories defined in 3GPP, as the elastic traffic behavior is in line with the eMBB and mMTC services, while inelastic behavior matches the requirements of URLCC services.

Network slice model. The network is divided into different logical slices, each of them belonging to one tenant. A network slice is characterized by (*i*) its traffic type (elastic or inelastic), and (*ii*) its number of users (*i.e.*, the subscribers of a given service) that have to be served.

A network slice comes with certain guarantees provided by an SLA agreement between the tenant and the InP. In our model, a tenant requests a network slice that comprises a certain number of end-users and a traffic type. Then, as long as the number of users belonging to a network slice is less or equal than the one included in the SLA agreement, each of them will be provided with the service guarantees corresponding to their traffic

type.

A network slice may be limited to a certain geographical area, in which case the corresponding guarantees only apply to the users residing in the region. In our model, we focus on the general case and consider network slices that span over the entire network. However, the model could be easily extended to consider restricted geographical areas.

Following state of the art approaches [28], network slicing onboarding is an automated process that involves little or no human interaction between the InP. Based on these approaches, we consider a bidding system in order to dynamically allocate network slices to tenants. With this, tenants submit requests for network slices (*i.e.*, a certain number of users of a given service) to the InP, which accepts or rejects the request according to an admission control algorithm such as the one proposed in this section. To that aim, we characterize slices request by:

- Network slice duration t: this is the length of the time interval for which the network slice is requested.
- Traffic type κ: according to the traffic model above, the traffic type of a slice can either be elastic or inelastic traffic.
- Network slice size N: the size of the network slice is given by the number of users it should be able to accommodate.
- Price ρ: the cost a tenant has to pay for acquiring resources for a network slice. The price is per time unit, and hence the total revenue obtained by accepting a network slice is given by $r = \rho t$.

Following the above characterization, an InP will have catalogs of network slice blueprinted by predefined values for the tuple $\{\kappa, N, \rho\}$, which we refer to as network slice classes. Tenants issue requests for one of the slice classes available in the catalogue, indicating the total duration t of the network slice. When receiving a request, an InP has two possible decisions: it can reject the network slice and the associate revenue to keep the resources free or it can accept the network slice and charge the tenant r dollars. If accepted, the InP grants resources to a tenant during a t-window.

To compute the profit received by the tenant, we count the aggregated revenue resulting from all the admitted slices. This reflects the net benefit of the InP as long as (i) the costs of the InP are fixed, or (ii) they are proportional to the network utilization (in the latter case, ρ reflects the difference between the revenue and cost of instantiating a slice). We argue that this covers a wide range of cases of practical interest such as spectrum resources or computational ones. Moreover, in the cases where costs are not linear with the network usage, our analysis and algorithm could be extended to deal with such cases by subtracting the cost at a given state from the revenue.

3.2. Admissibility region

An online admission control algorithm has to decide whether to accept or reject a new incoming network slice request issued by a tenant. Such a decision is driven by a number of variables such as the expected income and the resources available. The objective of an admission control algorithm is to maximize the overall profit while guaranteeing the SLA committed to all tenants. A fundamental component of such an algorithm is the *admissibility region*, *i.e.*, the maximum number of network slices that can be admitted in the system while guaranteeing that the SLAs are met for all tenants. Indeed, if admitting a new network slice in the system would lead to violating the SLA of already admitted slices, then such a request should be rejected.

In the following, we provide an analysis to determine the admissibility region, denoted by $\mathcal{A}$, as a first step towards the design of the optimal admission algorithm.

3.2.1. Theoretical analysis

We say that a given combination of inelastic users of the various classes and elastic users belongs to the admissibility region, *i.e.*, $\{N_1, \ldots, N_{|\mathcal{I}|}, N_e\} \in \mathcal{A}$, when the guarantees for elastic and inelastic traffic are satisfied for this combination of users. In the following, we compute the admissibility region $\mathcal{A}$.

In order to determine whether a given combination of users of different types, $\{N_1, \ldots, N_{|\mathcal{I}|}, N_e\}$, belongs to $\mathcal{A}$, we proceed as follows. We first compute the outage probability for an inelastic user of class $i \in \mathcal{I}$, $P_{out,i}$. Let R_b be the throughput consumed by the inelastic users at b. The average value of R_b can be computed as

$$\mathbb{E}[R_b] = \sum_{j \in \mathcal{I}} N_j P_{j,b} R_j, \tag{3.1}$$

and the typical deviation as

$$\sigma_b^2 = \sum_{j \in \mathcal{I}} N_j \sigma_{j,b}^2, \tag{3.2}$$

where $\sigma_{j,b}^2$ is the variance of the throughput consumed by one inelastic user of class j, which is given by

$$\begin{aligned} \sigma_{j,b}^2 &= P_{j,b}(R_j - P_{j,b}R_j)^2 + (1 - P_{j,b})(P_{j,b}R_j)^2 \\ &= P_{j,b}(1 - P_{j,b})R_j^2. \end{aligned} \tag{3.3}$$

Our key assumption is to approximate the distribution of the committed throughput at base station b by a normal distribution of mean R_b and variance σ_b^2, i.e., $\mathcal{N}(\mathbb{E}[R_b], \sigma_b^2)$. Note that, according to [33], this approximation is appropriate as long as the number of users per base station in the boundary of the admissibility region is no lower than 5,

which is generally satisfied by cellular networks (even in the extreme case of small cells).

The outage probability at base station b is given by the probability that the committed throughput exceeds the base station capacity, *i.e.*,

$$P_{out,b} = \mathbb{P}(R_b > C_b), \tag{3.4}$$

where C_b be the capacity of base station b.

To compute the above probability with the normal approximation, we proceed as follows:

$$P_{out,b} \approx 1 - \Phi\left(\frac{C_b + C_b - \mathbb{E}[R_{b,i}]}{\sigma_{b,i}}\right), \tag{3.5}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution and C_b is a continuity correction factor that accounts for the fact R_b is not a continuous variable. In line with [34], where this is applied to a binomial distribution and the correction factor is one half of the step size, in our case we set the continuity correction factor as one half of the average step size, which yields

$$C_b = \frac{1}{2}\frac{\sum_{j\in\mathcal{I}} P_{j,b} N_j R_j}{\sum_{j\in\mathcal{I}} P_{j,b} N_j}. \tag{3.6}$$

Once we have obtained $P_{out,b}$, we compute the outage probability of an inelastic user of class i with the following expression:

$$P_{out,i} = \sum_{b\in\mathcal{B}} P_{i,b} P_{out,b}. \tag{3.7}$$

Next, we compute the average throughput of an elastic user. To this end, we assume that (i) in line with [32], elastic users consume all the capacity left over by inelastic traffic, (ii) there is always at least one elastic user in each base station, and (iii) all elastic users receive the same throughput on average.

With the above assumptions, we proceed as follows. The average committed throughput consumed by inelastic users at base station b is given by

$$\mathbb{E}[R_b] = \sum_{i\in\mathcal{I}} N_i P_{i,b} R_i, \tag{3.8}$$

which gives an average capacity left over by inelastic users equal to $C_b - \mathbb{E}[R_b]$. This capacity is entirely used by elastic users as long as the base station is not empty. The total capacity usage by elastic users is then given by the sum of this term over all base stations. As this capacity is equally shared (on average) among all elastic users, this leads

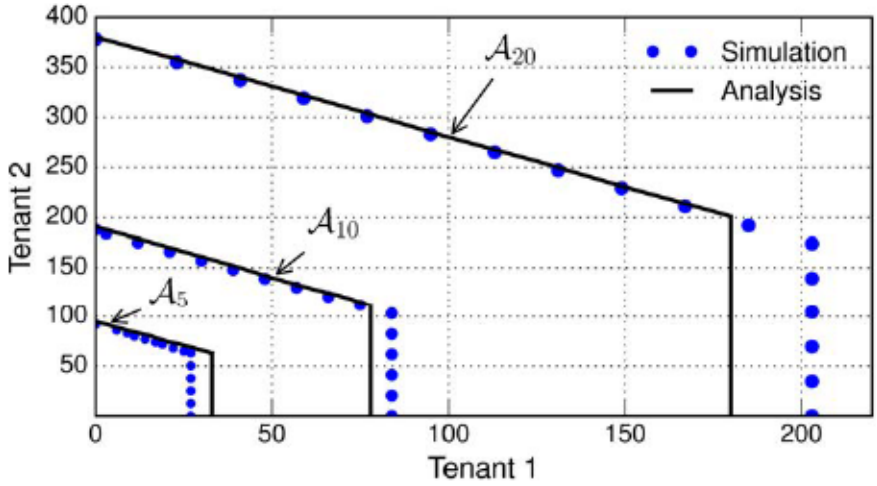


Figure 3.1: Admissibility region: analysis vs. simulation.

to the following expression for the average throughput of an elastic user:

$$r_e = \frac{\sum_{b \in \mathcal{B}} C_b - \mathbb{E}[R_b]}{N_e}. \tag{3.9}$$

Based on the above, we compute the admissibility region $\mathcal{A}$ as follows. For a given number of inelastic users in each class, N_i, $i \in \mathcal{I}$, and of elastic users, N_e, we compute the outage probability of the inelastic classes, $P_{out,i}$, and the average throughput of the elastic users, r_e. If the resulting values meet the requirements for all classes, i.e., $P_{out,i} \leq P_{out}$ $\forall i$ and $r_e \geq R_e$, then this point belongs to the admissibility region, and otherwise it does not.

3.2.2. Validation of the admissibility region

In order to assess the accuracy of the above analysis, we compare the admissibility region obtained theoretically against the one resulting from simulations. To this end, we consider the reference scenario recommended by ITU-T [35], which consists of $|\mathcal{B}| = 19$ base stations placed at a fixed distance of 200m. Following the system model of Section 3.1, we have elastic and inelastic users. All inelastic users belong to the same class, and all users (elastic and inelastic) move in the area covered by these base stations following the Random Waypoint (RWP) mobility model, with a speed uniformly distributed between 2 and 3 m/s.

The association procedure of elastic and inelastic users with base stations is as follows. Inelastic users try to associate to the nearest base station $b \in \mathcal{B}$, if it has at least R_i capacity left. Otherwise they do not associate and generate an outage event, joining again the network when their throughput guarantee can be satisfied. When associating, they consume a capacity R_i from the base station. The probability of association to each

base station (i.e., the $P_{i,b}$ values) are extracted from the simulations and fed into the analysis.

Similarly to inelastic users, elastic users always associate to the nearest base station. All the elastic users associated to a base station fairly share among them the capacity left over by inelastic users. Upon any association event, the throughput received by the users associated to the new and the old base station changes accordingly.

Following the above procedure, we have simulated all the possible combinations of inelastic and elastic users, $\{N_i, N_e\}$. For each combination, we have evaluated the average throughput received by elastic users, computed over samples of 10 seconds time windows, and the outage probability P_{out} of inelastic users, computed as the fraction of time over which they do not enjoy their guaranteed throughput. If these two metrics (average elastic traffic throughput and inelastic traffic outage probability) are within the guarantees defined for the two traffic types, we place this combination inside the admissibility region, and otherwise we place it outside.

Fig. 3.1 shows the boundaries of the admissibility region obtained analytically and via simulation, respectively, for different throughput guarantees for elastic and inelastic users ($\mathcal{A}_5 : R_i = R_e = C_b/5$, $\mathcal{A}_{10} : R_i = R_e = C_b/10$ and $\mathcal{A}_{20} : R_i = R_e = C_b/20$) and $P_{out} = 0.01$. We observe that simulation results follow the analytical ones fairly closely. While in some cases the analysis is slightly conservative in the admission of inelastic users, this serves to ensure that inelastic users' requirements in terms of outage probability are always met.

3.3. Optimising 5G Infrastructure Markets

While the admissibility region computed above provides the maximum number of elastic and inelastic users that can be admitted, an optimal admission algorithm that aims at maximizing the revenue of the InP may not always admit all the requests that fall within the admissibility region. Indeed, when the network is close to congestion, admitting a request that provides a low revenue may prevent the infrastructure provider from admitting a future request with a higher revenue associated. Therefore, the infrastructure provider may be better off by rejecting the first request with the hope that a more profitable one will arrive in the future.

The above leads to the need for devising an admission control strategy for incoming slice requests. Note that the focus is on the admission of slices, in contrast to traditional algorithms focusing on the admission of users; once a tenant gets its slice admitted and instantiated, it can implement whatever algorithm it considers more appropriate to admit users into the slice.

In the following, we model the decision-making process on slice requests as a Semi-Markov Decision Process (SMDP). The proposed model includes the definition of the

state space of the system, along with the decisions that can be taken at each state and the resulting revenues. This is used as follows: (i) to derive the optimal admission control policy that maximizes the revenue of the infrastructure provider, which serves as a benchmark for the performance evaluation, and (ii) to lay the basis of the machine learning algorithms proposed in Section 3.4, which implicitly rely on the states and decision space of the SMDP model.

3.3.1. Markovian decision-making process analysis

SMDP is a widely used tool to model sequential decision-making problems in stochastic systems such as the one considered in this paper, in which an agent (in our case the InP) has to take decisions (in our case, whether to accept or reject a network slice request) with the goal of maximizing the reward or minimizing the penalty. For simplicity, we first model our system for the case in which there are only two classes of slice requests of fixed size $N = 1$, i.e., for one elastic user or for one inelastic user. Later on, we will show how the model can be extended to include an arbitrary set of network slice requests of different sizes.

The Markov Decision Process theory [36] models a system as: (i) a set of states $s \in S$, (ii) a set of actions $a \in A$, (iii) a transition function $P(s, a, s')$, (iv) a time transition function $T(s, a)$, and (v) a reward function $R(s, a)$. The system is driven by events, which correspond to the arrival of a request for an elastic or an inelastic slice as well as the departure of a slice (without loss of generality, we assume that arrivals and departures never happen simultaneously, and treat each of them as a different event). At each event, the system can be influenced by taking one of the possible actions $a \in A$. According to the chosen actions, the system earns the associated reward function $R(s, a)$, the next state is decided by $P(s, a, s')$ while the transition time is defined by $T(s, a)$.

The inelastic and elastic network slices requests follow two Poisson processes $\mathcal{P}_i$ and $\mathcal{P}_e$ with associated rates of λ_i and λ_e, respectively. When admitted into the system, the slices occupy the system resources during an exponentially distributed time of average $\frac{1}{\mu_i}$ and $\frac{1}{\mu_e}$. Additionally, they generate a revenue per time unit for the infrastructure provider of ρ_i and ρ_e. That is, the total revenue r generated by, e.g., an elastic request with duration t is $t\rho_e$.

We define our space state S as follows. A state $s \in S$ is a three-sized tuple $(n_i, n_e, k \mid n_i, n_e \in \mathcal{A})$ where n_i and n_e are the number of inelastic and elastic slices in the system at a given decision time t, and $k \in \{i, e, d\}$ is the next event that triggers a decision process. This can be either a new arrival of a network slice request for inelastic and elastic slices ($k = i$ and $k = e$, respectively), or a departure of a network slice of any kind that left the system ($k = d$). In the latter case, n_i and n_e represent the number of inelastic and elastic slices in the system after the departure. Fig. 3.2 shows how the space state S relates to the admissibility region $\mathcal{A}$.

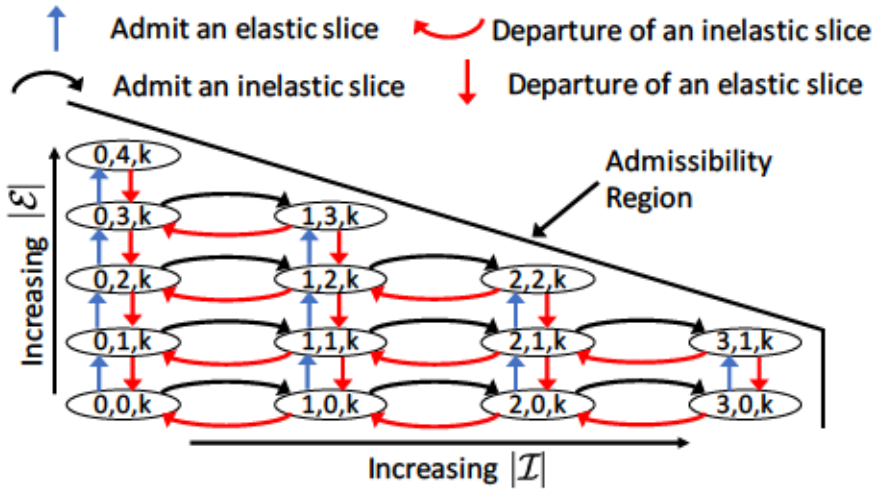


Figure 3.2: Example of system model with the different states.

The possible actions $a \in A$ are the following: $A = G, D$. The action G corresponds to admitting the new request of an elastic or inelastic slice; in this case, the resources associated with the request are granted to the tenant and the revenue $r = \rho_{i,e}t$ is immediately earned by the infrastructure provider. In contrast, action D corresponds to rejecting the new request; in this case, there is no immediate reward but the resources remain free for future requests. Note that upon a departure ($k = d$), the system is forced to a fictitious action D that involves no revenue. Furthermore, we force that upon reaching a state in the boundary of the admissibility region computed in the previous section, the only available action is to reject an incoming request ($a = D$) as otherwise we would not be meeting the committed guarantees. Requests that are rejected are lost forever.

The transition rates between the states identified above are derived next. Transitions to a new state with $k = i$ and $k = e$ happen with a rate λ_i and λ_e, respectively. Additionally, states with $k = d$ are reached with a rate $n_i\mu_i + n_e\mu_e$ depending the number of slices already in the system. Thus, the average time the system stays at state s, $T(s, a)$ is given by

$$T(s, a) = \frac{1}{\upsilon(n_i, n_e)}, \tag{3.10}$$

where n_i, and n_e are the number of inelastic and elastic slices in state s and $\upsilon(n_i, n_e) = \lambda_i + \lambda_e + n_i\mu_i + n_e\mu_e$.

We define a policy $\pi(S)$, $\pi(s) \in A$, as a mapping from each state s to an action A. Thus, the policy determines whether, for a given number of elastic and inelastic slices in the system, we should admit a new request of an elastic or an inelastic slice upon each arrival. With the above analysis, given such a policy, we can compute the probability of staying at each of the possible states. Then, the long-term average revenue R obtained

by the infrastructure provider can be computed as

$$R = \sum_{n_i, n_e, k} P\left(n_i, n_e, k\right)\left(n_i \rho_i + n_e \rho_e\right), \tag{3.11}$$

where ρ_i and ρ_e are the price per time unit paid by an inelastic and an elastic network slice, respectively.

The ultimate goal is to find the policy $\pi\left(S\right)$ that maximizes the long term average revenue, given the admissibility region and the network slices requests arrival process. We next devise the Optimal Policy when the parameters of the arrival process are known *a priori*, which provides a benchmark for the best possible performance. Later on, we design in Section 3.4 an adaptive online algorithm that aims at maximizing revenue by learning from the outcome resulting form the previous decisions, and a deep reinforcement learning algorithm that further extends the adaptive algorithm in terms of convergence speed and memory requirements.

3.3.2. Optimal policy

In order to derive the optimal policy, we build on Value Iteration [37], which is an iterative approach to find the optimal policy that maximizes the average revenue of an SMDP-based system. According to the model provided in the previous section, our system has the transition probabilities $P\left(s, a, s'\right)$ detailed below.

Let us start with $a = D$, which corresponds to the action where an incoming request is rejected. In this case, we have that when there is an arrival, which happens with a rate λ_i and λ_e for inelastic and elastic requests, respectively, the request is rejected and the system remains in the same state. In case of a departure of an elastic or an inelastic slice, which happens with a rate of $n_e\mu_e$ or $n_i\mu_i$, the number of slices in the system is reduced by one unit (recall that no decision is needed when slices leave the system). Formally, for $a = D$ and $s = (n_i, n_e, i)$, we have:

$$P\left(s, a, s'\right) = \begin{cases} \frac{\lambda_i}{v(n_i, n_e)}, & s' = (n_i, n_e, i) \\ \frac{\lambda_e}{v(n_i, n_e)}, & s' = (n_i, n_e, e) \\ \frac{n_i\mu_i}{v(n_i, n_e)}, & s' = (n_i - 1, n_e, d) \\ \frac{n_e\mu_e}{v(n_i, n_e)}, & s' = (n_i, n_e - 1, d) \end{cases} \tag{3.12}$$

When the chosen action is to accept the request ($a = G$) and the last arrival was an inelastic slice ($k = i$), the transition probabilities are as follows. In case of an inelastic slice arrival, which happens with a rate λ_i, the last arrival remains $k = i$, and in case of an elastic arrival it becomes $k = e$. The number of inelastic slices increases by one unit in all cases except of an inelastic departure (rate $n_i\mu_i$). In case of an elastic departure (rate $n_e\mu_e$), the number of elastic slices decreases by one. Formally, for $a = G$ and

$s = (n_i, n_e, i)$, we have:

$$P(s, a, s') = \begin{cases} \frac{\lambda_i}{v(n_i+1, n_e)}, & s' = (n_i + 1, n_e, i) \\ \frac{\lambda_e}{v(n_i+1, n_e)}, & s' = (n_i + 1, n_e, e) \\ \frac{(n_i+1)\mu_i}{v(n_i+1, n_e)}, & s' = (n_i, n_e, d) \\ \frac{n_e\mu_e}{v(n_i+1, n_e)}, & s' = (n_i + 1, n_e - 1, d) \end{cases} \tag{3.13}$$

If the accepted slice is elastic ($k = e$), the system exhibits a similar behavior to the one described above but increasing by one the number of elastic slices instead. Thus, for $a = G$, $s = (n_i, n_e, e)$, we have:

$$P(s, a, s') = \begin{cases} \frac{\lambda_i}{v(n_i, n_e+1)}, & s' = (n_i, n_e + 1, i) \\ \frac{\lambda_e}{v(n_i, n_e+1)}, & s' = (n_i, n_e + 1, e) \\ \frac{n_i\mu_i}{v(n_i, n_e+1)}, & s' = (n_i - 1, n_e + 1, d) \\ \frac{(n_e+1)\mu_e}{v(n_i, n_e+1)}, & s' = (n_i, n_e, d) \end{cases} \tag{3.14}$$

A reward is obtained every time the system accepts a new slice, which leads to

$$R(s, a) = \begin{cases} 0, & a = D \\ t\rho_i, & a = G, k = i \\ t\rho_e, & a = G, k = e \end{cases} \tag{3.15}$$

Applying the Value Iteration algorithm [37] for SMDP is not straightforward. The standard algorithm cannot be applied to a continuous time problem as it does not consider variable transition times between states. Therefore, in order to apply Value Iteration to our system, an additional step is needed: all the transition times need to be normalized to multiples of a faster, arbitrary, fixed transition time τ [38]. The only constraint that has to be satisfied by τ is that it has to be faster than any other transition time in the system, which leads to

$$\tau < \min T(s, a), \qquad \forall s \in S, \forall a \in A. \tag{3.16}$$

With the above normalization, the continuous time SMDP corresponding to the analysis of the previous section becomes a discrete time Markov Process and a modified Value Iteration algorithm may be used to devise the best policy $\pi(S)$ (see Algorithm 1). The discretized Markov Chain will hence perform one transition every τ interval. Some of these transitions correspond to transitions in continuous time system, while in the others the system keeps in the same state (we call the latter fictitious transitions).

The normalization procedure affects the update rule of step 2 in Algorithm 1. All the transition probabilities $P(s, a, s')$ are scaled by a factor $\frac{\tau}{T(s,a')}$ to enforce that the system

Algorithm 1 Value Iteration

Initialization:

$V(s) \leftarrow 0, \forall s \in S$ ▷ Initialize the long term expected revenue.

$n \leftarrow 1$ ▷ Initialize the step number.

$M_n \leftarrow 0$

$m_n \leftarrow 0$

1: **while** $(M_n - m_n) < 0 \vee (M_n - m_n) > \epsilon m_n$ **do**

2: Update the expected reward at time $n+1$, $V_{n+1}(s)$ using the rule

$$V_{n+1}(s) = \max_{a \in A} \left[\frac{R(s,a)}{T(s,a)} \tau + \frac{\tau}{T(s,a)} \sum_{s'} P(s,a,s') V_n(s') + \left(1 - \frac{\tau}{T(s,a)}\right) V_n(s) \right] \quad \forall s \in S$$

3: Compute the boundaries

$$M_n = \max_{s \in S} (V_{n+1}(s) - V_n(s))$$

$$m_n = \min_{s \in S} (V_{n+1}(s) - V_n(s))$$

4: **end while**

Output: $V(s), \forall s \in S$

stays in the corresponding state during an average time $T(s,a')$. Also, the revenue $R(s,a)$ is scaled by a factor of $T(s,a)$ to take into account the fact that the reward $R(s,a)$ corresponds to a period $T(s,a)$ in the continuous system, while we only remain in a state for a τ duration in the discrete system. In some cases, transitions in the sampled discrete time system may not correspond to any transition in the continuous time one: this is taken into account in the last term of the equation, i.e., in case of a fictitious transition, we keep in state $V_n(s)$.

As proven in the next section, Algorithm 1 is guaranteed to find the optimal policy $\pi(S)$. Such an optimal policy is illustrated in Fig. 3.3 for the case where the price of inelastic slice is higher than that of elastic slice ($\rho_i > \rho_e$). The figure shows those states for which the corresponding action is to admit the new request (straight line), and those for which it is to reject it (dashed lines). It can be observed that while some of the states with a certain number of elastic slices fall into the admissibility region, the system is better off rejecting those requests and waiting for future (more rewarding) requests of inelastic slice. In contrast, inelastic slice requests are always admitted (within the admissibility region).

The analysis performed so far has been limited to network slice requests of size one. In order to extend the analysis to requests of an arbitrary size, we proceed as follows. We set the space state to account for the number of slices of each different class in the system (where each class corresponds to a traffic type and a given size). Similarly, we compute the transition probabilities $P(s,a,s')$ corresponding to arrival and departures of different

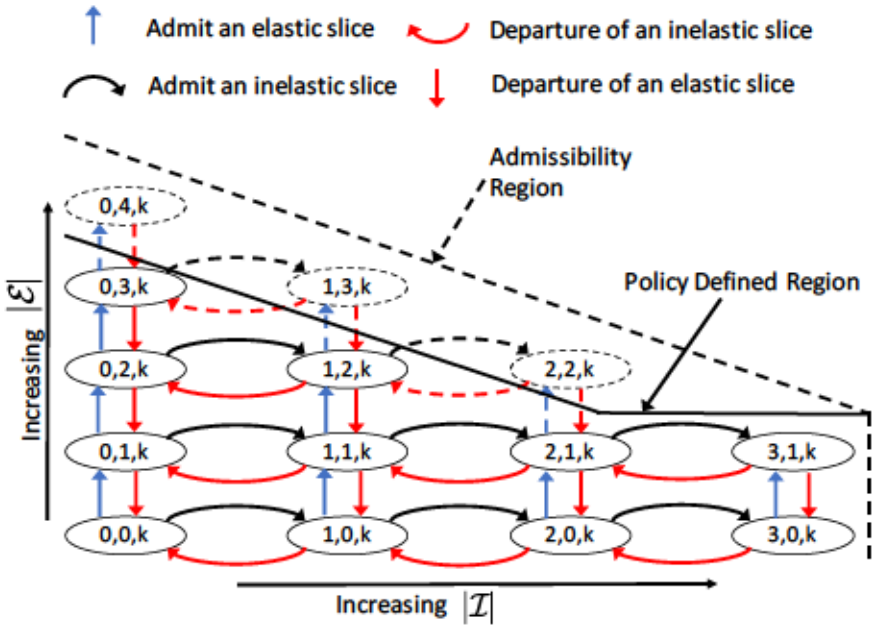


Figure 3.3: Example of optimal policy for elastic and inelastic slices.

classes. With this, we can simply apply the same procedure as above (over the extended space state) to obtain the optimal policy.

3.3.3. Optimality and convergence analysis

In the following, we provide some insights on the optimality and convergence of Algorithm 1, showing that: (i) the algorithm converges to a certain policy, and (ii) the policy to which the algorithm converges performs arbitrarily close to the optimal policy. Theorem 6.6.1 in [39] proves that the policy $\pi(S)$ obtained using Algorithm 1 provides a long-run average reward $g_s(\pi(S))$ that is arbitrarily bounded by an ϵ value when compared to the optimal one g^*. Thus,

$$0 \leq \frac{g^* - g_s(\pi(S))}{g_s(\pi(S))} \leq \frac{M_n - m_n}{m_n} \leq \epsilon, \qquad \forall s \in S$$

The convergence of Algorithm 1 is guaranteed by the third term of the inequality above, that acts as a decreasing envelope of the second term, as shown by Theorem 6.6.3 in [39]:

$$m_{n+1} \geq m_n, \quad M_{n+1} \leq M_n, \qquad \forall n \geq 1.$$

By applying step 3 of Algorithm 1, the obtained $\pi(S)$ is ϵ-bounded to the optimal. While the aforementioned Theorems in [39] solve a cost minimisation problem, we adapted them to our revenue maximisation scenario. In our experiments, we set $\epsilon = 0.001$.

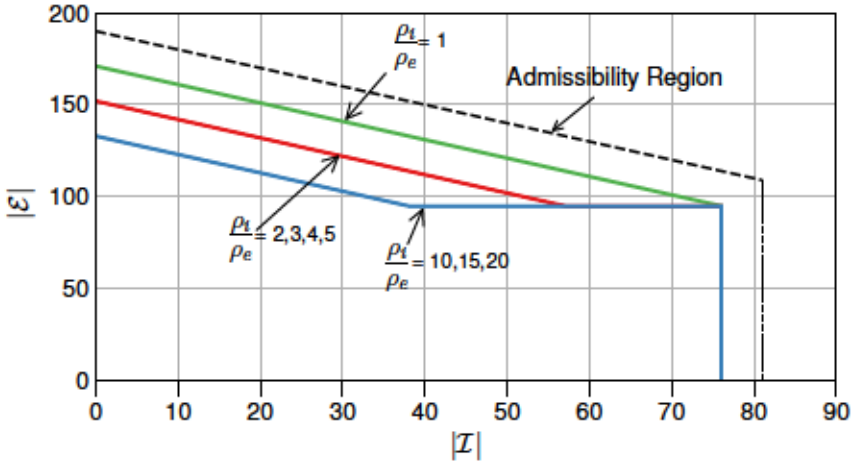


Figure 3.4: Optimal admission policy for elastic traffic.

3.3.4. Optimal admission policy assessment

In order to analyse the admission policy resulting from the optimal algorithm designed in Section 3.3.2, we consider a scenario with four slice classes, two for elastic traffic and two for inelastic. We set $\mu = 5$ for all network slices classes, and the arrival rates equal to $\lambda_i = 2\mu$ and $\lambda_e = 10\lambda_i$ for the elastic and inelastic classes, respectively. Two network slice sizes are considered, equal to $C/10$ and $C/20$, where C is the total network capacity. Similarly, we set the throughput required guarantees for elastic and inelastic traffic to $R_i = R_e = C_b/10$. We analyze the optimal admission policy for different ratios between ρ_i and ρ_e, the average revenue per time unit generated by inelastic and elastic slices, respectively. Note that, given that inelastic traffic is more demanding, it is reasonable to assume that it pays a higher price than elastic traffic, *i.e.*, $\rho_i \geq \rho_e$. As inelastic traffic provides a higher revenue, in order to maximise the total revenue, the InP will always admit inelastic network slice requests. In contrast, it is to be expected that, while elastic traffic requests will be admitted when the utilisation is low, they may be rejected with higher utilisation in order to avoid losing the opportunity to admit future (and more rewarding) inelastic requests. Furthermore, it is to be expected that this behaviour will be exacerbated as the ρ_i/ρ_e grows larger.

The optimal admission policy for elastic traffic resulting from our algorithm is shown in Fig. 3.4. As expected, we can observe that the region corresponding to the admission of elastic network slices requests is smaller than the admissibility region, implying that we are more restrictive in the admission of elastic traffic. Furthermore, and also as expected, this region becomes smaller for larger ρ_i/ρ_e ratios. These results thus confirm our intuitions on the optimal admission policy.

3.4. ML approach for 5G Market optimization

The Value Iteration algorithm described in Section 3.3.2 provides the optimal policy for revenue maximisation under the framework described of Section 3.3.1. While this is very useful in order to obtain a benchmark for comparison, the algorithm itself has a very high computational cost, which makes it impractical for real scenarios. Indeed, as the algorithm has to update all the V values $V(s)$, $s \in S$ at each step, the running time grows steeply with the size of the state space, and may become too high for large scenarios.

Building on the analysis of the previous section, in the following we design an adaptive algorithm based on reinforcement learning that aims at maximising revenue by learning from the outcome resulting from the previous decisions. In contrast the optimal policy of the previous section, this algorithm is executed online while taking admission control decisions, and hence does not require of high computational resources.

3.4.1. Q-Learning model

Our adaptive algorithm is based on the Q-Learning framework [40]. Before describing the algorithm itself, we describe how we model the algorithm under the Q-Learning framework.

Q-Learning is a machine learning framework for designing adaptive algorithms in SMDP-based systems such as the one analysed in Section 3.3.1. It works taking decisions that move the system to different states within the SMDP state-space and observing the outcome. Thus, it leverages the "exploration vs. exploitation" principle: the algorithm learns by visiting unvisited states and takes the optimal decision when dealing with already visited ones.

Q-Learning provides two key advantages as compared to Value Iteration framework described in the previous section:

- The resulting algorithm is *model-free*. Indeed, it makes no assumptions on the underlying stochastic processes, but rather learns by observing the events that take place in the system.

- It is an *online* algorithm that constantly learns the characteristics of the system by exploring it and taking decisions.

Our Q-Learning framework builds on the SMDP-based system model of Section 3.3.1. The Q-Learning space state is similar to the one of the SMDP model:

$$(n_i^\star, n_e^\star, k \mid o(n_i^\star, n_e^\star) \in \mathcal{A})$$

where $n_i^\star$ and $n_e^\star$ are defined as a n-dimension tuples $(n_1, n_2, \ldots, n_c)$ describing the number

of slices of different sizes in the system for inelastic and elastic traffic types. Analogously, o is the occupation of the system, and $k \in \{i^\star, e^\star\}$ where $i^\star$ and $e^\star$ are the sets of events associated to an arrival of an inelastic or elastic slice request of a given size.

With Q-Learning, we do not need to include departures in the space state, since no decision is taken upon departures. Similarly, we do not need to include the states in the boundary of the admissibility region; indeed, in such states we do not have any option other than rejecting any incoming request, and hence no decisions need to be taken in these states either. Furthermore, the system is not sampled anymore, as all transitions are triggered by an arrival event and the subsequent decision $a \in A$.

The key idea behind the Q-Learning framework is as follows. We let $Q(s,a)$ denote the expected reward resulting from taking an action a at a certain state s. The system keeps memory for each state of $Q(s,a)$. It starts with empty $Q_0(s,a)$ and at the decision step n it takes an action a based on the past estimations of $Q(s,a)$. Hence, the system experiences a transition from state s at the decision step n, to state s' at decision step $n+1$. Then, once in step $n+1$, the algorithm has observed both the reward obtained during the transition $R(s,a)$ and a sample t_n of the transition time. Then, the algorithm updates the $Q(s,a)$ involved in the decision process at step n using the newly gathered reward and transition time information. After a learning phase, the optimal admission policy at a certain state will be the one that maximises the resulting expected revenue, i.e.,

$$V(s) = \max_{a \in A} Q(s,a)$$

3.4.2. Algorithm description

Building on the above model, we describe our Q-Learning algorithm in the following. The algorithm maintains the Q-values which are updated iteratively following a sample-based approach as described in Algorithm 2, in which new events are evaluated at the time when they happen. In addition to the procedure to update the Q-values described in Algorithm 2, the Q-Learning algorithm also relies on two other procedures: the *TD-learning* and *exploration - exploitation* procedures.

TD-learning ensures the convergence of the algorithm by employing the α parameter, which is the learning rate. The requirements for setting α are two [41]: (i) $\sum_{n=0}^{\infty} \alpha_n = \infty$ and (ii) $\sum_{n=0}^{\infty} \alpha_n^2 < \infty$. The Q-values update process in step 3 of Algorithm 2 needs to build a correct estimation of the expected revenue obtained by choosing an action a while in state s. On the one hand, new samples ω (with more updated information) should be weighted by a larger weight than the estimation built on all the past samples $Q(s,a)$, especially if the first exploration steps did not provide a good result. On the other hand, α_n coefficients have to decrease with time, in order to eventually converge to a fixed set of $Q(s,a)$ values. When setting α according to these requirements, we make the following

Algorithm 2 Q-Learning update procedure

An event is characterized by:
s, a, s', r, t [starting state, action taken, landing state, obtained reward, transition time].

Initialization:
$Q(s,a) \leftarrow 0, \forall s \in S,\ a \in A.$

1: **procedure** UPDATE(α,ω) ▷ Update the old $Q(s,a)$
2: Evaluate the new sample observation as follows:

$$\omega = R(s,a,s') - \sigma t_n + \max_{a'} Q(s',a')$$

▷ t_n is the transition time between two subsequent states s and s' after action a

3: Integrate the new sample in a running exponential average estimation of $Q(s,a)$:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\omega$$

4: **end procedure**

additional considerations: too slowly descending α sequences will delay the convergence of the algorithm, but too fast ones may make the algorithm unaware of new choices too soon. Based on all these requirements and considerations, we set $\alpha = \frac{0.5}{\eta(s,a)}$, where $\eta(s,a)$ is the number of times the action a was selected, being in state s.

Exploration - exploitation drives the selection of the best action to be taken at each time step. While choosing the action a that maximises the revenue at each step contributes to maximising the overall revenue (i.e., *exploitation* step), we also need to visit new (still unknown) states even if this may lead to a suboptimal revenue (i.e., *exploration* step). The reason for this is that the algorithm needs to explore all possible (s,a) options in order to evaluate the impact of the different decisions. The trade-off between exploitation and exploration is regulated by the γ parameter; in this paper we take $\gamma = 0.1$ in order to force that sometimes the wrong decision is taken and thus we learn all possible options, which ultimately improves the accuracy of the algorithm. The probability of taking wrong choices decreases as the α_n values become smaller, up to the point where no wrong decisions are taken any more, once the algorithm already visited all state s a number of times sufficiently large to learn the best Q-value.

3.4.3. Adaptive algorithm performance

We next evaluate the performance of our adaptive algorithm by comparing it in the scenario described in Section 3.3.4 against: (i) the benchmark provided by the optimal algorithm, and (ii) two naive policies that always admit elastic traffic requests and always reject them, respectively. Fig. 3.5 shows the relative average reward obtained by each of this policies, taking as baseline the policy that always admit all network slice requests (as this would be the most straightforward algorithm).

We observe from the figure that our adaptive algorithm performs very closely to the

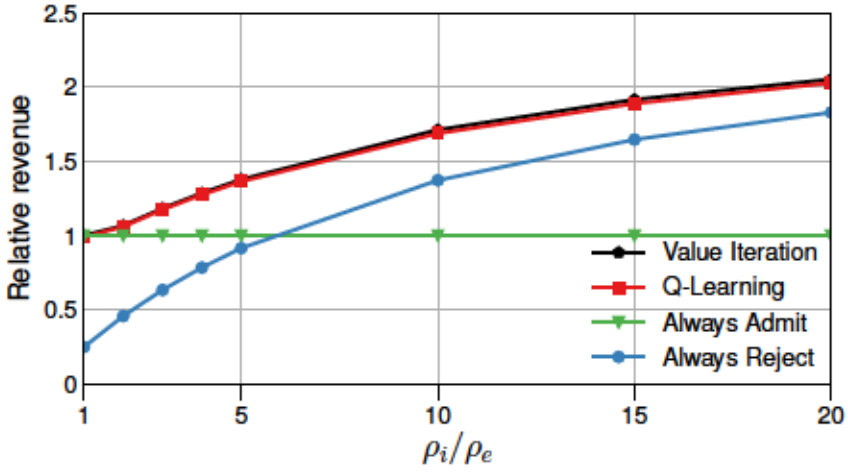


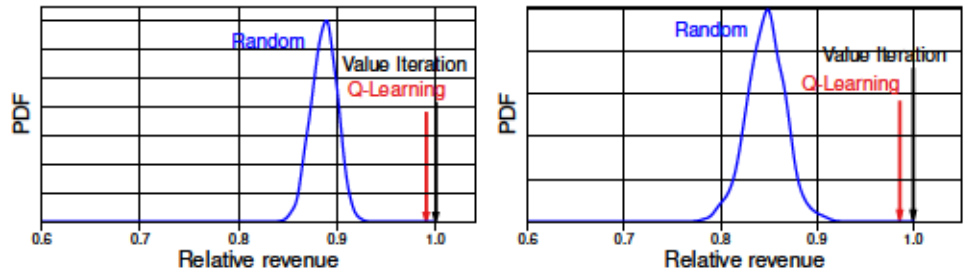
Figure 3.5: Revenue vs. ρ_i/ρ_e.

optimal policy, which serves to validate the algorithm design proposed in this paper. We further observe that the revenue improvements over the naive policies is very substantial, up to 100% in some cases. As expected, for small ρ_i/ρ_e the policy that always admits all requests is optimal, as in this case both elastic and inelastic slices provide the same revenue; in contrast, for very large ρ_i/ρ_e ratios the performance of the "always reject" policy improves, as in this case the revenue obtained from elastic traffic is (comparatively) much smaller.

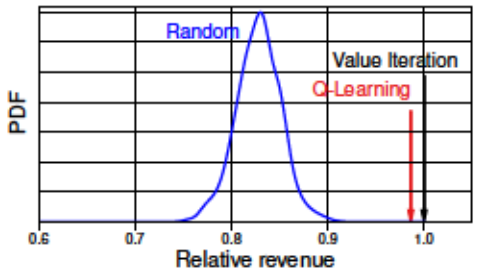
While this result shows that the proposed algorithm performs close to optimal, it is only compared against two naive policies and thus does not give an insight on the revenue gains that could be achieved over smarter yet not optimal policies. To this end, we compare the performance of our adaptive algorithm against a set of "smart" random policies defined as: inelastic network slices requests are always accepted ($k = i \Rightarrow a = G$), while the decision of rejecting an elastic request ($k = e \Rightarrow a = D$) is set randomly. Then, by drawing a high number of random policies, it is to be expected that some of them provide good performance.

Fig. 3.6 shows the comparison against 1000 different random policies. The results confirm that (i) none of the random policies outperforms our approach, further confirming the optimality of the approach, and (ii) substantial gains (around 20%) are obtained over the random policies. This result confirms that a smart heuristic is not effective in optimizing revenue, and very substantial gains can be achieved by using a close to optimal policy such as our adaptive algorithm.

The previous results have assumed that (i) arrivals and departures follow Poisson process with exponential times, and (ii) the optimal algorithm has a perfect estimation of the statistics of this process. In this section we address a more realistic case in which neither of these assumption holds. We hence introduce two modifications: (i) arrivals and



(a) $\rho_t/\rho_e = 5$ (b) $\rho_t/\rho_e = 10$



(c) $\rho_t/\rho_e = 15$

Figure 3.6: The distribution of the revenues obtained by random smart policies compared to the proposed algorithms.

departures are Pareto-distributed, and (ii) we let the real arrival process $\ddot{\aleph}$ deviate from the estimated one λ: $\ddot{\aleph}(j) = \frac{\lambda}{j+1}$ as a function of a parameter $j > -1$. That is, the optimal policy obtained by Value Iteration under the original assumptions is computed offline, with the estimated parameter, and applied to the real system. Note that for negative j values, the system receives a number of request per time unit higher than the estimated λ, while positive j values indicate a lower requests arrival rate.The results, depicted in Fig. 3.7, show that our adaptive algorithm, which automatically learns the network slice behaviour on the fly and hence is not affected by possible estimation errors, substantially outperforms the optimal policy built upon flawed assumptions and estimations.

3.4.4. N3AC: a Deep Reinforcement Learning approach

The Q-Learning algorithm described in Section 3.4.2 represents an adaptive algorithm for practical usage that achieves close to optimal performance. While it represents a first upgrade compared with the Value Iteration algorithm described in Section 3.3.2, it needs to store and update the expected reward value (*i.e.*, the Q-value) for each state-action pair. As a result, learning the right action for every state becomes infeasible when the space state grows, since this requires experiencing many times the same state-action pair before having a reliable estimation of the Q-value. This leads to extremely long convergence times that are unsuitable for most practical applications. Additionally, storing and

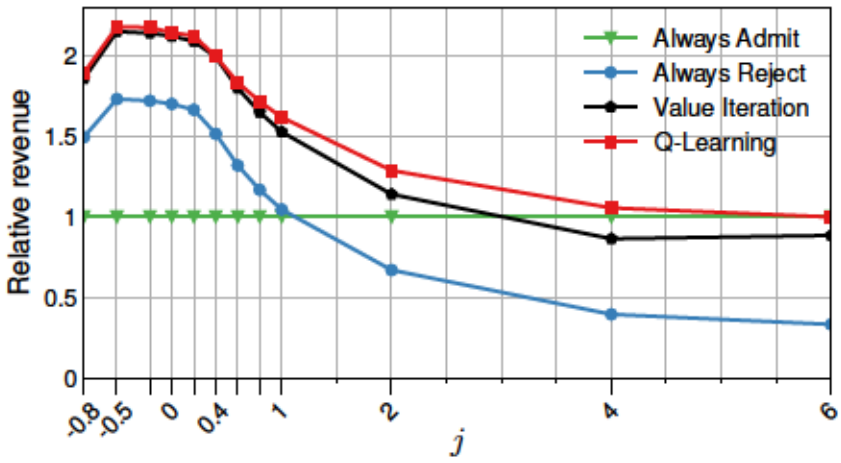


Figure 3.7: Revenue in a perturbed scenario, $\rho_i/\rho_e = 5$.

efficiently visiting the large number of states poses strong requirements on the memory and computational footprint of the algorithm as the state space grows. For the specific case studied in this paper, the number of states in our model increases exponentially with the number of network slicing classes. Hence, when the number of network slicing classes grows, the computational resources required rapidly become excessive.

In the following, we present an alternative approach, the *Network-slicing Neural Network Admission Control (N3AC)* algorithm [8], which has a low computational complexity and can be applied to practical scenarios.

Learning algorithms are in the spotlight since Mnith et al. [42] designed a deep learning algorithm called "deep Q-network" to deal with Atari games, and further improved it in [43] making the algorithm able to learn successful policies directly from high-dimensional sensory inputs and reach human-levels performance in most of Atari games.

The application of Reinforcement and Machine learning approaches to mobile networks is also gaining popularity. Machine learning has been applied to a wide span of applications in 5G networks, ranging from channel estimation/detection for massive MIMO channel to user behavior analysis, location prediction or intrusion/anomaly detection [44].

N3AC falls under category of the Deep Reinforcement Learning (DRL). With N3AC, an agent (the InP) interacts with the environment and takes decisions at a given state, which lead to a certain reward. These rewards are fed back into the agent, which "learns" from the environment and the past decisions using a learning function $\mathcal{F}$. This learning function serves to estimate the expected reward (in our case, the revenue).

Reinforcement Learning (RL) algorithms rely on an underlying Markovian system such as the one described in Section 3.3.2. They provide the following features: (i) high scalability, as they learn online on an event basis while exploring the system and thus

avoid a long learning initial phase, (*ii*) the ability to adapt to the underlying system without requiring any *a priori* knowledge, as they learn by interacting with the system, and (*iii*) the flexibility to accommodate different learning functions $\mathcal{F}$, which provide the mapping from the input state to the expected reward when taking a specific action.

The main distinguishing factor between different kinds of RL algorithms is the structure of the learning function $\mathcal{F}$. Techniques such as Q-Learning [40] employ a lookup table for $\mathcal{F}$, which limits their applicability due to the lack of scalability to a large space state [45]. A common technique to avoid the problems described above for Q-learning is to *generalize* the experience learned from some states by applying this knowledge to other similar states, which involves introducing a different $\mathcal{F}$ function. The key idea behind such *generalization* is to exploit the knowledge obtained from a fraction of the space state to derive the right action for other states with similar *features*. There are different generalization strategies that can be applied to RL algorithms. The most straightforward technique is the linear function approximation [46]. With this technique, each state is given as a linear combination of functions that are representative of the system features. These functions are then updated using standard regression techniques. While this approach is scalable and computationally efficient, the right selection of the feature functions is a very hard problem. In our scenario, the Q-values associated to states with similar features (*e.g.*, the number of inelastic users) are increasingly non linear as the system becomes larger. As a result, linearization does not provide a good performance in our case.

Neural Networks (NNs) are a more powerful and flexible tool for generalization. RL algorithms that employ NNs are called DRL algorithms: N3AC belongs to this family. One of the key features of such a NN-based approach is that it only requires storing a very limited number of variables, corresponding to the weights and biases that compose the network architecture; yet, it is able to accurately estimate the $\mathcal{F}$ function for a very large number of state/action pairs.

In the rest of this section, we review the DRL design principles (Section 3.4.5) and explain how these principles are applied to a practical learning algorithm for our system (Section 3.4.6).

3.4.5. Deep Reinforcement Learning framework

The fundamental building blocks of DRL algorithms are the following ones [19]:

- A set of labeled data (*i.e.*, system inputs for which the corresponding outputs are known) which is used to train the NN (*i.e.*, teach the network to approximate the features of the system).

- A loss function that measures the neural network performance in terms of training error (*i.e.*, the error made when approximating the known output with the

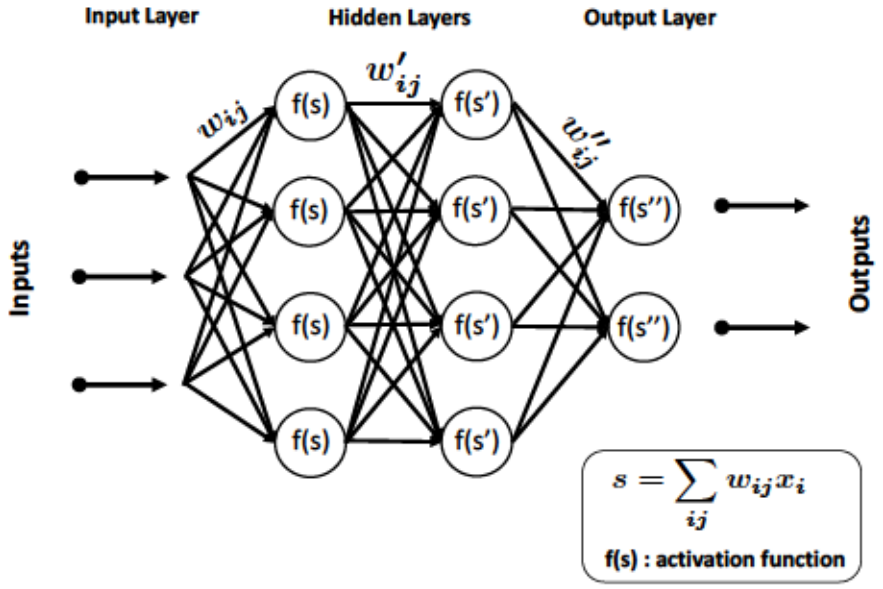


Figure 3.8: Neural networks internals.

given input).

- An optimization procedure that reduces the loss functions at each iterations, making the NN eventually converge.

There are many different Machine Learning (ML) schemes that make use of NNs, which are usually categorized as supervised, unsupervised and RL. A ML system is supervised or unsupervised depending on whether the labeled data is available or not, and it is a RL system when it interacts with the environment receiving feedback from its experiences. N3AC falls under the latter category, and, within RL, it falls under the category of DRL. Since the seminal work in [42], DRL techniques have gained momentum and are nowadays one of the most popular approaches for RL. In spite of the bulk of literature available for such techniques, devising the N3AC algorithm involves a number of design decisions to address the specificities of our problem, which are summarized in the following.

Neuron internal configuration. An exemplary NN is illustrated in Figure 3.8, where we have multiple layers of interconnected neurons organized as: (*i*) an input, (*ii*) an output and (*iii*) one or more hidden layers. As activation function N3AC employs the Rectified Linear Unit (ReLU) [47].

Neural Network Structure. One of the design choices that needs to be taken when devising a NN approach is the the way neurons are interconnected among them. The most common setup is *feed-forward*, where the neurons of a layer are fully interconnected with the ones of the next. There are also other configurations, such as the *convolutional* or the *recurrent* (where the output is used as input in the next iteration). However, the

best choice for a system like the one studied in this paper is the feed-forward. Indeed convolutional networks are usually employed for image recognition, while recurrent are useful when the system input and the output have a certain degree of mutual relation. None of these match our system, which is memoryless as it is based on a Markovian approach. Furthermore, our NN design relies on a single hidden layer. Such a design choice is driven by the following two observations: (*i*) it has been proven that is possible to approximate any function using NN with a single hidden layer [48], and (*ii*) while a larger number of hidden layers may improve the accuracy of the NN, it also involves a higher complexity and longer training period; as a result, one should employ the required number of hidden layers but avoid building a larger network than strictly necessary.

Back-propagation algorithm selection. N3AC following classical ML applications, adjusts weights using a Gradient Descent approach: the measured error at the output layer is back-propagated to the input layer changing the weights values of each layer accordingly [47]. More specifically, N3AC employs the RMSprop [49] Gradient Descent algorithm.

Integration with the RL framework. One of the critical requirements for N3AC is to operate without any previously known output, but rather interacting with the environment to learn its characteristics. Indeed, in N3AC we do not have any "ground truth" and thus we need to rely on estimations of the output, which will become more accurate as we keep exploring the system. While this problem has been extensively studied in the literature [19, 50], we need to devise a solution that is suitable for the specific problem addressed. In N3AC, we take as the output of the NN the average revenues expected at a given state when taking a specific decision. Once the decision is taken, the system transitions to the new state and we measure the average revenue resulting from the decision taken (0 in case of a rejection and ρt in case of an acceptance). Then, the error between the estimated revenue and the measured one is used to train the NN, back-propagating this error into the weights. As depicted in Fig 3.9, N3AC uses two different NNs: one to estimate the revenue for each state when the selected action is to accept the incoming request, and another one when we reject the request. Upon receiving a request, N3AC polls the two NNs and selects the action with the highest expected revenue; then, after the transition to the new state is performed, the selected NN is trained. More details about the N3AC operation are provided in the next section.

Exploration vs exploitation trade-off. N3AC drives the selection of the best action to be taken at each time step. While choosing the action that maximizes the revenue at each step contributes to maximizing the overall revenue (referred to as *exploitation* step), in order to learn we also need to visit new (still unknown) states even if this may eventually lead to a suboptimal revenue (referred to as *exploration* step). This procedure is especially important during the initial interaction of the system, where estimates are very inaccurate. In N3AC, the trade-off between exploitation and exploration is regulated

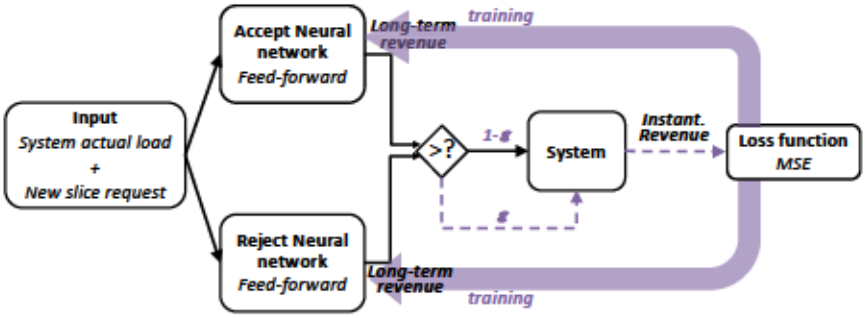


Figure 3.9: High-level design of AI-based slice admission control.

Algorithm 3 N3AC algorithm.

An event is characterized by:
s, a, s', r, t [starting state, action taken, landing state, obtained reward, transition time].

Initialization:
Neural Network's weights ← random values.

Procedure

1: Estimate $Q(s', a')$ for each action a' available in state s' through the NN.
2: Build the target value with the new sample observation as follows:

$$target = R(s, a, s') - \sigma t_n + \max_{a'} Q(s', a') \tag{3.17}$$

▷ where t_n is the transition time between two subsequent states s and s' after action a.
3: Train the NNs through RMSprop algorithm:
4: **if** $s \notin$ admissibility region boundary **then**
train the NN with the error given by the difference between the target value (eq. 3.17) and the measured one.
5: **else**
train the NN corresponding to accepted requests by applying a *"penalty"* and train the NN corresponding to rejected requests as in step 4.
6: **end if**

by the γ parameter, which indicates the probability of taking an exploration step. In the setup used in this paper, we take $\gamma = 0.1$. Once the NNs are fully trained, the system goes into exploitation only, completely omitting the exploration part.

3.4.6. Algorithm description

In the following, we describe the proposed N3AC algorithm. This algorithm builds on the Neural Networks framework described above, exploiting RL to train the algorithm without a ground truth sequence. The high-level algorithm design is illustrated in Fig 3.9 and it consists of the following high level steps (see Algorithm 3 for the pseudocode):

- *Step 1, acceptance decision:* In order to decide whether to accept or reject an incoming request, we look at the expected average revenues resulting from

accepting and rejecting a request in the two NNs, which we refer to as the Q-values. Specifically, we define $Q(s,a)$ as the expected cumulative reward when starting from a certain state s with action a, compared to a baseline σ given by the optimal policy reward when starting from state 0, i.e.,

$$Q(s,a) = \mathbb{E}\left[\lim_{t\to\infty} \sum_{n=0}^{D(t)} R_n - \sigma t | s_0 = s, a_0 = a\right] \tag{3.18}$$

where $D(t)$ is the number of requests received in a period t, R_n is the revenue obtained with the n^{th} request and $\sigma = \mathbb{E}[\lim_{t\to\infty} \frac{1}{t}\sum_{n=0}^{D(t)} R_n | s_0 = 0]$ under the optimal policy. Then, we take the decision that yields the highest Q-value. This procedure is used for elastic slices only, as inelastic slices shall always be accepted as long as there is sufficient room. When there is no room for an additional slice, requests are rejected automatically, regardless of their type.

- *Step 2, evaluation:* By taking a decision in Step 1, the system experiences a transition from state s at step n, to state s' at step $n+1$. Once in step $n+1$, the algorithm has observed both the reward obtained during the transition $R(s,a)$ and a sample t_n of the transition time. The algorithm trains the weights of the corresponding NN based on the error between the expected reward of s estimated at step n and the target value. This step relies on two cornerstone procedures:

  Step 2a, back-propagation: This procedure drives the weights update by propagating the error measured back through all the NN layers, and updating the weights according to their gradient. The convergence time is driven by a learning rate parameter that is used in the weight updates.

  Step 2b, target creation: This procedure is needed to measure the accuracy of the NNs estimations during the learning phase. At each iteration our algorithm computes the observed revenue as follows:

$$\omega = R(s,a,s') - \sigma t_n + \max_{a'} Q(s',a'), \tag{3.19}$$

  where $R(s,a,s')$ is the revenue obtained in the transition to the new state. As we do not have labeled data, we use ω to estimate the error, by taking the difference between ω and the previous estimate $Q_{n+1}(s,a)$ and using it to train the NN. When the NN eventually converges, ω will be close to the Q-values estimates.

- *Step 3, penalization:* When a state in the boundary of the admissibility region is reached, the system is forced to reject the request. This should be avoided as it may force the system to reject potentially high rewarding slices. To avoid such

cases, N3AC introduces a *penalty* on the Q-values every time the system reaches the border of the admissibility region. With this approach, if the system is brought to the boundary through a sequence of highly rewarding actions, the penalty will have small effect as the Q-values will remain high even after applying the penalty. Instead, if the system reaches the boundary following a chain of poorly rewarding actions, the impact on the involved Q-values will be much higher, making it unlikely that the same sequence of decisions is chosen in the future.

- *Step 4, learning finalization:* Once the learning phase is over, the NN training stops. At this point, at a given state we just take the the action that provides the highest expected reward.

We remark that the learning phase of our algorithm does not require specific training datasets. Instead, the algorithm learns from the real slice requests on the fly, during the real operation of the system; this is the so-called *exploration phase*. The training corresponding to such an exploration phase terminates when the algorithm has converged to a good learning status, and is triggered again when the system detects changes in the system that require new training.

3.5. Performance Evaluation

In this section we evaluate the performance of NN via simulation. Unless otherwise stated, we consider the scenario described in Section 3.3.4.

Following the N3AC algorithm proposed in the previous section, we employ two feed-forward NNs, one for accepted requests and another one for rejected. Each neuron applies a ReLU activation function, and we train them during the exploration phase using the NNs RMSprop algorithm implementation available in Keras (https://keras.io/); the learning parameter of the RMSprop Gradient Descent algorithm [49] is equal to 0.001. The number of input nodes in the NN is equal to the size of the space state (*i.e.*, the number of considered classes plus one for the next request k), the number of neurons in the hidden layer equal to 40 for the scenario described in Sections 3.5.3 and 3.5.4 and 20 for the others, and the output layer is composed of one neuron, applying a linear function. Note that, while we are dealing with a specific NN structure, one of the key highlights of our results is that the adopted structure works well for a wide range of different 5G networks.

In the results given in this section, when relevant we provide the 99% confidence intervals over an average of 100 experiments (note that in many cases the confidence intervals are so small that they cannot be appreciated).

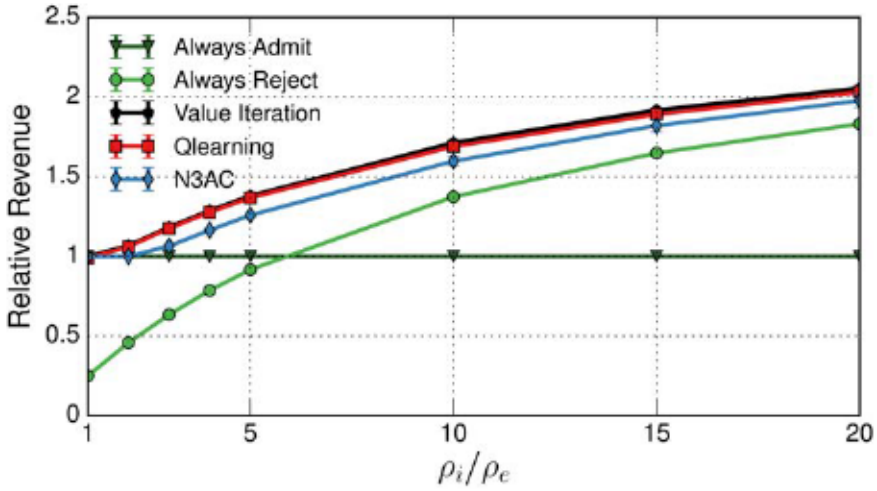


Figure 3.10: Revenue vs. ρ_i/ρ_e.

3.5.1. Algorithm Optimality

We first evaluate the performance of the N3AC algorithm (which includes a hidden layer of 20 neurons) by comparing it against: (*i*) the benchmark provided by the optimal algorithm, (*ii*) the Q-learning algorithm proposed in Section 3.4.2, and (*iii*) two naive policies that always admit elastic traffic requests and always reject them, respectively. In order to evaluate the optimal algorithm and the Q-learning one, which suffers from scalability limitations, we consider a relatively small scenario. Figure 3.10 shows the relative average reward obtained by each of these policies, taking as baseline the policy that always admit all network slice requests (which is the most straightforward algorithm).

We observe that N3AC performs very closely to the Q-learning and optimal policies, which validates the proposed algorithm in terms of optimality. We further observe that the revenue improvements over the naive policies is very substantial, up to 100% in some cases. Again, as expected, for small ρ_i/ρ_e the policy that always admits all requests is optimal: in this case both elastic and inelastic slices provide the same revenue. In contrast, for very large ρ_i/ρ_e ratios the performance of the "always reject" policy improves, as in this case the revenue obtained from elastic traffic is (comparatively) much smaller.

3.5.2. Learning time and adaptability

One of the key advantages of the N3AC algorithm as compared with Q-learning is that it requires a much shorter learning time. This is due to the fact that with N3AC the knowledge acquired at each step is used to update the Q-values of all states, while Q-learning just updates the Q-value of the lookup table for the state being visited. To evaluate the gain provided by the NNs in terms of convergence time, we analyze the evolution of the expected revenue over time for the N3AC and the Q-learning algorithms.

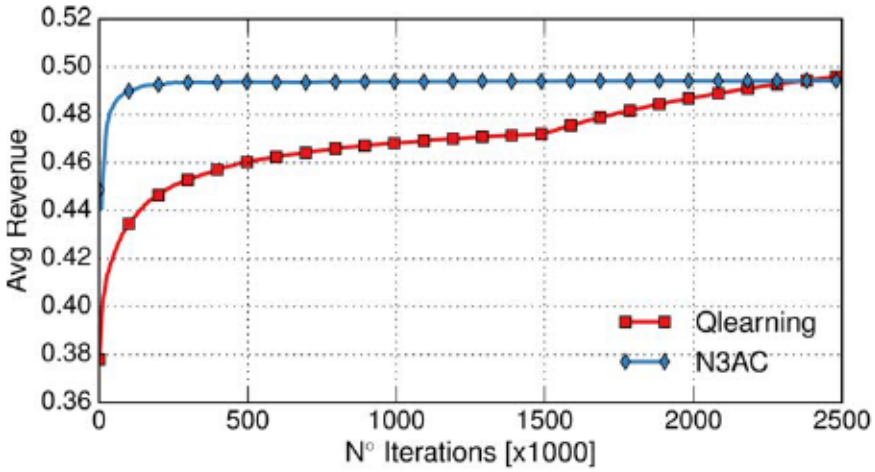


Figure 3.11: Learning time for N3AC and Q-learning.

The results are shown in Figure 3.11 as a function of the number of iterations. We observe that after few hundred iterations, N3AC has already learned the correct policy and the revenue stabilizes. In contrast, Q-learning needs several thousands of iterations to converge. We conclude that N3AC can be applied to much more dynamic scenarios as it can adapt to changing environments. Instead, Q-learning just works for relatively static scenarios, which limits its practical applicability. Furthermore, Q-learning cannot scale to large scenarios, as the learning time (and memory requirements) would grow unacceptably for such scenarios.

When the network conditions change, *e.g.*, the arrival pattern of slice requests, this is detected by the system, and a new training period is triggered. To evaluate the system performance under such conditions, Figure 3.12 illustrates the behavior of N3AC and Q-learning. In this experiment, the arrival rate of elastic network slices is reduced to one half at a given point in time, and this is detected as the revenue drops beyond a given threshold (which we set to 10%). We observe that N3AC rapidly moves to the best point of operation, while Q-learning needs much more time to converge, leading to a substantially lower revenue. We further observe that, even in the transients, N3AC obtains a fairly good performance.

3.5.3. Large-scale scenario

The previous results have been obtained for a relatively small scenario where the evaluation of the optimal and Q-learning algorithm was feasible. In this section, we assess the performance of the N3AC algorithm in a large-scale scenario; indeed, one of the design goals of this algorithm is its scalability to large scenarios. We consider a scenario with eight slice classes, four for elastic traffic and four for inelastic. For each traffic type, we allow four network slice sizes, linearly distributed among $C/10$ and $C/20$.

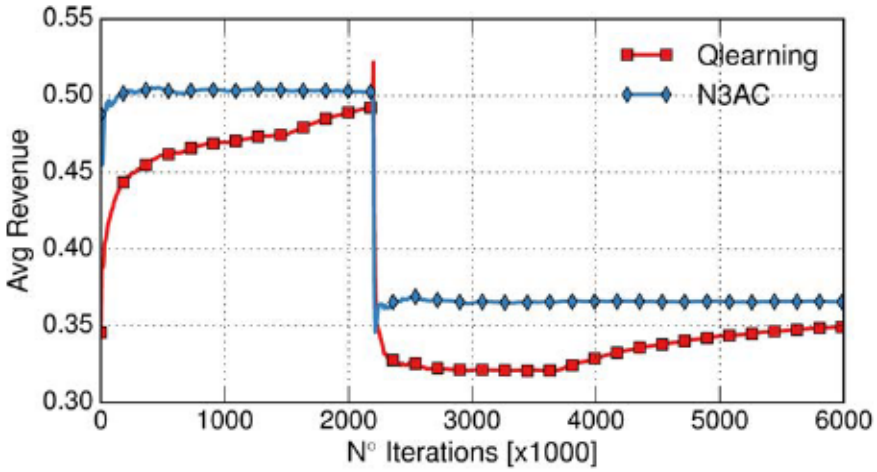


Figure 3.12: Performance under changing conditions.

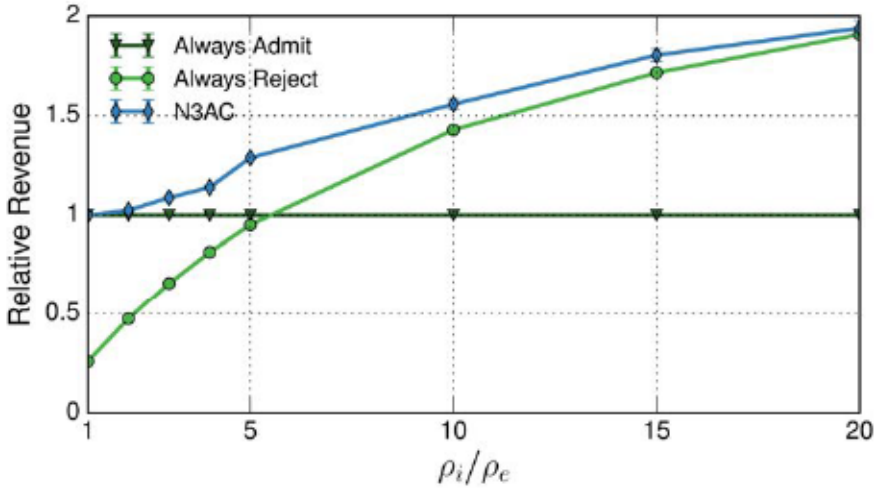


Figure 3.13: Revenue vs. ρ_i/ρ_e.

We have the same throughput guarantees for elastic and inelastic traffic as in the previous experiment ($R_i = R_e = C_b/10$) and thus we have the same admissibility region (although the space state is much larger now). We set μ and λ parameters in a way that the load of the network is similar to the previous experiment.

In this larger scenario, the optimal and Q-learning algorithms are not feasible. Hence, we evaluate the performance of N3AC and compare it against the naive policies only. Figure 3.13 shows the relative average reward obtained by each of these policies, taking as baseline the policy that always admits all network slice requests. Similarly to the evaluation performed in the previous experiment, we observe that the N3AC algorithm always substantially outperforms the naive policies. As expected, for small ρ_i/ρ_e the policy that always admits all requests is optimal, while for very large ρ_i/ρ_e ratios the performance of "always reject" policy improves since the revenue obtained from the elastic

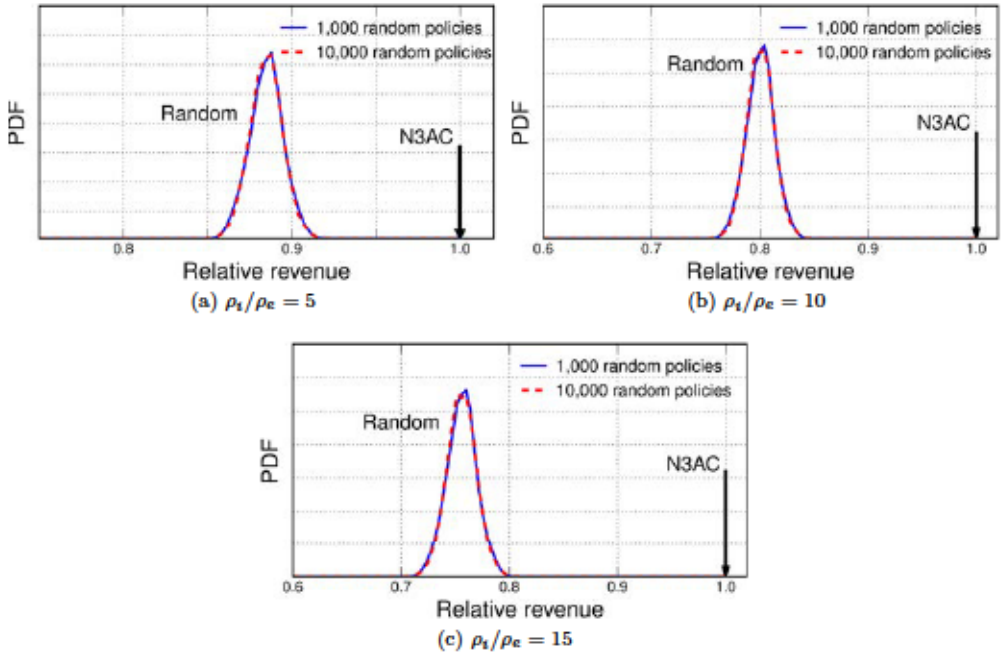


Figure 3.14: The distribution of the revenues obtained by random smart policies compared to the N3AC algorithm.

traffic is much smaller.

3.5.4. Gain over random policies

While the result of the previous section shows that the proposed algorithm provides high gains, it is only compared against two naive policies and thus does not give an insight on the real revenue gains that could be achieved over smarter, yet not optimal policies. To this end, we compare the performance of the N3AC algorithm against a set of "smart" random policies which work as follows: (i) inelastic network slices requests are always accepted, and (ii) the decision of rejecting an elastic request is chosen randomly upon defining the policy for each different state. Then, by drawing a high number of random policies, it is to be expected that some of them provide good performance.

Figure 3.14 compares N3AC against the above approach with 1,000 and 10,000 different random policies, respectively. We note that the improvement achieved with 10,000 random policies over 1,000 is very small, which shows the the chosen setting for the random policies approach is appropriate and provides the best performance that can be achieved with such an approach. From the figure, we can see that N3AC provides substantial gains over the best performing random policy (around 20%). This confirms that a smart heuristic is not effective in optimizing revenue; indeed, with such a large space state it is very difficult to calibrate the setting for the acceptance of elastic slices

that maximizes the resulting revenue. Instead, by using a NN-based approach such as N3AC, we are capable of accurately capturing such a large space state within a limited range of parameters and thus drive acceptance decisions towards very high performance.

3.5.5. Memory and computational footprint

One of the key aspects of the proposed framework is the memory footprint, which has a strong impact on scalability. By using NNs, N3AC does not need to keep track of the expected reward for each individual state-action $Q(s,a)$, but it only stores the weights of the NNs. Indeed, NNs capture the dynamics of the explored system based on a small number of weights, which are used to estimate the Q-values for all the states of the system. This contrasts with Q-learning, which requires to store data for each individual state. As the number of weights, fixed by the NN layout, is much smaller than the total number of states, this provides a much higher scalability, specially when the number of states grows substantially. For example, the large scale scenario evaluated in Section 3.5.3 has an internal space state of around 500 thousand states, which makes the Q-learning technique unfeasible for such a scenario. In contrast, N3AC only requires storing state for around 400 parameters, which represents a huge improvement in terms of scalability.

In addition to memory, the computational footprint also has a strong impact on scalability. In order to understand the computational load incurred by N3AC, we measured the time elapsed in the computation for one iteration. Table 3.1 gives the results obtained with a NVIDIA GTX 1080 GPU platform for different system scenarios in terms of neurons, number of base stations and number of users. Results show that computational times are very low, and the differences between the various scenarios are almost negligible, which further confirms the ability of N3AC to scale up to very large network scenarios.

Number of neurons	Number of base stations	Number of users	Computational time (sec)
40	50	500	0.0181
40	100	1000	0.0194
40	250	1000	0.0195
100	250	2500	0.0192
100	500	2500	0.0197
100	500	5000	0.0199

Table 3.1: Computational load for different network scenarios.

3.5.6. Different traffic types

Our analysis so far has focused on two traffic types: elastic and inelastic traffic. In this section, we address a different scenario that includes the traffic types corresponding to the

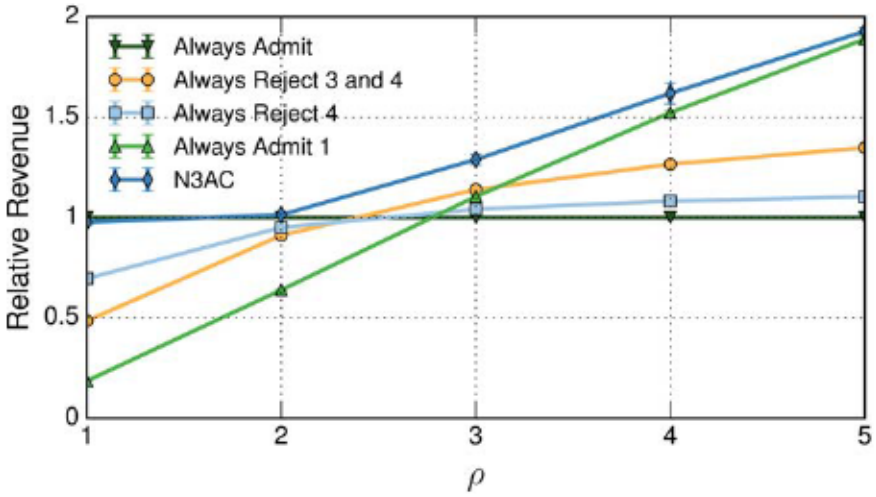


Figure 3.15: Revenue vs. ρ_i/ρ_e.

four service classes defined by 3GPP [51] (hereafter we refer to them as class 1 to class 4, where class 1 is the one with most stringent delay requirements). In line with the analysis of Section 3.2, for this scenario with 4 different traffic types we take the admissibility region $\mathcal{A}^*$ given by (i) $|\mathcal{T}_1| \leq \mathcal{T}_1^{max}$, (ii) $|\mathcal{T}_1|+|\mathcal{T}_2| \leq \mathcal{T}_2^{max}$, (iii) $|\mathcal{T}_1|+|\mathcal{T}_2|+|\mathcal{T}_3| \leq \mathcal{T}_3^{max}$, and (iv) $|\mathcal{T}_1|+|\mathcal{T}_2|+|\mathcal{T}_3|+|\mathcal{T}_4| \leq \mathcal{T}_4^{max}$. For this scenario, we run the same experiment as in Section 3.5.3, varying the price ratio ρ among different classes as follows: $r_k = \rho \cdot r_{k+1}$ $\forall k$, where r_k is the revenue generated by class k. Figure 3.15 compares the performance provided by the N3AC algorithm in this scenario against the one provided by naive policies which only accept a subset of classes. We can observe that N3AC provides very high gains when compared to all the naive policies, which confirms that our approach can be successfully applied to scenarios with more traffic types, such as, e.g., the 3GPP service classes.

4 Resource Orchestration for Network Slicing

Network Slicing will enables the sharing of the next mobile networks generation by dividing the network infrastructure into logical slices devoted to different services and customized to their needs. In the first part of this thesis we analyzed the impact of Network Slicing over the business model of the next mobile network generation. We unveiled the network management complexity when the resource to be shared is spectrum. In this scenario we identified the new players, *i.e.*, the tenants and the Infrastructure Provider (InP), and the need of a network capacity broker algorithm to decide on whether admit or reject a new slice request to meet the Service Level Agreements (SLAs) and maximizing the InP's revenue.

An optimal and a heuristic algorithms have been designed to optimize the network slicing market: the first provides a benchmark to evaluate the quality of the practical algorithm based on Deep Reinforcement Learning (DRL) that provides flexibility and scalability in order to be employed in large and complex real scenarios while providing close to optimal performance.

The emergence of sliced networks also promises to skyrocket the complexity of resource management and orchestration, moving from the rather limited reconfiguration possibilities offered by current Operations and Business Support System (OSS/BSS) to a rich, software-defined layer that manages thousands of slices belonging to hundreds of tenants on the same infrastructure [52].

To cope with the new milieu, network operators are striving to make resource management and orchestration (MANO) processes highly automated. To realize the 5G principle of *cognitive network management* [53], two complementary technologies are needed: (*i*) technical solutions that enable end-to-end Network Function Virtualization (NFV), and provide the flexibility necessary for resource reallocation; and, (*ii*) data analytics that operate on mobile traffic measurement data, automatically identify demand patterns, and anticipate their future evolution.

From a technical standpoint, solutions that implement NFV at different network levels are well established, and start to be tested and deployed. Examples include

current MANO platforms architectures like ETSI NFV [54], and implementations such as OSM [55] or ONAP [56], which allow to reconfigure and reassign resources to Virtual Network Functions (VNFs) on the fly. By contrast, the integration of data analytics in cognitive mobile networks is still at an early stage. Nowadays, resource assignment to VNFs is a reactive process, mostly based on hysteresis thresholding and aimed at self-healing or fault tolerance. There is a need for proactive, data-driven, automated solutions that enable cost-efficient network resource utilization, by anticipating future needs for capacity and timely reallocating resources just where and when they are required. The focus of our work is precisely on the design of data analytics for the anticipatory allocation of resources in cognitive mobile networks. Specifically, we seek a machine learning solution that runs on traffic measurements and provides operators with information about the capacity needed to accommodate future demands at each network slice – a critical knowledge for data-driven resource orchestration. In Section 4.1, we first introduce *capacity* forecast, a new concept at the base of our solution, and then provide a review of related works highlighting the novelty of our proposed method in Section 4.2. We then describe the overall framework of DeepCog in Section 4.3 and detail the design of its most critical component, *i.e.*, the loss function, in Section 4.4. The quality of the solution is then assessed in realistic scenarios in Section 4.5.

4.1. Network management and forecasting

In this section we introduce the concept of capacity forecast, a fundamental notion that inspired the design of DeepCog. Indeed, affecting the resource orchestration decisions, capacity forecast directly influences the monetary impact for the network operator in terms of operating expenses.

We start identifying two macroscopic categories of operating cost, *Overprovisioning* and *Service Level Agreement (SLA) violation*. Then, the motivation behind the need of capacity forecast instead of traffic forecast is presented. We conclude the section with a comparison to highlight the benefits obtained by employing capacity forecast.

In the context of network resource management and orchestration, an Infrastructure Provider (InP) at the moment of select the amount of resources needed to accommodate mobile traffic load, can incur in two main cost categories:

- *Overprovisioning* – when providing excess capacity with respect to the actual resource demand, the operator incurs a cost due to the fact that it is reserving more resources than those needed to a network entity (*e.g.*, a network slice, a network function, or a virtual machine). As resources are typically isolated across slices, this seizes the excess resources from other network entities that may have possibly used them. At a global system level, continued overprovisioning implies that the operator will have to deploy more resources than those required to accommodate the

user demand, limiting the advantage of a virtualized infrastructure and of cognitive networking solutions in general.

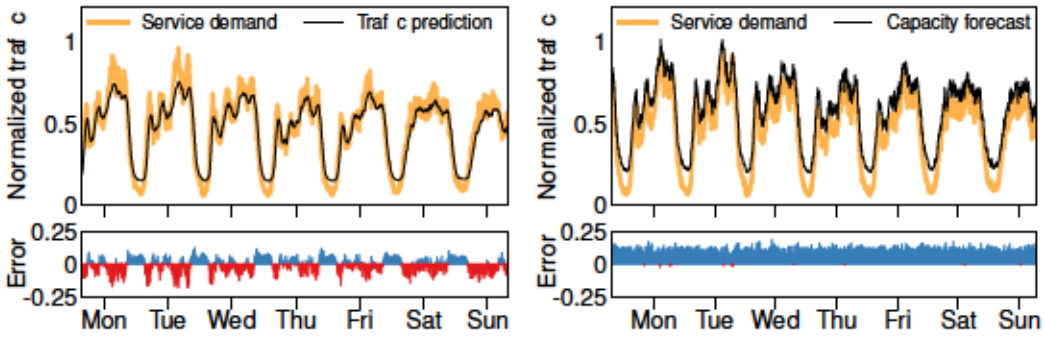
- *SLA violation* – if insufficient resources are allocated to a network entity, users will suffer low Quality of Service (QoS), or even discontinued service. This has an indirect price for the operator, in terms of customer dissatisfaction and increased churning rates, which is not simple to quantify. However, in emerging contexts such as those promoted by network slicing, underprovisioning also entails a different, more direct and quantifiable economic penalty for the operator. Under slicing, operators will sign SLAs with the mobile service providers, which need to be strictly enforced. Underprovisioning means violating such SLAs, which results in substantial monetary fees for the network operator.

Clearly, the cost is not the same in the two cases, and it may also vary depending on the specific settings, including the nature of the concerned resources, the technologies deployed in the network infrastructure, or the market strategies of the operator. In all cases, we posit that, once suitably modeled, such costs shall be at the core of the orchestrating decisions.

Legacy techniques for the prediction of mobile network traffic, such as the one reviewed in Section 4.2, fall short in this respect. Such models aim at perfectly matching the temporal behavior of traffic, independently of whether the anticipated demand is above or below the target, and are thus agnostic of the aforementioned costs. As a result, they return forecasts as that depicted in Fig. 4.1a, which refers to a real-world case study of YouTube video streaming traffic at a core network datacenter. Note that no distinction is made between positive and negative errors, which leads to substantial SLA violations covering roughly half of the observation time. The operator may then attempt to apply overprovisioning to the output provided by such a traffic predictor. Unfortunately, legacy forecast models do not offer any insight on how large the excess resource allocated on top of the forecast demand should be.

We argue that a more effective anticipatory resource allocation can be achieved by designing machine learning solutions that anticipate the minimum provisioned *capacity* needed to cut down SLA violations. This closes the present gap between simple traffic prediction and practical orchestration as it provides the operator with an explicit *capacity forecast* that mitigates underprovisioning in Fig. 4.1b while minimizing unnecessary resource reservation.

Once the resource allocation is determined according to the capacity forecast provided by DeepCog, there is the need of lower level mechanisms to enforce the envisioned allocation. As an example, we have designed CARES, a computational-aware radio resources scheduler. The current Radio Access Network (RAN) protocol stack has been designed under the assumption that required computational resources



(a) Legacy traffic predictor

(b) DeepCog capacity forecast

Figure 4.1: Top: actual and predicted weekly demands for YouTube at a datacenter controlling 470 4G eNodeBs. Bottom: levels of overprovisioning (blue) and capacity violations (red) over time. (a) Output of a recent deep learning predictor of mobile traffic [57]. (b) Output of DeepCog, tailored to anticipatory network resource allocation. The best view of this figure is in colors.

are always available, and RAN functions are not prepared to cope with computational outages [58]. Indeed, when such computational outages occur, current virtualized RAN implementations [59] just drop the frame being processed, and as a result they see their performance severely degraded. To provide a more robust behavior against computational outages at the process of enforcing radio resources to network slices, a re-design of the RAN protocol stack is required. As a first step towards this end, in [6, 7] we propose CARES, a mechanism that *jointly* optimizes scheduling and Modulation and Coding Scheme (MCS) selection making these functions *computationally aware*. In this way, our solution provides a graceful performance degradation in presence of computational outages, in contrast to the severe degradation with the current stack, improving the performance of a virtualized RAN system with temporarily *limited* computational resources.

In the next section we introduce DeepCog, a machine learning solution that running on traffic measurements provides such a critical knowledge for data-driven resource orchestration.

4.2. Related Works

Applications to networking problems of machine learning in general, and of deep learning in particular, are starting to become popular. Artificial intelligence can indeed be applied to solve many different problems that emerge in computer networks, as highlighted in recent comprehensive surveys on the topic [60, 61].

In the context of network management, emerging paradigms like slicing increase

substantially the complexity of orchestrating network functions and resources, at all levels. For instance, intelligence is needed for the admission control of new slices: as resources are limited and slicing entails their strong isolation, this is critical to ensure that the system operates efficiently. With potentially hundreds of slices allocated simultaneously, and a need to anticipate highly profitable future requests, the decision space for admission control becomes so large that traditional approaches become impractical. Solutions based on deep learning architectures represent here a viable approach [62]. Similar considerations apply to other aspects of sliced network management, *e.g.*, the allocation of computational resources to slices at the radio access, based on transmission (*e.g.*, modulation and coding scheme, channel load) and environmental (*e.g.*, signal quality, hardware technology) conditions [63], or the anticipatory reservation of Physical Resource Blocks (PRBs) to user traffic to be served in target network slices [64].

Our specific problem relates to the orchestration of generic resources (*e.g.*, CPU time, memory, storage, spectrum) to slices at different network entities, which is tightly linked to mobile traffic prediction. The literature on forecasting network traffic is in fact vast [60, 65]. Solutions to anticipate future offered loads in mobile networks have employed a variety of tools, from autoregressive models [66–68] to information theoretical tools [69], passing by Markovian models [70] and deep learning [57, 62, 64, 71, 72]. However, we identify the following major limitations of current predictors when it comes to supporting resource orchestration in mobile networks.

First, predictors of mobile traffic invariably focus on providing forecasts of the future demands that minimize some absolute error [60, 65]. As explained in Section 4.1, this approach leads to predicted time series that deviate as little as possible from the actual traffic time series, as exemplified in Fig. 4.1a for a real-world case study. While reasonable for many applications, such an output is not appropriate for network resource orchestration. The operator aims at provisioning sufficient capacity to accommodate the offered load at *all* times, since failing to do so implies high costs in terms of high subscribers' churn rates, as well as significant fees for violating SLAs signed with tenants. Yet, if an operator decided to allocate resources based on a legacy prediction like that in Fig. 4.1a, it would incur into capacity violations most of the time (as illustrated in the bottom subplot).

Second, with the adoption of network slicing, forecasts must occur at the slice level, *i.e.*, for specific mobile services in isolation. However, most traffic predictors, including recent ones, are evaluated with demands aggregated over all services [57, 71, 72]. This is an easier problem, since aggregate traffic yields smoother and more regular dynamics, hence previous solutions may not handle well the bursty, diversified traffic exhibited by each service. The only attempts at anticipating the demands generated by specific mobile services have been made by using multiple-input single-output (MISO) autoregressive models [73], and hybrid prediction methods that incorporate α-stable models and sparsity

with dictionary learning [69].

Third, existing machine learning predictors for mobile traffic typically operate at base station level [57,72]. However, Network Function Virtualization (NFV) operations mainly occur at datacenters controlling tens (*e.g.*, at the mobile edge) to thousands (*e.g.*, in the network core) of base stations. Here, prediction should be more efficient when performed on the aggregate traffic at each datacenter, where orchestration decisions are taken, rather than combining independent forecasts from each base station.

Our proposed solution, DeepCog, addresses all of the open problems above, by implementing a first-of-its-kind predictor that anticipates the minimum provisioned capacity needed to cut down SLA violations. This closes the present gap between traffic prediction and practical orchestration, as it provides the operator with an explicit capacity forecast that mitigates underprovisioning in Fig. 4.1b while minimizing unnecessary resource reservation.

4.3. A DL Framework for Resource Orchestration

In this section we introduce DeepCog [11], a new mobile traffic data analytics tool that is explicitly tailored to solve capacity forecast problem. The design of DeepCog yields multiple novelties, summarized as follows:

- It hinges on a deep learning architecture inspired by recent advances in image and video processing, which exploits space- and time-independent correlations typical of mobile traffic and computes outputs at a datacenter level;
- It leverages a customized loss function that targets capacity forecast rather than plain mobile traffic prediction, letting the operator tune the balance between overprovisioning and demand violations;
- It provides long-term forecasts over configurable prediction horizons, operating on a per-service basis in accordance with network slicing requirements.

Overall, these design principles jointly solve the problem of capacity forecast in network slicing. This is illustrated by Fig. 4.1b, which shows an example of the required capacity forecast by DeepCog in a real-world case study. We remark that DeepCog is one of the very first examples of rigorous integration of machine learning into a cognitive network management process, and marks a difference from the common practice of embedding vanilla deep learning structures into network operation [60].

The design of DeepCog is outlined in Fig. 4.2. Its organization is that typical of deep learning systems, and it stems from (*i*) properly formatted *input* data used to build the forecast, which, in our case, represents the current and past traffic associated to a specific network slice as a tensor. Such input is fed to (*ii*) a *deep neural network* architecture that

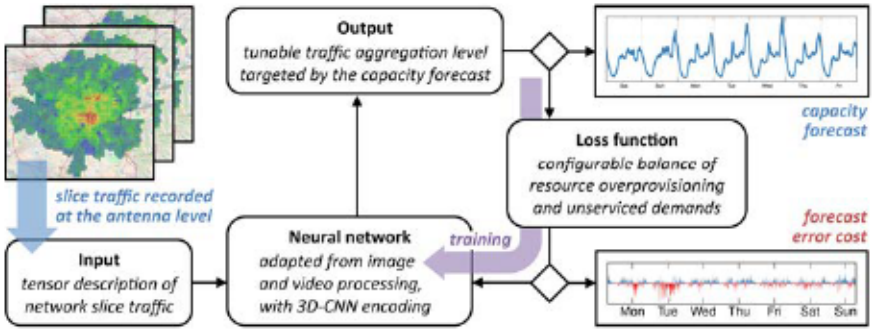


Figure 4.2: Outline and interaction of the DeepCog components.

extrapolates and processes input features to provide (*iii*) an *output* value: the capacity forecast. During the training phase, the output is used to evaluate (*iv*) a *loss function* that quantifies the error with respect to the ground truth, and, in DeepCog, accounts for the costs associated to resource overprovisioning and service request denial.

Below, we present each of the components of the framework, and discuss its mapping to the elements of a 5G network architecture running cognitive resource management.

4.3.1. DeepCog Framework

In our network model, we consider that time is divided in slots, which we denote by t. Let $\delta_s^i(t)$ be the traffic associated with slice s that is observed at base station $i \in \mathcal{N}$ and time t. A *snapshot* of the demand of slice $s \in \mathcal{S}$ at time t is given by a set $\boldsymbol{\delta}_s(t) = \{\delta_s^1(t), \ldots, \delta_s^N(t)\}$, and provides a global view of the traffic for that slice at time t across the whole network. We let $\mathcal{N}$ denote the set of N base stations in the network, and $\mathcal{M}$ the set of $M < N$ datacenters. For each slice $s \in \mathcal{S}$, base stations are associated to datacenters via a surjective mapping $f_s : \mathcal{N} \to \mathcal{M}$, such that a datacenter $j \in \mathcal{M}$ serves the aggregated load of slice s for all of the associated bases stations.[1] With this mapping, the traffic for slice s processed by datacenter j at time t is given by $d_s^j(t) = \sum_{i|f_s(i)=j} \delta_s^i(t)$. Then, the set of demands across all datacenters is given by $\mathbf{d}_s(t) = \{d_s^1(t), \ldots, d_s^M(t)\}$.

Let us denote the allocated capacity for slice s at datacenter j and time t as $c_s^j(t)$, and the set of capacities at all $j \in \mathcal{M}$ as $\mathbf{c}_s(t) = \{c_s^1(t), \ldots, c_s^M(t)\}$. Then, the capacity forecast problem is that of computing a *constant* capacity $\mathbf{c}_s(t, T_h) = \{c_s^1(t, T_h), \ldots, c_s^M(t, T_h)\}$ that is allocated in the network datacenters over a time horizon T_h, *i.e.*, through an interval between the present time t and a future time $t + T_h$. In practice, this models the typical situation where the resource reconfiguration frequency is limited (*e.g.*, by the NFV technology), and the operator must decide in advance the amount of resources that

[1] We remark that DeepCog works for any arbitrary mapping, including, *e.g.*, flows from a slice in the same base station being split across datacenters, or associations among base stations and datacenters varying over time. As a matter of fact, DeepCog's learning process is based exclusively on the traffic load at each individual datacenter and is thus independent of the actual sources generating such traffic.

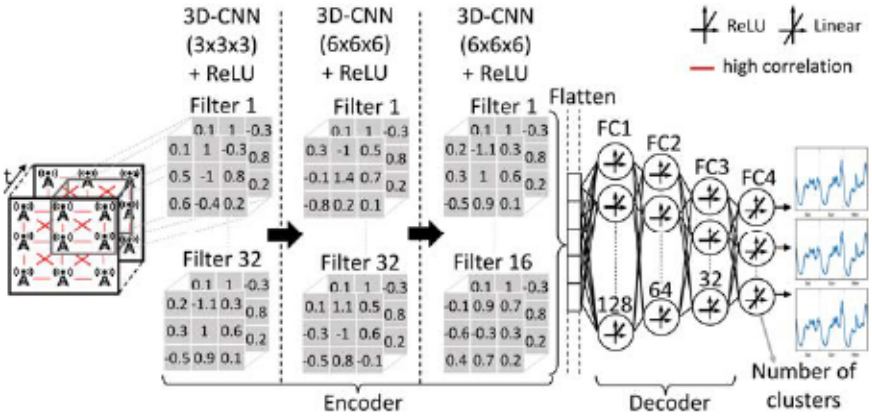


Figure 4.3: DeepCog neural network encoder-decoder structure.

will stay assigned to a slice until the next reallocation takes place. The time horizon T_h thus corresponds to the reconfiguration period, and the allocated capacity is such that $c_s^j(t) = c_s^j(t, T_h)\ \forall j \in \mathcal{M}, t \in [t, t+T_h]$.

The forecast builds on knowledge of the previous T_p traffic snapshots $\boldsymbol{\delta}_s(t-1), \ldots, \boldsymbol{\delta}_s(t-T_p)$. The quality of the capacity forecast $\mathbf{c}_s(t, T_h)$ is measured by means of a suitable loss function $\ell\left(\mathbf{c}_s(t, T_h), \mathbf{d}_s(t), \ldots, \mathbf{d}_s(t+T_h)\right)$. This function $\ell(\cdot)$ determines the compound cost of overprovisioning and underprovisioning network resources at the target datacenters, as produced by allocating a constant capacity $\mathbf{c}_s(t, T_h)$ when the actual time-varying demand is in fact $\mathbf{d}_s(t), \ldots, \mathbf{d}_s(t+T_h)$.

4.3.2. The Neural Network

DeepCog leverages a deep neural network structure composed of suitably designed encoding and decoding phases, performing a capacity forecasting prediction over a given time horizon. The structure is general enough that it can be trained to solve the capacity forecast problem for (i) network slices dedicated to different services with significantly diverse demand patterns, (ii) any datacenter configuration, and (iii) any time horizon T_h. The hyperparameters of the neural network have been tuned through extensive simulation and testing.

The design of the neural network structure in DeepCog is inspired by recent breakthroughs [74] in deep learning for image and video processing. As summarized in Fig. 4.3, the network is composed of an encoder that receives an input representing the mobile traffic data $\boldsymbol{\delta}_s(t-1), \ldots, \boldsymbol{\delta}_s(t-Tp)$ and maps important spatial and temporal patterns in such data onto a low-dimensional representation. Intuitively, the encoder extracts the relevant features from the input traffic tensors $\boldsymbol{\delta}_s(t-1), \ldots, \boldsymbol{\delta}_s(t-Tp)$ and the decoder leverages such features to generate a capacity forecast that is tailored to a given combination of slice, prediction time horizon, and datacenter class; *e.g.*, datacenters

deployed close to the radio access will show different features from those co-located with the Internet gateways.

The result of the encoder undergoes a flattening process that converts the 3D (space and time) tensor data into a unidimensional vector format. This is the input format required by the fully connected layers that form the decoder, which then generates the final capacity forecast $\mathbf{c}_s(t, T_h)$ at the target set of datacenters $\mathcal{M}$. Below, we detail the encoder and decoder implementations, and discuss the training procedure.

The Encoder. It is composed by a stack of three three-dimensional Convolutional Neural Network (CNN) layers [75]. Generic CNNs are a specialized kind of deep learning structure that can infer local patterns in the feature space of a matrix input. In particular, two-dimensional CNNs (2D-CNNs) have been extensively utilized in image processing, where they can complete complex tasks on pixel matrices such as face recognition or image quality assessment [76]. 3D-CNNs extend 2D-CNNs to the case where the features to be learned are spatiotemporal in nature, which adds the time dimension to the problem and transforms the input into a 3D-tensor. Since mobile network traffic exhibits correlated patterns in both space and time, our encoder employs 3D-CNN layers[2].

Formally, the 3D-CNN layers receive a tensor input $\mathcal{T}\left(\boldsymbol{\delta}_s(t-1)\right), \ldots, \mathcal{T}\left(\boldsymbol{\delta}_s(t-T_p)\right)$, where $\mathcal{T}(\cdot)$ is a transformation of the argument snapshot into a matrix. This input is processed by three subsequent 3D-CNN layers. Each neuron of these layers runs a filter $\mathcal{H}\left(\sum_\tau \mathbf{I}(\tau) * \mathbf{K}(\tau) + \mathbf{b}\right)$ where $\mathbf{I}(\tau)$ is the input matrix passed to the neuron (*e.g.*, $\mathbf{I}(\tau) = \mathcal{T}\left(\boldsymbol{\delta}_s(\tau)\right)$ at the very first layer, for slice s and generic time τ), $*$ denotes the 3D convolution operator, $\mathbf{K}(t)$ is the kernel of filters, $\mathcal{H}(\cdot)$ is a non-linear activation function, and $\mathbf{b}$ is a bias vector. We use two different kernel configurations $\mathbf{K}(\tau)$, as shown in Fig. 4.3: a $3 \times 3 \times 3$ kernel for the first 3D-CNN layer, and a $6 \times 6 \times 6$ kernel for the second and third layers. These settings allow limiting the *receptive field*, *i.e.*, the portion of input analyzed by each neuron, to small regions: in presence of strong local correlation of the input data, this approach is known to yield good performance with fairly limited training, in particular compared to RNNs. As for the choice of the activation function, many different options have been proposed in the literature, spanning from linear functions to tanh, sigmoid or Rectified Linear Unit (ReLU). Among these, we select ReLU, and set $\mathcal{H}(\mathbf{x}) = \max(0, \mathbf{x})$, which provides advantages in terms of discriminating performance and faster learning [77]. Finally, $\mathbf{b}$ is randomly set at the beginning of each training phase.

The second and third 3D-CNN layers are interleaved with Dropout layers: such layers regularize the neural network and reduce overfitting [77] by randomly setting to zero a number of output features from the preceding layer during the training phase. The *dropout rate* defines the probability with which output features undergo this effect. During

[2]We have employed CNNs instead of Recurrent Neural Networks (RNNs) (typically used for forecasting application) because the mobile load at a time instant t mainly depends on previous T_p instants and not on all the past values (as confirmed by our analysis). For this reason, CNNs provide us enough temporal memory while being cheaper to train in terms of computational cost compared with RNNs

training, we employ two Dropout layers with dropout rate equal to 0.3.

The Decoder. It uses Multi-Layer Perceptrons (MLPs) [78], a kind of fully-connected neural layers, where every neuron of one layer is connected to every neuron of the next layer. This provides the ability to solve complex function approximation problems. In particular, MLPs are able to learn global patterns in their input feature space [47], allowing the neural network structure to forecast the targeted load value leveraging the local features extracted by the encoder. In our structure, each layer performs an operation $\mathcal{H}'(\mathbf{x} \cdot \mathbf{W} + \mathbf{b})$, where $\mathbf{x}$ is the MLP layer input vector, $\mathbf{W}$ a weight matrix related to the neurons of each layer, and $\mathbf{b}$ the bias vector. $\mathbf{W}$ plays a similar role to $\mathbf{K}(t)$ in the encoder part: its values drive the prediction through the layers of the decoding part.

As for the activation functions $\mathcal{H}$, we employ ReLU for all MLP layers except for the last one, where a linear activation function is used since the desired output takes real values. The last linear layer can be configured to produce multiple predictions in parallel, each matching the aggregate capacity required by a subset of base stations, thus allowing to forecast the needed capacity for different datacenters comprising a subset of base stations. Ultimately, this organization makes the DeepCog neural network capable of predicting per-slice capacity requirements at datacenter level, in a way that can adapt to any configuration of $\mathcal{M}$ and to any time horizon T_h.

4.3.3. The training procedure

We leverage the popular Adam optimizer, which is a Stochastic Gradient Descent (SGD) method that provides faster convergence compared to other techniques [79]. SGD trains the neural network model, evaluating at each iteration the loss function $\ell(\cdot)$ between the forecast and the ground truth, and tuning the model parameters in order to minimize $\ell(\cdot)$. For the configuration of the Adam optimizer, we use the default configuration with a learning rate of 5×10^{-4}.

An important element that concerns the training of the DeepCog architecture is that the encoder and the decoder described in Section 4.3.2 have independent roles. Therefore, while the decoder heavily depends on the forecast specifications, the encoder does not, and is agnostic to the final usage of the extracted features. This fact allows adopting a *transfer learning* approach during training: instead of treating the two blocks as a whole (and performing the training over the full system for all the possible slices, datacenter classes and horizons), we can train them separately. Specifically, an horizon-independent encoder can be trained on past traffic tensors at maximum time granularity, and then reused in combination with dedicated decoders tailored to each T_h value. Beside reducing the training time, this strategy reduces the need for neural-network-wide training to different settings of slice and datacenter only.

4.3.4. Arrangement of input data

The input is composed by measurement data generated in a specific network slice, and recorded by dedicated probes deployed within the network infrastructure. Depending on the type and location of the probe, the nature of the measurement data may vary, describing the demands in terms of, *e.g.*, signal quality, occupied resource blocks, bytes of traffic, or computational load on Virtual Network Functions (VNFs). DeepCog leverages a set of transformations to map any type of slice traffic measurements into a tensor format that can be processed by the learning algorithm.

The 3D-CNN layer adopted as the first stage of the decoder requires a multidimensional tensor input. We thus need to define the transformation $\mathcal{T}(\cdot)$ of each traffic snapshot into a matrix. Note that 3D-CNN layers best perform in presence of a tensor input that features a high level of local correlation, so that neurons operate on similar values. In image processing, where close-by pixels typically have high correlation, this is easily solved by treating the pixel grid as a matrix. In line with this strategy, the current common practice in mobile network traffic prediction is to leverage the geographical locations of the base stations, and assign them to the matrix elements so that their spatial proximity is preserved as much as possible [57, 60]. However, this approach does not consider that correlations in mobile service demands at a base station level do not depend on space, rather on land use [80]: base stations exhibiting strongly correlated network slice traffic may be far apart, *e.g.*, covering the different train stations within a same large city. Thus, we aim at creating a tensor input whose neighboring elements correspond to base stations with strongly correlated mobile service demands. To this end, we construct the mapping of base stations into a matrix structure as follows.

- For each base station i, we define its historical time series of total traffic as $\boldsymbol{\tau}^i = \{\delta^i(1), \ldots, \delta^i(t-1)\}$, where $\delta^i(t) = \sum_s \delta^i_s(t)$. Then, for each pair i and j, we determine the similarity of their recorded demands by computing $\text{SBD}^{ij} = f_{\text{SBD}}(\boldsymbol{\tau}^i, \boldsymbol{\tau}^j)$, where $f_{\text{SBD}}(\cdot)$ is the shape-based distance, a state-of-the-art similarity measure for time series [81]. All pairwise distances are then stored in a distance matrix $\mathbf{D} = (\text{SBD}^{ij}) \in \mathbb{R}^{M \times M}$.

- We compute virtual bidimensional coordinates $\mathbf{p}_i$ for each base station i so that the values in the distance matrix $\mathbf{D}$ are respected as much as possible. Formally, this maps to an optimization problem whose objective is $\min_{x_1,\ldots,x_M} \sum_{i<j} (\|\mathbf{p}_i - \mathbf{p}_j\| - \text{SBD}^{ij})^2$, efficiently solved via Multi-Dimensional Scaling (MDS) [82].

- We match each point $\mathbf{p}_i$ to an element e of the input matrix $\mathbf{I}$, again minimizing the total displacement. To this end, we: (i) quantize the virtual surface encompassing all points $\mathbf{p}_i$ so that it results into a regular grid of N cells; (ii) assume that each cell is an element of the input matrix; (iii) compute the cost k_{ie}

of assigning a point $\mathbf{p}_i$ to element e as the Euclidean distance between the point and the cell corresponding to e. We then formalize an assignment problem with objective $\min_a \sum_{i\in\mathcal{N}} \sum_{e\in\mathbf{I}} k_{ie} x_{ie}$, where $x_{ie} \in [0,1]$ is a decision variable that takes value 1 if point $\mathbf{p}_i$ is assigned to element e, and must fulfill $\sum_{i\in\mathcal{N}} x_{ie} = 1$ and $\sum_{e\in\mathbf{I}} x_{ie} = 1$. The problem is solved in polynomial time by the Hungarian algorithm [83].

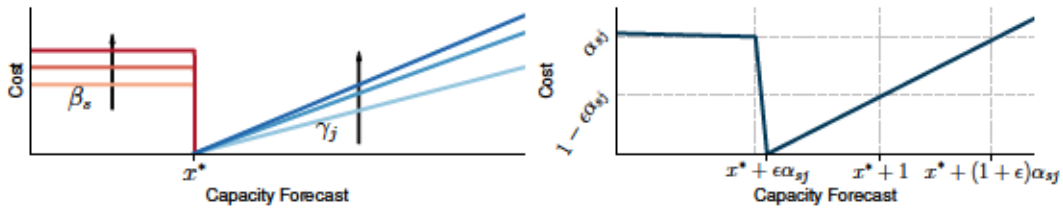
The solution of the assignment problem is the transformation $\mathcal{T}(\cdot)$ of the original base stations into elements of the matrix $\mathbf{I}$. The mapping function $\mathcal{T}(\cdot)$ allows translating a traffic snapshot $\boldsymbol{\delta}_s(t)$ into matricial form. Applying this to snapshots at different times, $\boldsymbol{\delta}_s(t-1), \ldots, \boldsymbol{\delta}_s(t-T)$, we can thus build the tensor required by the entry encoder layer in Fig. 4.3.

4.3.5. The Output function

DeepCog is designed for flexibility, and can be used for different orchestration scenarios. This is achieved thanks to an adaptable last layer of the deep neural network, and a configurable loss function. In general, the learning algorithm returns a forecast of the capacity required to accommodate the future demands for services associated to a specific network slice. This generic definition of output can then be applied to different orchestration use cases that may differ in the traffic aggregation level at which the resource configuration takes place, and/or in the frequency at which resource reallocation can be realized.

For instance, the anticipatory assignment of baseband processing units to network slices in a Cloud Radio Access Networks (C-RAN) datacenter requires a prediction of the capacity needed to accommodate the traffic of a few tens of base stations; instead, reserving memory resources for a specific network slice at a core network datacenter implies forecasting capacity for the data sessions of subscribers associated to hundreds of base stations. The output format of DeepCog can accommodate any datacenter layout, by tailoring the last linear layer of the neural networks to the specific requirements of the layout (as discussed in Section 4.3.2).

Also, as discussed previously, the time horizon over which the forecast is performed is another relevant system parameter, which depends on NFV technology limitations and current trends in *commoditization* of softwarized mobile network. When the technology limitations do not allow frequent reconfiguration opportunities, resources need to be allocated over long periods, e.g., of tens of minutes or even hours. In this case, forecasting over long-term horizons provides the operator with information on the constant capacity to be allocated during long intervals. To realize this, DeepCog operates on configurable time horizons, thanks to the flexible loss function that we will discuss next.



(a) Ideal Model (b) Actual Implementation

Figure 4.4: Cost model $\ell'(c_s^j(t) - d_s^j(t))$. Left: ideal model. Right: actual implementation in (4.1).

4.4. α-OMC

One of the key components of the system proposed in the previous section is the *loss function*, denoted by $\ell(\cdot)$. This function determines the penalty incurred when making a prediction error. In this paper, we propose a novel loss function that is tailored to the specific requirements of the capacity forecast problem. Our design of $\ell(\cdot)$ accounts for the costs resulting from (i) forecasting a lower value than the actual offered load, which leads to an *SLA violation* due to the provisioning of insufficient resources, (ii) predicting a higher value than the actual one, which leads to *overprovisioning*, allocating more resources than those needed to meet the demand. In order to ensure that we drive the system towards an optimal trade-off between overprovisioning and SLA violations, over a generic time horizon T_h, $\ell(\cdot)$ must account for the penalty inflicted in each case. In what follows, we describe the design of α-OMC (Operator Monetary Cost) [10], a loss functions that provides DeepCog with the capability of optimizing the overall running costs of the system.

4.4.1. Loss function design

In DeepCog, the loss function steers the behavior of the neural network by adjusting the weights of the neurons according to the error between the estimated value and the real one. To achieve the objective of minimizing the overall cost, a custom loss function for the capacity forecasting problem is composed by a term $f(x, x^*)$ that deals with the resource overprovisioning penalty, and a term $g(x, x^*)$ that models the cost of resource violations. The variable x represents the allocated resources at a given time interval, while x^* is the real demanded load for the same period. So the overall cost is due by the discrepancy between x and x^* in any time horizon.

The shape of overall cost function $f(x, x^*) + g(x, x^*)$ is depicted in Fig. 4.4a. A perfect algorithm (*i.e.*, an *oracle*) always keeps the system in the optimal operation point $x = x^*$ where no penalty is introduced, *i.e.*, $f(x^*, x^*) = g(x^*, x^*) = 0$. Of course, errors are inherent to predictions, and it is very unlikely that the forecast perfectly matches the real

demand: hence, a penalty value is back-propagated depending on whether x is above or below the target operation point x^*.

4.4.1.1. $g(x, x^*)$, a reactive approach to SLA violations.

When the orchestrated resources are less than those needed in reality (*i.e.*, $x < x^*$) the network operator pays a monetary compensation to the tenant. We assume an SLA that enforces a proportional compensation depending on the number of time intervals in which an operator fails to meet the requirements set by a tenant due to insufficient capacity allocated to the slice. Thus, SLA violations determine a fixed cost for the operator at every time interval where the tenant demand is not satisfied. Accordingly, we let the system learn that the operation point x^* is actually higher than the estimated one by applying a penalty β_s as soon as the estimation falls below the real value. The parameter β_s can be customized to the needs of the slice $s \in \mathcal{S}$: higher values may be used for cases where reliability is paramount like, *e.g.*, in Ultra Reliable Low Latency Communication (URLLC) network slices; instead, lower values can be applied for slices where Key Performance Indicator (KPI) commitments are provided over longer time intervals. Note that higher β_s values are likely to bring the system toward $x > x^*$, incurring hence in higher deployment costs, as discussed next.

4.4.1.2. $f(x, x^*)$, a monotonically increasing cost for resource overprovisioning.

While SLA violations depend on the agreements between the tenants and the operator, the overprovisioning cost solely depends on the network operator, and more specifically on the deployment costs associated with excess allocated capacity. We assume that such a cost grows with the amount of unused capacity at each time interval, and model it as a positive monotonic function that is only applied when $x > x^*$: the higher the resource provisioning error, the more (unnecessarily) expensive is the deployment. The exact expression of $f(x, x^*)$ may vary, and one could consider, *e.g.*, linear, super-linear, or exponential shapes. For DeepCog, we design α-OMC to use a linear function, as shown in Fig. 4.4a. The linear scaling factor γ_j is configurable by the operator, and represents the monetary cost of the excess resource allocation. The cost depends on the specific datacenter $j \in \mathcal{M}$ at which the capacity forecasting takes place: for instance, spectrum resources at the edge are typically scarcer and more expensive to deploy than computational resources in a network core datacenter. In case of expensive resources (characterized by a large γ_j), a positive forecasting error will have a higher impact, favoring a capacity forecast with a lower level of overprovisioning.

4.4.1.3. Balancing the two cost contributions.

Overall, the amount of resources that a network operator is willing to allocate depends on the cost that it has to pay when failing to meet the demands for a given slice (given by β_s) and the cost associated with adding extra resources at a specific datacenter (given by γ_j). These two parameters, β_s and γ_j, push the capacity allocations towards opposite directions, namely overdimensioning and underdimensioning, respectively. Rather than their absolute values, what really matter for the resulting allocation is the ratio between the two parameters, which determines the trade-off between overdimensioning and underdimensioning. Accordingly, in the following we express the custom loss as a function of a single parameter $\alpha_{sj} \doteq \frac{\beta_s}{\gamma_j}$. We remark that α_{sj} indicates the monetary costs of SLA violations with respect to the overprovisioning: failing to meet the slice requirements once costs as much as allocating α_{sj} units of excess capacity. Thus, a higher α_{sj} implies higher SLA violation costs relative to the deployment (*i.e.*, overprovisioning) cost. A mobile network operator can easily set this parameter based on its deployment costs, SLA fees, and market strategies.

Another important remark is that the SGD method used to train the neural network does not work with constant or step functions, and requires that the loss function be differentiable in all its domain. We solve this problem by introducing minimum slopes of very small intensity ϵ for $x < x^*$ and at $x = x^*$. We name the resulting loss function Operator Monetary Cost, which has a single configurable parameter α_{sj}. The final expression of α-OMC is

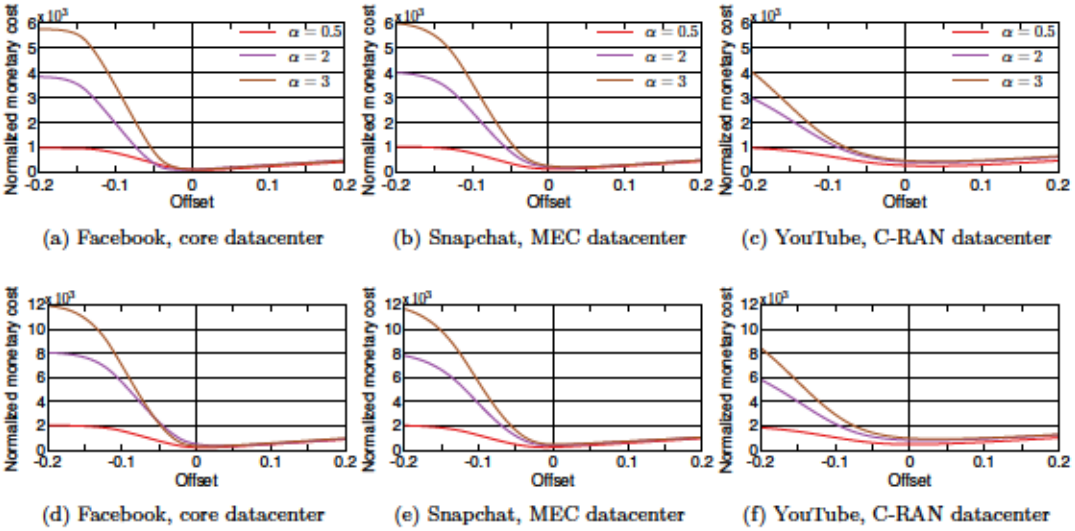
$$\alpha\text{-OMC}(x, x^*) = \begin{cases} \alpha_{sj} - \epsilon\,(x - x^*) & \text{if } x \leq x^* \\ \alpha_{sj} - \frac{1}{\epsilon}\,(x - x^*) & \text{if } x^* < x \leq x^* + \epsilon\alpha_{sj} \\ x - x^* - \epsilon\alpha_{sj} & \text{if } x > x^* + \epsilon\alpha_{sj}. \end{cases} \tag{4.1}$$

Fig. 4.4b provides a sample illustration of (4.1) above.

The final loss function $\ell(\cdot)$ then measures the quality of the forecast over the time horizon T_h, by applying the α-OMC expression over multiple time intervals as follows:

$$\begin{gathered} \ell\left(\mathbf{c}_s(t, T_h), \mathbf{d}_s(t), \ldots, \mathbf{d}_s(t + T_h)\right) = \\ \sum_{j \in \mathcal{M}} \sum_{\tau=0}^{T_h} \alpha\text{-OMC}\left(c_s^j(t, T_h), d_s^j(t + \tau)\right). \end{gathered} \tag{4.2}$$

For the sake of readability and without loss of generality, in the remainder of the paper we will employ a constant $\beta_s = \beta$ across slices and a constant $\gamma_j = \gamma$ across datacenter deployment, leading to $\alpha_{sj} = \alpha$ for all $s \in \mathcal{S}, j \in \mathcal{M}$.



(a) Facebook, core datacenter (b) Snapchat, MEC datacenter (c) YouTube, C-RAN datacenter

(d) Facebook, core datacenter (e) Snapchat, MEC datacenter (f) YouTube, C-RAN datacenter

Figure 4.5: Monetary cost (aggregated over time and normalized by the cost of one capacity unit) incurred when the overprovisioning level is shifted from that selected by DeepCog (at the abscissa origin). Each plot refers to one case study, *i.e.*, a combination of (*i*) mobile service associated to a dedicated slice and (*ii*) datacenter type. Top row: $T_h = 5$ minutes, bottom row: $T_h = 30$ minutes.

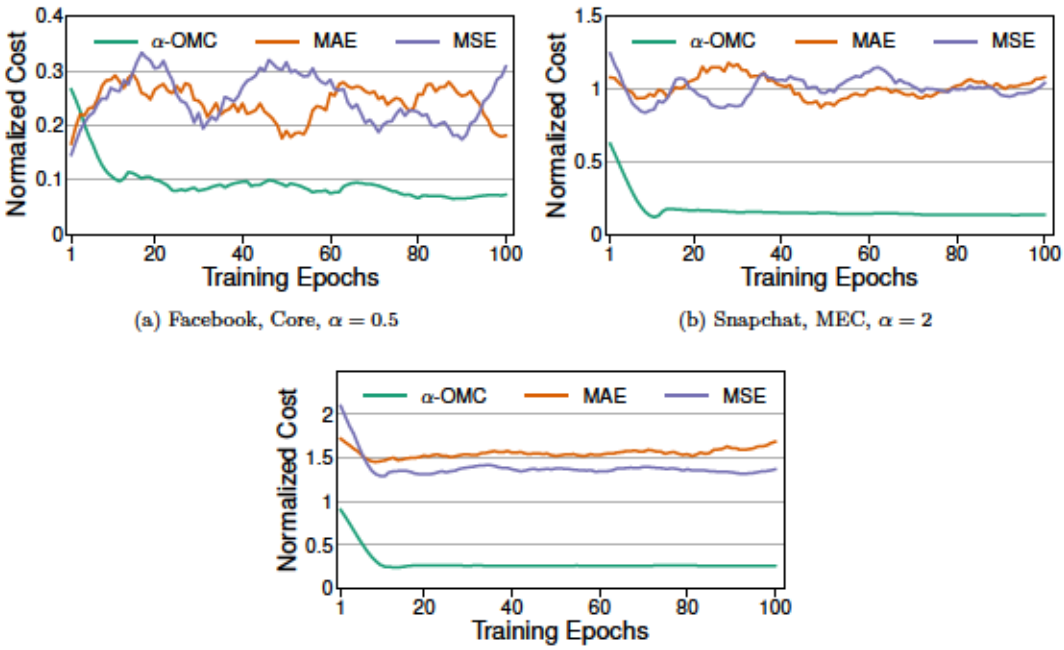
4.4.2. Correctness and convergence

We now analyze the proposed loss function in terms of (*i*) correctness, *i.e.*, its capability of achieving a performance that is close to the optimal, and (*ii*) convergence, *i.e.*, the time it requires to learn such a correct strategy.

In Fig. 4.5, we run DeepCog in the representative network resource management case studies that are later detailed in Section 4.5, where a slice is dedicated to one particular mobile service and runs in a specific class of network datacenter. For each case study, DeepCog forecasts a given level of capacity to be allocated which leads to an associated monetary cost. In order to investigate the correctness of the solution, we vary the provisioned capacity by adding to or subtracting a fixed offset from the capacity indicated by DeepCog.

The curves of Fig. 4.5 illustrate the variation of the monetary cost (in the y axis) as the offset is shifted (in the x axis), where increasingly positive (respectively, negative) values on the x axis correspond to a higher (respectively, lower) level of capacity provisioning with respect that suggested by our solution. The results prove that DeepCog always identifies the capacity allocation that minimizes the monetary cost for the operator under the inherently inaccurate prediction, as both a higher and a lower level of overprovisioning leads to a greater cost. This holds under any combination.[3] of target mobile service,

[3]Fig. 4.5 shows results for α in the range $[0.5, 3]$, and two exemplary T_h values, 5 and 30 minutes.



(a) Facebook, Core, $\alpha = 0.5$

(b) Snapchat, MEC, $\alpha = 2$

(c) YouTube, C-RAN, $\alpha = 3$

Figure 4.6: Average cost versus the learning epochs, when the DeepCog neural network architecture is trained with α-OMC, MSE and MAE loss functions.

datacenter class, and system settings α or T_h, which demonstrates the high consistency of our solution in balancing costs caused by SLA violations and overprovisioning.

We next assess the convergence properties of the loss function that drives DeepCog, by observing its behavior over time. Specifically, we measure the normalized cost of the solution identified by our learning algorithm, and compare it against that returned by the same neural network trained with legacy loss functions.

Fig. 4.6 shows how the average normalized cost of network operation varies during the training phase for different α, services and datacenter classes. While the α-OMC loss function minimizes the monetary cost of the operator in less than 20 epochs, both Mean Absolute Error (MAE) and Mean Squared Error (MSE) converge to a fixed fee that grows as α increases. This confirms that classical loss functions are not effective when dealing with capacity forecasting, resulting in high penalties for operators. The results are consistent across all of the different configuration scenarios we tested.

4.5. Performance Evaluation

In this section we evaluate DeepCog performance in realistic settings. To this end, we consider the mobile network infrastructure of a major operator in a large metropolitan

Similar curves characterize all α values and prediction horizons (up to 8 hours) we tested.

Table 4.1: Mobile services retained for dedicate network slices.

Service name	*Service class*	*Traffic %*	*Service name*	*Service class*	*Traffic %*
YouTube	streaming	27.3	iTunes	streaming	20.0
Netflix	streaming	1.8	Facebook	social media	20.4
Instagram	social media	3.4	Twitter	social media	3.2
Snapchat	messaging	8.9	Google Play	online store	4.3
Apple Store	online store	10.5	Pokemon Go	mobile gaming	0.1

region. The area under study covers around 100 km^2 with a resident population of more than 2 millions, and is surrounded by a conurbation of 11 millions inhabitants who often commute to it. We run DeepCog on real-world measurement data of an operator with a market share of 35% in the target region that captures the traffic generated by millions of users. The data were collected by monitoring the GPRS Tunneling Protocol (GTP) via dedicated probes deployed at the network gateway and the classification of IP flows into services was performed via Deep Packet Inspection (DPI) with proprietary models developed by the network operator. The traffic demands, expressed in bytes, refer to individual mobile services; they are aggregated at the antenna sector level and over intervals of 5 minutes. The demands capture the highly heterogeneous and time-varying loads that characterize real-world mobile network deployments, with differences in the offered traffic volume of up to two orders of magnitude between antenna sectors. Independent network slices are then assigned to a representative set of services, listed in Tab. 4.1

We consider services that belong to different categories, including video streaming, messaging and social networks. These services impose a broad range of requirements into the network. For instance, video streaming services are consumed ubiquitously, they have significant bandwidth requirements (up to 6 Mbps for a 1080p video in YouTube [84]) as well as latency constraints to avoid interruptions in playing of the video (as shown in [85]), and they also require significant data center resources from the server side [84]. Messaging services, instead, have a large component of uplink traffic; overall, it has a rather relaxed requirements in terms of bandwidth and latency, as it does not involve any interactive communications. Social networks need considerable bandwidth and with some latency requirements (according to [86], bandwidth of 8 Mbps and access delay not exceeding 100 ms are required to achieve a high overall quality). Beyond the specific requirements of each service, what really matters in the context of this paper is the fact that these are services of a very different nature with highly diverse requirements, and therefore they are likely to be served by different slices in a mobile network supporting network slicing.

We employ the measurement data to design three case studies combining several

popular mobile services and different classes of network datacenters[4]. Each class is defined by the network location and number of served eNodeBs, ranging from centralized datacenters located in the core and serving many eNodeBs to more distributed ones located in the edge and serving a smaller number of eNodeBs. By selecting a diverse set of case studies, we can assess the DeepCog flexibility serving heterogeneous NFV scenarios, comprising different services and datacenter classes (C-RAN, Mobile Edge Computing (MEC) and core). In a first case study, we consider that a slice is instantiated for the incumbent video streaming service, *i.e.*, YouTube, at C-RAN datacenters in the target metropolitan area, each located in proximity of the radio access and performing baseband processing and scheduling for around ten eNodeBs. In the second case study, we look into MEC datacenters that handle the traffic of around 70 eNodeBs each, where a dedicated slice accommodates the traffic generated by Snapchat, a favored messaging app. The third case study focuses on a network slice dedicated to social network services provided by Facebook that are run at a core network datacenter controlling all 470 4G eNodeBs in the target metropolitan area.

The three case studies cover applications with diverse requirements in terms of bandwidth and latency; also, they entail very different spatiotemporal dynamics of the mobile traffic, as the considered services feature different loads and activity peaks [88]. In addition, the datacenter classes we consider have dissimilar geographical coverage and aggregated traffic volumes, as they serve the demands associated to a variable number of antennas, from ten to several hundreds. Overall, the three case studies considered for the DeepCog's performance evaluation are very useful to understand the effect of network slicing on the network operation costs. In fact, our results illustrate for the first time the impact of the slice isolation requirements –critical to future softwarized networks– on services that have a dominant role in today's traffic and are expected to keep playing a very relevant role in future mobile networks. In our tests, we do not parametrize different excess resource costs for each datacenter nor different SLA violation penalties for diverse services, leading to homogeneous α settings (*i.e.*, $\alpha_{sj} = \alpha$ for all $s \in \mathcal{S}, j \in \mathcal{M}$). However, since DeepCog provisions resources for each slice and each datacenter independently, by evaluating different α values our results provide insights on the behavior of heterogeneous α settings as well.

As discussed in Section 4.3.5, DeepCog outputs a capacity forecast within a variable time-horizon T_h. We measure this time in the number of steps it comprises, where each step corresponds to the 5 mins granularity of our measurement data. In our evaluation T_h ranges from 5 minutes (which maps to a next-step prediction) to 8 hours (which corresponds to a forecast with a 96 time steps look-ahead). These are reasonable values in our context, since resource reallocation updates in the order of minutes are typical

[4]The internal organization of the mobile network – hence the demand recorded at each datacenter – is inferred by adopting the methodology proposed in [87].

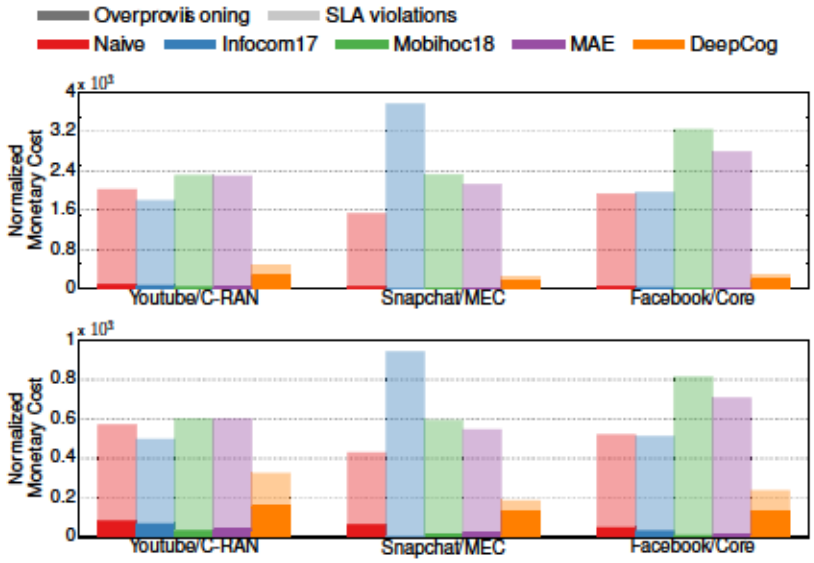


Figure 4.7: Comparative evaluation of DeepCog with four benchmarks in three representative case studies. The monetary cost (normalized by the cost of one capacity unit) incurred by the operator is split into costs due to overprovisioning (dark) and SLA violations (light). Left: $\alpha = 2$. Right: $\alpha = 0.5$.

for computational and memory resources in architectures implementing NFV [2], and are in line with those supported by any state-of-the-art Virtual Infrastructure Manager (VIM) [89]. Conversely, larger intervals are more suitable for operations involving manual intervention, *e.g.*, spectrum leasing.

In all cases, we use the previous 30 minutes of traffic (*i.e.*, $T_p = 6$) as the DeepCog input, arranged in a 47×10 matrix. This configuration proved to yield the best results when confronted to a number of other design strategies for the input that we explored, including longer, shorter, or non-continuous historical data time intervals. Capacity is predicted in terms of bytes of traffic, which is a reasonable metric to capture for resource utilization in actual virtual network functions [90], and is independent of the exact type of resources relevant for the mobile operator in each case study. We employ two months of mobile traffic data for training, two weeks of data for validation and another two for the actual experiments. This setting is also used for all benchmark approaches. All results are derived with a high level of confidence and low standard deviation.

4.5.1. Gain over state-of-the-art traffic predictors

We first focus on the particular case of next-step prediction, *i.e.*, $T_h = 5$ minutes, as this benchmark lets us compare our framework against state-of-the-art solutions that can only perform a forecast for the following time interval. As discussed before, DeepCog is designed as a building block within a network resource orchestration framework. A

fundamental advantage over existing solutions in the literature is that it targets capacity forecast, avoiding SLA violations, rather than a mere prediction of traffic load which may incur into frequent violations. We compare DeepCog against four benchmarks: (*i*) a `naive` technique that forecasts the future offered load by replicating the demand recorded at the same time during the previous week; (*ii*) the first approach proposed to predict mobile traffic based on a deep learning structure, referred to as `Infocom17` [57]; (*iii*) a recent solution for mobile network demand prediction that leverages a more complex deep neural network, referred to as `MobiHoc18` [72]; (*iv*) a reduced version of DeepCog, which replaces α-OMC with a legacy MAE loss function[5].

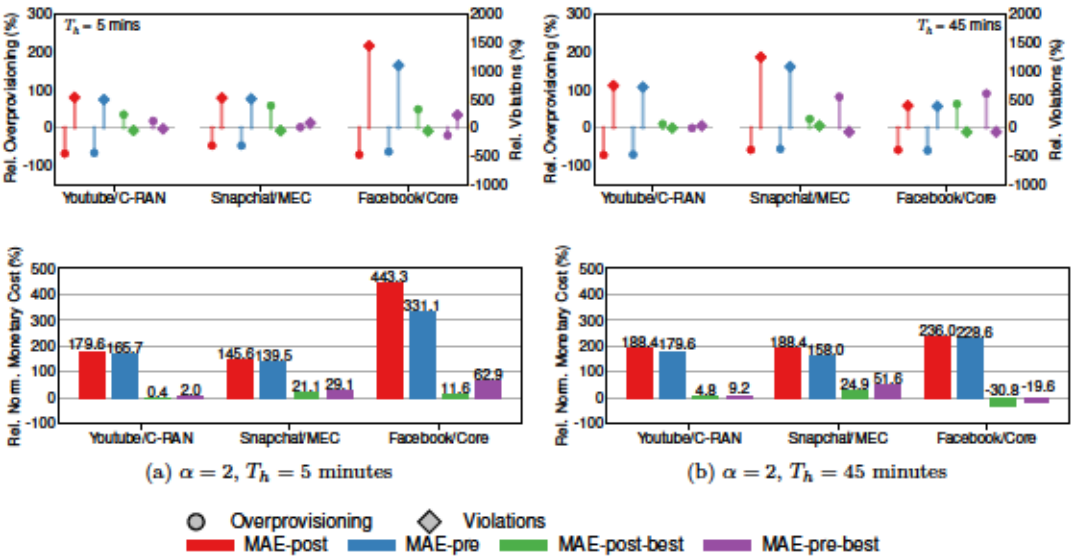
The results achieved in our three reference case studies by DeepCog and by the four benchmarks above are shown in Fig. 4.7. The plots report the normalized monetary cost for the operator, broken down into the expenses for unnecessary resource allocation (*i.e.*, overprovisioning) and fees for unserviced demands (*i.e.*, SLA violations). We observe that DeepCog yields substantially lower costs than all other solutions. Indeed, the cost incurred by DeepCog for $\alpha = 2$ ranges between 15% (Facebook/Core) and 27% (Youtube/C-RAN) of the cost provided by the *best* competitor, depending on the case study. `Infocom17`, as all other benchmarks, targets mobile network traffic prediction, whereas DeepCog aims at forecasting capacity. As a result, DeepCog balances overprovisioning and SLA violations so as to minimize operation expenses, while `Infocom17` is oblivious to such practical resource management considerations. In other words, legacy predictors follow as closely as possible the general trend of the time series and allocate resources based on their prediction, which leads to systematic SLA violations that are not acceptable from a market viewpoint and determine huge fees for the operator. Instead, DeepCog selects the appropriate level of overprovisioning that, by suitably overestimating the offered load, minimizes monetary penalties (see Fig. 4.5). Indeed, even when choosing a low value such as $\alpha = 0.5$, which inflicts a small penalty for a SLA violation, the cost incurred by DeepCog is 64% of that incurred by the best performing benchmark.

4.5.2. Comparison with overprovisioned traffic prediction

In the light of the above results, a more reasonable approach to resource allocation could be to consider a traditional mobile traffic prediction as a basis, and adding some *overprovisioning offset* on top of it. In order to explore the effectiveness of such an approach, we design and implement several variants to `MAE`, as follows.

A first variant adds an *a-posteriori* constant overprovisioning offset to the `MAE` output. This strategy, referred to as `MAE-post`, requires selecting a value of the static offset, which is then added to the predicted traffic. We dimension the offset as a certain percentage of the peak traffic activity observed in the whole historical data, and set it at 5%, which

[5]We also experimented with other popular loss functions, *e.g.*, MSE, with comparable results, omitted for space reasons.



(a) $\alpha = 2$, $T_h = 5$ minutes (b) $\alpha = 2$, $T_h = 45$ minutes

Figure 4.8: Relative performance of overprovisioned traffic predictors, expressed as a percent of the cost attained by DeepCog. Top: relative overprovisioning and SLA violations. Bottom: relative monetary cost. Results refer to $\alpha = 2$ and prediction horizons of 5 (left) and 45 (right) minutes.

we deem a reasonable value in presence of a decently accurate prediction. Alternatively, we also consider a best-case version of this solution, named `MAE-post-best`, where an a-posteriori overprovisioning is chosen by performing an exhaustive search over all possible offset values and selecting the one that minimizes the loss function $\ell(\cdot)$.

A second variant accounts for some level of overprovisioning in a *preemptive* fashion, by introducing the offset during the deep neural network training. To this end, the `MAE-pre` solution replaces the MAE loss function with a new loss function $\mathcal{O} + \frac{1}{\mathcal{M}} \sum_{j \in \mathcal{M}} |c_s^j(t) - d_s^j(t)|$, where $\mathcal{O}$ denotes the a-priori overprovisioning offset. Also in this case, we set $\mathcal{O}$ equal to 5% of the peak traffic in the historical data. To compare against the best possible operation of this scheme, we also consider a `MAE-pre-best` variant where $\mathcal{O}$ is set equal to the average overprovisioning level provided by DeepCog for the test period.

We remark that the `MAE-post-best` and `MAE-pre-best` approaches are oracles and not feasible in practice, since they require knowledge of the future to determine the best a-posteriori values for the offset and the value of $\mathcal{O}$, respectively. Yet, they provide a benchmark for comparing the performance of DeepCog against optimal solutions that rely on traditional mobile network traffic prediction.

Fig. 4.8 shows the relative performance of the four variants above with respect to that attained by DeepCog, for $T_h = 5$ min (left) and $T_h = 45$ min (right). The figure shows the oveprovisioned capacity, unserviced traffic, and total economic cost incurred by the operator relative to the performance offered by DeepCog (in percentage). For $T_h = 5$ min, the results highlight how using a static overprovisioning in combination with a traditional

traffic prediction is largely suboptimal, both when the additional offset is considered preemptively or a-posteriori. Indeed, the two practical solutions considered, *i.e.*, `MAE-post` and `MAE-pre`, cause SLA violations that are two- to three-fold more frequent than that incurred into by DeepCog, resulting in an economic cost that is 140% to 400% higher. Interestingly, even when parametrized with the best possible offsets, the approaches based on legacy traffic prediction cannot match the performance of DeepCog: `MAE-post-best` and `MAE-pre-best` dramatically reduce the penalties of their viable counterparts, yet lead to monetary costs that are up to 60% higher than those of DeepCog.

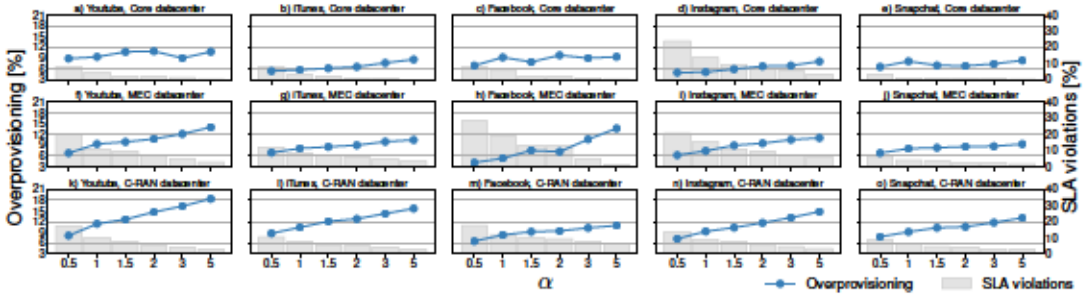
The results for $T_h = 45$ min[6], show that the above considerations hold across different values of the prediction horizon.The advantage over feasible overprovisioned traffic predictors such as `MAE-pre` or `MAE-post` is aligned with that observed under a next-step prediction, as such solutions increase the overall cost by 188% to 236%. When considering a long horizon of 45 minutes, oracle methods based on overprovisioning like (*i.e.*, `MAE-pre-best` and `MAE-post-best`) can outperform DeepCog, further reducing the operator cost by 19% to 30%. This is due to the fact that, when prediction must be performed with significant time advance, the accuracy of DeepCog cannot be as high as an oracle that knows future demand and has hence a significant advantage. However, even under such conditions DeepCog performs almost as good as the oracles or better.

We conclude that traffic predictors – no matter how they are enhanced – are not appropriate for the capacity forecast problem, for the simple reason that they are designed for a different purpose. Indeed, they ignore the economic penalties incurred by SLA violations, and this limits drastically their ability to address this problem. Strategies that rely on integrating such costs into the solution after the traffic prediction is performed are largely suboptimal.

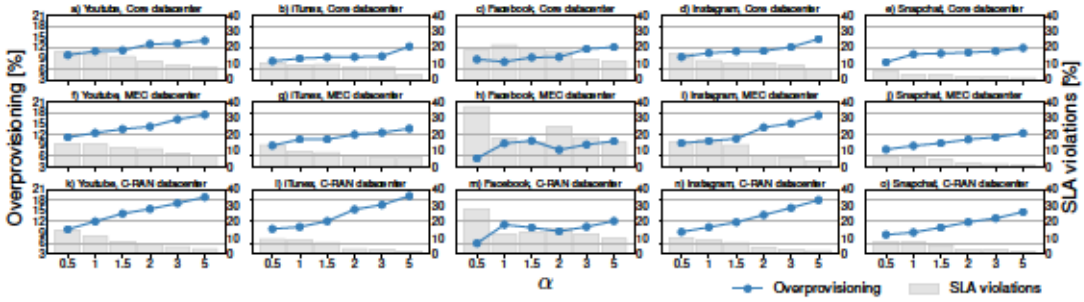
4.5.3. Controlling resource allocation trade-offs with α

As discussed in Section 4.4, DeepCog addresses a fundamental trade-off between overprovisioning and SLA violations, aiming to find the best possible compromise between the two. An operator is given the flexibility of choosing the desired operation point within this trade-off, by suitably setting the α parameter. In the following, we carry out an extensive analysis of the trade-off between overprovisioning of resources and failing to meet service demands. This study is conducted for a large number of practical scenarios that extend the original three case studies considered in the comparative analysis. Specifically, we select five different network slices, dedicated to the same number of popular mobile services: the three we already studied, *i.e.*, YouTube, Facebook, and Snapchat, plus iTunes and Instagram. We then investigate the performance of DeepCog when such slices are deployed at the three classes of datacenters introduced before, *i.e.*, at the C-RAN,

[6]Note that, in order to perform a fair comparison, we had to extended the MAE policy to compute the average absolute error on each time slot in the $[t, \ldots, t + T_h]$ interval.



(a) $T_h = 5$ minutes



(b) $T_h = 120$ minutes

Figure 4.9: Tradeoff between resource overprovisioning (expressed as a percentage of the actual demand) and SLA violation (expressed as a percentage of time slots), as a function of the α parameter. Results refer to 15 different scenarios, and two values of the prediction horizon T_h, *i.e.*, 5 minutes (a) and 120 minutes (b).

MEC, and network core. Overall, this leads to 15 distinct scenarios.

Fig. 4.9a shows results in all of the above settings under different economic strategies that are reflected by the α parameter of the loss function $\ell(\cdot)$. Configurations range from policies that prioritize minimizing overprovisioning over avoiding SLA violations ($\alpha = 0.5$) to others that strictly enforce the SLAs at the price of allocating additional resources ($\alpha = 5$). The plots tell apart the contribution of the two components that contribute to the total monetary cost: overprovisioning is expressed as a percentage of the actual demand, and SLA violations are measured as a percentage of the time slots in the test period. As expected, higher α values reduce the number SLA violations, as they become increasingly expensive; this occurs at the cost of provisioning additional capacity, which becomes instead cheaper in proportion[7]. The trend is consistent across all scenarios, confirming that α effectively drives resource orchestration towards the desired operation point.

Our analysis also reveals that the level of overprovisioning grows in most cases as one moves from datacenters in the network core outwards. This trend applies across slices,

[7]Sporadic disruptions in the monotonicity of the cost curves are due to the inherent randomness of the measurement data; indeed, the data correspond to a specific time period and it may show some biases that would not be observed over different (or longer) time periods.

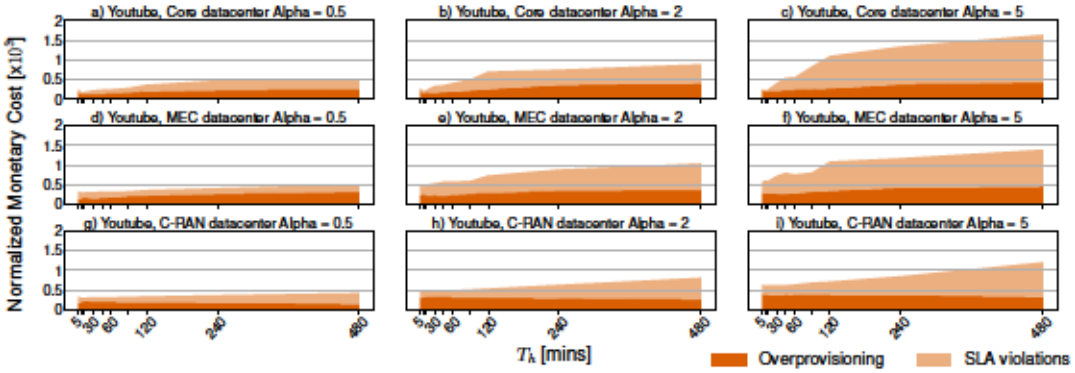


Figure 4.10: Monetary cost (normalized by the cost of one capacity unit) incurred by the operator, versus the prediction time horizon T_h. The plots refer to different combinations of datacenter class and economic strategies modeled by α, for a slice dedicated to the YouTube mobile service.

and is due to the fact that more centralized datacenters serve an increasingly aggregate traffic that is generally less noisy and easier to predict.

Under such conditions, DeepCog needs a reduced additional capacity to limit unserviced demands: as a result, SLA violations are often lower at core datacenters.

Fig. 4.9a refers to a short-term prediction for $T_h = 5$ minutes, however the same trends discussed above are confirmed for larger prediction horizons. For instance, Fig. 4.9b reports the same results for $T_h = 120$ minutes. The only remarkable difference is that overprovisioning and SLA violations are higher than in the case of a 5-minute prediction, as forecasting on larger time horizons is obviously harder. Yet, the impact of α is equivalent to that observed for $T_h = 5$ minutes. We analyze in more detail DeepCog's performance as a function of T_h in the next section.

Overall, the results presented above show that DeepCog finds good trade-offs between resource overprovisioning and SLA violations in very different cost settings across slices and datacenter types. Since each DeepCog instance for a slice at a datacenter runs independently, this shows that DeepCog will grant good performance also in scenarios where resource costs may differ across datacenters, *e.g.*, due to diverse operation and management costs in urban and rural facilities.

4.5.4. Long-term capacity prediction with DeepCog

DeepCog aims at forecasting the (constant) capacity that should be allocated over a long-term horizon, so as to minimize the monetary cost incurred by the operator. As discussed in Section 4.3.1, this is particularly useful in practical settings where the NFV technology imposes limits on the frequency upon which resources can be reallocated. In the following, we thoroughly study how the performance of DeepCog varies with the prediction horizon.

Fig. 4.10 summarizes the overall trend of the monetary cost incurred by DeepCog, as the periodicity of the reconfiguration opportunities ranges from 5 minutes to 8 hours. The plots outline a diversity of scenarios, combining different datacenter classes (C-RAN, MEC, and core) and relative expenses of overprovisioning and SLA violations (α equal to 0.5, 2, and 5). The results correspond to the case where one slice is dedicated to the traffic generated by YouTube, but equivalent behaviors were observed for the other services. In all settings, the cost grows with the prediction horizon, which, as already mentioned, is largely expected. What is less expected, however, is the quasi-linear relationship between the cost and T_h. This is a very important result, as it shows that even if we increase the intervals for resource reallocation (*i.e.*, the time horizon), the economic expenses of the operator remain bounded and do not skyrocket (as they would if the growth was, *e.g.*, exponential). The result thus demonstrates the efficiency of DeepCog in limiting the unavoidable increased penalty associated to forecasting long-term capacity: as an indicative figure, the cost is roughly increased by two when moving from a 5-minute prediction to one that spans the following 8 hours which is a very reasonable factor.

The impact of the other system parameters is in line with our previous analysis: higher monetary fees for SLA violations (*i.e.*, higher α values) lead to increased costs, whereas the performance is comparable across resource allocations over different classes of datacenter (C-RAN, MEC and core), each corresponding to different traffic volumes. It is nonetheless interesting to note that the property of a linear growth of the cost over T_h is preserved under any combination of such parameters.

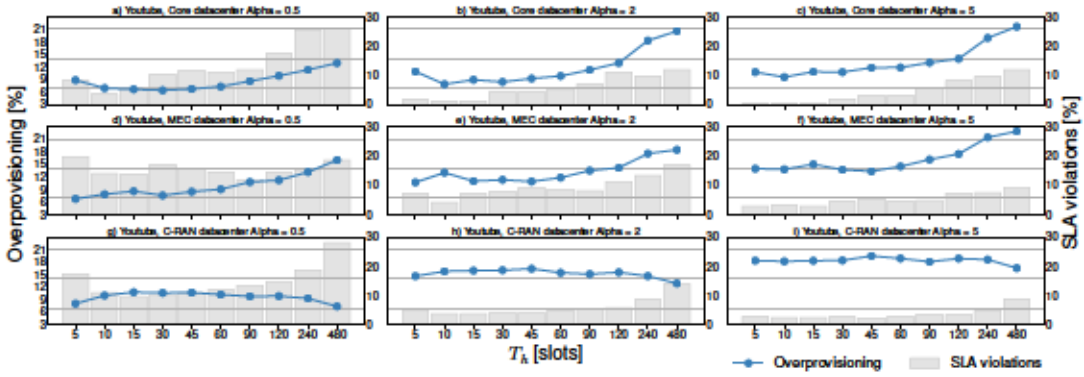
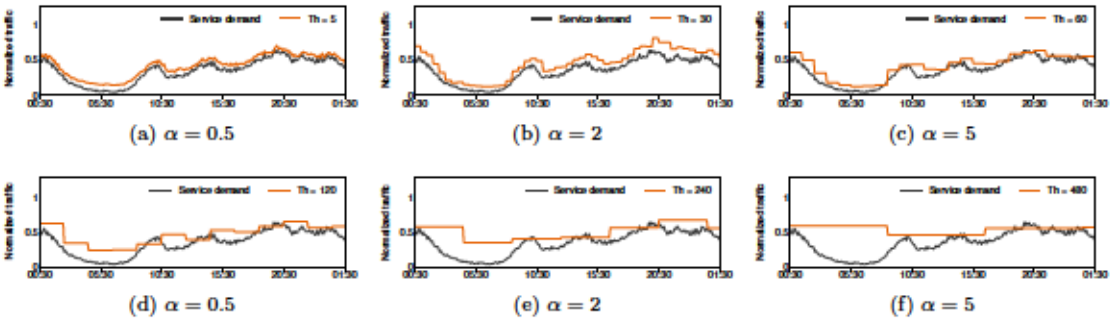


Figure 4.11: Breakdown of monetary costs into two contributions: (*i*) overprovisioning (expressed as a percentage of the actual demand) and (*ii*) SLA violations (expressed as a percentage of time slots), in the scenarios of Fig. 4.10.

Fig. 4.10 also offers a breakdown of the overall monetary costs into the two contributions (overprovisioning and SLA violations). Violations of SLAs yield substantially higher absolute costs and dominate the increase of total cost with T_h; the effect is clearly stronger for higher values of α. A more detailed view that highlights the exact evolution of the two cost components as a function of T_h is provided in Fig. 4.11,



(a) $\alpha = 0.5$ (b) $\alpha = 2$ (c) $\alpha = 5$

(d) $\alpha = 0.5$ (e) $\alpha = 2$ (f) $\alpha = 5$

Figure 4.12: Illustrative examples of the capacity forecast returned by DeepCog behavior under different prediction time horizons. he scenario refers to a network slice dedicated to the YouTube mobile service that is deployed at a core datacenter, under $\alpha = 2$.

showing that both contribute to increasing costs over longer-term forecasts. However, and interestingly, the dynamics of the two components with T_h are diverse depending on the system settings such as the datacenter class and the value of α. The common trend here is that the penalty associated with both overprovisioning and SLA violations is fairly stable when the horizon is increased from 5 minutes up to two hours. For forecasts beyond two hours, however, these fees (one of the two or both) tend to increase substantially with T_h.

We ascribe these behaviors to (i) the relationship between T_h and the timescale of temporal fluctuations in the input demand, and (ii) the way DeepCog reacts to the problem of devising a capacity forecast, which becomes harder for larger T_h values. The first point relates to the characteristics of the input data (see Fig. 4.12 for illustrative examples of the temporal oscillation of the service demand). For very large T_h (above 120 minutes) the prediction task performed by DeepCog resorts to an "envelope" of the demand that accommodates the peak over the T_h period. This means that for those times where demand is below the peak, we incur a high level of overprovisioning that increases the resulting cost. In contrast, smaller T_h allow to adapt the capacity forecasting to the actual demand at each point in time, providing an advantage in terms of cost. The second point relates to the behavior and performance of DeepCog under large T_h values. DeepCog aims at providing a similar level of overprovisioning over time, as exemplified by the top three plots of Fig. 4.12. For large T_h values this yields increased SLA violations, since the oscillations make it more likely that the constant capacity falls below the demand curve at some point during T_h. Additionally, larger T_h values make the prediction task inherently harder, which further contributes to increasing the SLA violations costs.

5 Network Slicing with shared resources providing hard guarantees

In Chapter 4 we have investigated the impact of Network Slicing in next mobile network generation resource orchestration. To cope with the need for proactive, data-driven, automated solutions that enable cost-efficient network resource orchestration, we have designed DeepCog, a Deep Learning solution for the anticipatory allocation of resources in cognitive mobile networks. Running on traffic measurements, we have shown how DeepCog is efficient in providing operators with information about the capacity needed to accommodate future demands at each network slice. We demonstrated that leveraging on the information provided by DeepCog, a Infrastructure Provider (InP) is able to reduce his monetary cost of more than 200% compared with state of the art traffic predictor algorithms.

DeepCog represents to the best of our knowledge the only work to date where a deep learning architecture is explicitly tailored to the problem of anticipatory resource orchestration in mobile networks, it takes into account the costs derived by (*i*) the allocation of unnecessary resources that go unused (*i.e.*, overprovisioning), and (*ii*) the insufficient provisioning of resources that lead to Service Level Agreement (SLA) violations. Hence, DeepCog has been designed to aim at minimizing overprovisioning while avoiding SLA violations.

However, limiting the problem to this simple trade-off implicitly assumes that resource instantiation and reconfiguration occurs at no cost. In the following sections, we propose an original model for the anticipatory allocation of capacity to network slices, which is mindful of all operating costs linked to (*i*) unnecessary resource overprovisioning, (*ii*) non-serviced demands, (*iii*) resource instantiation, and (*iv*) resource reconfiguration. In Section 5.1 we first motivate the need to include in our analysis the costs deriving by resource instantiation and reconfiguration. In Section 5.2 we present our orchestration model, formalizing the different costs and trade-offs in the resource management of sliced networks. Building on such a new model, we develop a complete framework for the anticipatory allocation of capacity to network slices, named `AZTEC` [12]; the framework relies on a combination of deep learning architectures and a traditional optimizer, as

detailed in Section 5.3, 5.4 and 5.5. When informed of the economic penalty associated to each source of cost, AZTEC anticipates the dedicated and shared capacity to be allotted to each network slice in a way to cut down monetary losses due to instantiation and reconfiguration, while keeping fees entailed by resource provisioning and non-serviced demands under control. We demonstrate the quality of the solution with real-world measurement data collected in a metropolitan-scale mobile network, in Section 5.6; in typical settings, AZTEC outperforms state-of-the-art traffic [57] and capacity [9] predictors by at least a factor of 1.7.

5.1. Capacity forecasting for resource management

Network slicing enables the desired strong service differentiation by capitalizing on recent developments in Network Function Virtualization (NFV). It creates multiple logical instances of the physical network, the so-called *network slices*, ensuring strict traffic isolation among them [3], and tailoring the network resources of each slice to a specific (class of) application [91]. Network slicing has the potential to enable the coexistence of a wide range of mobile services in the same network infrastructure; however, it also poses several of technical challenges.

A prominent difficulty is resource management. By running dedicated Artificial Intelligence solutions, network orchestrators are expected to enable the vision of *zero-touch networks* [92], *i.e.*, fully self-operating communication infrastructures whose standardization is currently under consideration [93].

For some types of resource (*e.g.*, CPU time within the same bare metal machine) resource instantiation and reconfiguration represent a negligible cost. This is not generally valid for slice resource management scenarios. Instantiation and reconfiguration costs are capital in NFV technologies that enable the *cloudification* of the access and core networks by entrusting many network functions to Virtual Machines (VMs) running in datacenters. Examples include baseband processing in Cloud Radio Access Networks (C-RAN) [94], interconnection functionalities towards the external packet networks through the User Plane Function (UPF) [95], or central office operations [96].

In all the above cases, resource instantiation is not for free: VM boot times in prominent public cloud services like Amazon AWS or Microsoft Azure consistently exceed 40 seconds, topping at 400 seconds in worst-case scenarios [97]; even in very recent tests, booting a lightweight VM containing an Alpine Linux takes around 30 seconds in a local deployment [98]. Reconfiguring already allocated resources has also a non-negligible cost: modern software architectures such as Kubernetes need several seconds to execute new pods, *e.g.*, on VMs that are already running [98]. In addition, re-orchestration often implies recomputing paths on the transport networks and implementing them via, *e.g.*, Software Defined Networking (SDN) architectures: the latency is in the order of hundreds

of milliseconds in a small five-switch topology and with precomputed routing [99], and this figure has to be scaled to thousands of switches with on-the-fly path re-calculation.

All unavoidable delays above entail monetary fees for the operator, in terms of both violations of the Service Level Agreement (SLA) with the tenants (*e.g.*, due to infringement of guarantees on end-to-end latency), and user dissatisfaction (with ensuing high churn rates). By neglecting these sources of cost, present capacity forecast solutions risk to introduce uncontrolled data flow latency once deployed in operational networks, ultimately causing economic losses to the operator.

In the following section, we propose an original model for the anticipatory allocation of capacity to network slices, which is mindful of all operating costs linked to (*i*) unnecessary resource overprovisioning, (*ii*) non-serviced demands, (*iii*) resource instantiation, and (*iv*) resource reconfiguration. Our approach is based on the concept of *multi-timescale orchestration* illustrated in Figure 5.1.

- On the left, Deepcog, the algorithm introduced in Section 4.3, tries to accommodate the demand and to limit overprovisioning, by reconfiguring resources at every re-orchestration opportunity at disposal (top plot); by doing so, it minimizes costs (*i*) and (*ii*) above (second plot). However, it also ceaselessly instantiates or de-commissions capacity, and reallocates available resources in a sustained way. This incurs in substantial instantiation and reconfiguration fees (third plot) that ultimately lead to a high overall economic cost (bottom plot).

- On the right, our model performs the orchestration at two timescales, and by telling apart two classes of resources. A *long-timescale orchestrator* operates over extended intervals that span multiple re-orchestration opportunities; it allocates a *dedicated capacity* to each slice and also reserves an additional *shared capacity* accessible by any slice. Both capacities remain constant across the extended interval, limiting the frequency of instantiation and thus cost (*iii*). Only the shared capacity is then reallocated at every re-orchestration opportunity by a *short-timescale orchestrator*, while the configuration of the dedicated capacity is preserved throughout the extended interval, thus reducing cost (*iv*). Both long- and short-timescale orchestrators decide on the amount of (dedicated and shared) resources to be allocated to each slice to also minimize the usual costs (*i*) and (*ii*). This comprehensive strategy results in a 47% reduction of the total cost.

Figure 5.1 exemplifies how reducing instantiation and reconfiguration costs has a price in terms of increased overprovisioning; a multi-timescale orchestration model allows exploring this and more trade-offs for the first time.

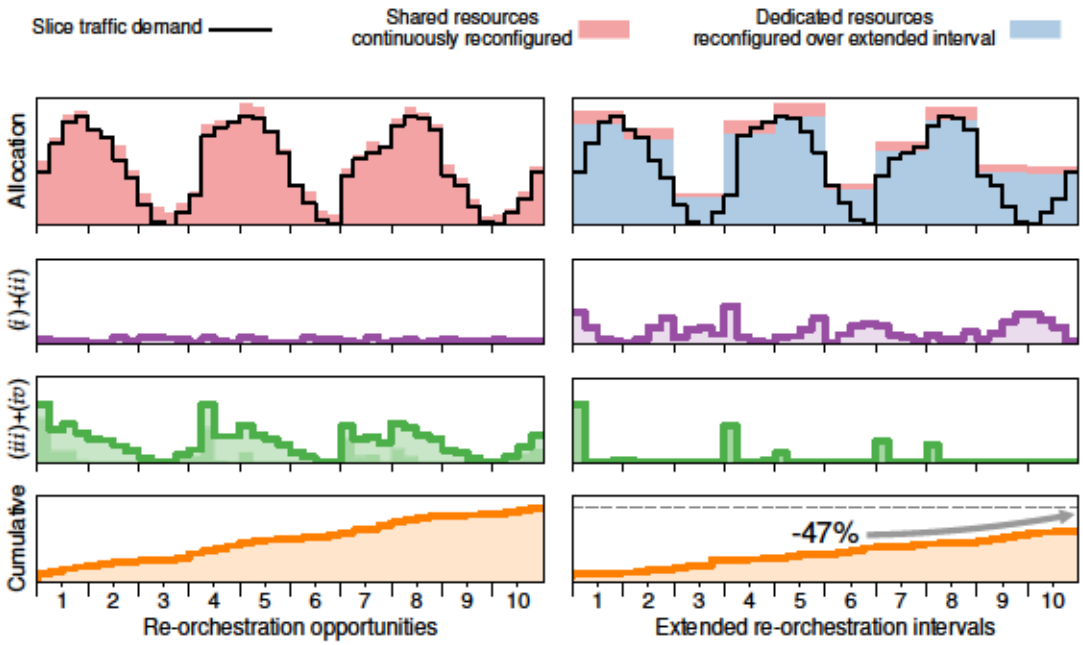


Figure 5.1: Toy example illustrating the costs of resource allocation in network slicing. Top: traffic demand generated by the slice (black solid line), along with the capacity allocated based on the prediction, and reconfigured at all available re-orchestration opportunities (*i.e.*, shared, in red) or only periodically over extended intervals (*i.e.*, dedicated, in blue). Second row: Monetary costs due to (i) overprovisioning and (ii) non-serviced slice traffic. Third row: Monetary costs of resource (iii) instantiation and (iv) reconfiguration. The costs are obtained with $\kappa_o = \kappa_s = \kappa_i = 1$ and $\kappa_r = 0.5$; the meaning of these parameters is explained in Section 5.2. Bottom: cumulative overall cost over time, for components (i)-(iv). Left: DeepCog [9] updates the prediction at the fastest rate possible, following well the demand fluctuations, but forcing continuous reconfigurations. Right: our proposed multi-timescale capacity forecasting trades slightly increased overprovisioning for reduced instantiation costs (only incurred once per extended interval) and reconfiguration fees (absent for dedicated resources).

5.2. Orchestration Model and Trade-offs

Our orchestration model is outlined in Figure 5.2, which also serves the purpose of illustrating the notation used in the remainder of the paper. Let us denote by $\lambda_i(t)$ the traffic demand generated by services running in slice $i \in \mathbb{S}$ at time t. The long-timescale orchestrator operates on extended intervals of duration T_l. At the beginning of each such interval, it takes decisions on the dedicated capacity $x_i^d(t)$ allotted to slice i, $\forall i \in \mathbb{S}$, and on the additional shared capacity $x^s(t)$ available to all slices; all capacities are conserved throughout the following T_l. The bottom plot (A) in Figure 5.2 depicts an example of allocation resulting from a long-timescale orchestration.

Within an extended interval, the short-timescale orchestrator assigns resources to each

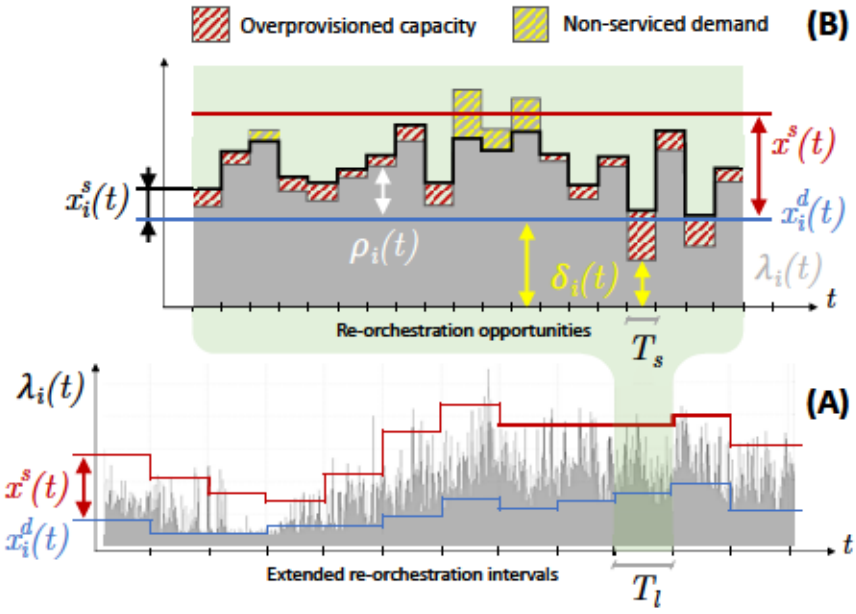


Figure 5.2: Orchestration model. (A) Long-timescale orchestration. The background time series represents the traffic demand generated by slice i (grey). The curves portray the time evolution of the dedicated capacity $x_i^d(t)$ allocated to slice i (blue), and of the shared capacity $x^s(t)$ (red) over extended intervals of duration T_l. Note that $x^s(t)$ is added to the dedicated resources to determine the total available capacity, and, unlike $x_i^d(t)$, is not reserved for slice i but available to all slices. (B) Short-timescale orchestration during one extended interval. At every $T_s < T_l$, a portion $x_i^s(t)$ (black solid curve) of the (fixed) shared capacity $x^s(t)$ is allocated to slice i, based on the residual demand $\rho_i(t)$ not satisfied by the (fixed) dedicated resources $x_i^d(t)$. The plot also highlights the volume of overprovisioned capacity and non-serviced demand (pattern regions), and the slice traffic below dedicated capacity $\delta_i(t)$.

slice i at all re-orchestration opportunities, occurring at every T_s. Decisions are based on the (estimated) future residual demand $\rho_i(t) = \max\{0, \lambda_i(t) - x_i^d(t)\}$, and lead to the allocation of an additional capacity $x_i^s(t)$, for each slice i. The resources $x_i^s(t)$ may be re-configured at every T_s, and are provisioned on top of the dedicated $x_i^d(t)$. The top plot (B) in Figure 5.2 illustrates these definitions for a sample short-timescale orchestration during one extended interval.

5.2.1. Sources of monetary cost

Building on the notation above, we can formally introduce the different costs associated to the management of resources in sliced networks. As anticipated in Section 5.1, there are four sources of economic penalty for the operator, as follows.

(*i*) Unnecessary resource provisioning: the operator incurs a monetary cost in terms of both Capital Expenditure (CAPEX) and Operating Expenses (OPEX) that is

directly proportional to the amount of unused resources it allocates to a slice. Such capacity it is instantiated and configured to no purpose and could be allotted, *e.g.*, to other slices to increase the global system efficiency. This cost at time t is

$$\begin{aligned}&\sum_{i\in\mathbb{S}} f_1\left(\max\{0, x_i^d(t) - \delta_i(t)\}\right) + \\ &\sum_{i\in\mathbb{S}} f_1\left(\max\{0, x_i^s(t) - \rho_i(t)\}\right) + \\ &f_1(x^s(t) - \sum_{i\in\mathbb{S}} x_i^s(t)),\end{aligned} \tag{5.1}$$

where $\delta_i(t) = \min\{\lambda_i(t), x_i^d(t)\}$ denotes the portion of the demand of slice i served by the dedicated capacity at time t, as shown in plot (B) of Figure 5.2. The first two terms in (5.1) denote the cost of overprovisioning at slice i and time t, due to the unneeded allocation of dedicated and shared capacity, respectively; again, we refer the reader to plot (B) of Figure 5.2 for an exemplification. The third term captures instead the overprovisioned shared capacity that is not allocated to any slice by the short-term orchestrator. Function $f_1(\cdot)$ describes the scaling of cost with capacity overprovisioning. As for DeepCog in Section 4.4.1, here we assume a linear increase of the penalty, *i.e.*, $f_1(x) = \kappa_o x$, where κ_o is the monetary cost of one unit of capacity and is expressed in \$/Mbps. However, our model can easily accommodate different definitions of the scaling law, which may apply to specific network functions.

***(ii)* Non-serviced demand**: every time the operator does not allocate sufficient resources to serve the traffic demand of a slice, it violates the SLA with the tenant, which triggers a monetary compensation. The associated cost at time t is

$$\sum_{i\in\mathbb{S}} \kappa_s \cdot \mathbb{1}_{<\rho_i(t)}\left(x_i^s(t)\right), \tag{5.2}$$

where $\mathbb{1}_A(x)$ is an indicator function that takes a value 1 if the argument satisfies condition A, and 0 otherwise. Thus, $\mathbb{1}_{<\rho_i(t)}\left(x_i^s(t)\right)$ activates when the portion of shared capacity assigned to i does not suffice to meet the service demand; this corresponds to an underprovisioning situation, as depicted in Figure 5.2. In these cases, the operator has to indemnify the tenant for a value κ_s, in \$, per SLA violation. This definition is also in line with those used in DeepCog.

***(iii)* Resource instantiation**: in presence of substantial variations of the total traffic demand, the operator needs to instantiate new resources to serve the demand of the slice. In these cases, as discussed in Section 5.1, there exists a cost associated to enabling such new resources. As an example, if additional VMs Virtual Machines (VMs) have to be instantiated or migrated to serve the slice, the operator has increased expenses in terms of power consumption and CPU cycles. In addition, there may be an indirect penalty in

terms of perceived Quality of Service (QOS), as this operation may take minutes [98] and disrupt the end user experience. The cost, triggered at every T_l in our multi-timescale model, can be modeled as

$$\sum_{i \in \mathbb{S}} f_2\left(\delta_i(t)\right) \cdot \mathbb{1}_{>x_i^d(t-1)}\left(x_i^d(t)\right) + f_2\left(\min\{\rho_i(t), x_i^s(t)\}\right) \cdot \mathbb{1}_{>x^s(t-1)}\left(x^s(t)\right). \tag{5.3}$$

The first term in (5.3) represents the penalty incurred when new dedicated resources must be instantiated, which occurs when $x_i^d(t) > x_i^d(t-1)$. The second term is equivalent to the first one, but refers to the shared capacity instantiated to all slices in $\mathbb{S}$. Note that the costs induced by both terms are functions $f_2(\cdot)$ of the affected traffic that may experience disruption, *i.e.*, $\delta_i(t)$ and $\min\{\rho_i(t), x_i^s(t)\}$, respectively.[1] In our performance evaluation, we consider the cost to be directly proportional to the affected traffic, *i.e.*, $f_2(x) = \kappa_i x$, where the parameter κ_i captures the estimated fee for delaying one unit of capacity due to resource instantiation, expressed in \$/Mbps.

***(iv)* Resource reconfiguration**: while resources are only instantiated at every T_l, a short-timescale orchestration of the shared capacity within each extended interval allows accommodating faster fluctuations of the slice demand. Every time the operator reconfigures the shared capacity, it incurs a cost; as mentioned in Section 5.1, this is the case with the reconfiguration of the SDN transport networks, or the setup of load balancers, or the creation of new instances of a Virtual Network Function (VNF) on a VM previously used by another slice. All these operations have a price in terms of management delay [98], expressed as

$$\sum_{i \in \mathbb{S}} f_3\left(\min\{\rho_i(t), x_i^s(t)\}\right) \cdot \mathbb{1}_{\neq x_i^s(t-1)}\left(x_i^s(t)\right). \tag{5.4}$$

The above cost is present whenever the shared resources must be reconfigured for a slice i, *i.e.*, $x_i^s(t) \neq x_i^s(t-1)$. In such situations, the cost is dependent on the amount of traffic affected by the reconfiguration process, *i.e.*, $\rho_i(t)$ bounded by $x_i^s(t)$. In our study, we assume that the economic penalty is the same for any bit of traffic using reconfigured resources, hence $f_3(x) = \kappa_r x$, where κ_r is in \$/Mbps. Also in this case, other functions can be easily embedded in the overall framework to represent distinctive cost models available to the operator.

[1] Accounting for instantiation and reconfiguration costs proportional to the full amount of impacted demand (rather than, *e.g.*, to the increment of demand with respect to the previous re-orchestration opportunity) is a safe choice when considering real network operation. Without guarantees on the stability of the assigned resources in a multi-slice environment, finding a stable and optimal configuration easily entails the reconfiguration of resources allocated to a large portion of the network, or even to all slices [100].

5.2.2. Trade-offs in capacity allocation

The basic trade-off in anticipatory resource assignment is that between overprovisioning and non-serviced demands.

- **Trade-off A.** Increasing the amount of resources makes overprovisioning more likely, but reduces the probability that the allocated capacity is not sufficient to serve the future demand. This results in opposing costs (i) and (ii).

Current capacity forecasting models aim at identifying the optimal compromise that minimizes the joint penalty of the costs in trade-off A above [9]. However, these models do not offer any control over instantiation and reconfiguration. By adopting a multi-timescale approach, now we are instead capable of factoring such variables in. Specifically, the model presented in Section 5.2.1 tells apart the capacity allocated to each slice into a dedicated capacity, re-orchestrated over long timescales with period T_l, and a shared capacity, re-orchestrated over short timescales with period T_s. This unlocks additional degrees of freedom: the orchestrator can decide not only how many resources to assign to a slice, but also which portion of those shall be of each type, and for how long they stay unaltered.

As in DeepCog, AZTEC still allows addressing trade-off A above, by controlling the total allocated capacity during each extended interval T_l, *i.e.*, $x^s(t) + \sum_{i \in \mathbb{S}} x_i^d(t)$, and modulate the costs of overprovisioning and non-serviced demands. Yet, its flexibility enables the exploration of the following additional trade-offs.

- **Trade-off B.** By increasing the dedicated capacity $x_i^d(t)$ allocated to slice i during an extended interval, the orchestrator can serve a larger fraction of the slice traffic with resources that do not need reconfiguration. However, such resources cannot be reused by other slices during T_l whenever they are not needed by slice i. For instance, in plot (B) of Figure 5.2, increasing $x_i^d(t)$ would reduce the residual demand $\rho_i(t)$ that is served with reconfiguration-heavy shared capacity; but it would also generate additional overprovisioning, *e.g.*, in the fourth and second to last re-orchestration opportunities. This leads to a trade-off between costs (i) and (iv).

- **Trade-off C.** Allocating a larger shared capacity $x_i^s(t)$ to slice i during an interval T_s reduces the risk that the resources will not be sufficient to serve the future slice demand. Nevertheless, it also causes the reconfiguration of more resources. As an example, in plot (B) of Figure 5.2, increasing $x_i^s(t)$ in the third T_s slot could remove the non-serviced demand, but would also grow the reconfiguration penalty. A trade-off exists between costs (ii) and (iv).

- **Trade-off D.** Increasing the duration T_l of the extended interval reduces the cost of resource instantiation, which only occurs once per extended interval.

However, a higher T_l also forces the dedicated capacities $x_i^d(t)$ and the total shared capacity $x^s(t)$ to remain constant for a longer time. With a reduced capability to tailor the network resources to the fluctuations of the slice traffic demands, the orchestrator may incur in increased overprovisioning or underprovisioning. For instance, extending the timespan of plot (B) in Figure 5.2 to cover a prolonged demand $\lambda_i(t)$ may create additional situations where $x_i^d(t) > \lambda_i(t)$, *i.e.*, dedicated resources go wasted, as in the second to last T_s interval; it can also generate new cases where $x_i^s(t) < \rho_i(t)$, and traffic peaks are not serviced, as in the central part of the example. This results in a trade-off between cost (*iii*) and joint costs (*i*) and (*ii*).

In the next section, we present a framework that takes automated, anticipatory decisions on capacity allocation by addressing all trade-offs outlined above, thanks to the multi-timescale model.

5.3. The AZTEC Framework

Our framework, named AZTEC (*i.e.*, capacity Allocation for Zero-Touch nEtwork sliCing), automatically solves trade-offs A, B and C above by finding an effective compromise among the opposing goals of reducing the operator's costs in terms of (*i*) overprovisioning, (*ii*) non-serviced slice demands, and (*iv*) resource reconfiguration. In addition, AZTEC offers the operator a handle to control, via a single system parameter, the penalty associated with (*iii*) capacity instantiation, to address trade-off D. Next, we first provide an overview of the framework in Section 5.3.1, and then discuss the implementation of its different components for long- and short-timescale orchestration, in Sections 5.4 and 5.5, respectively.

5.3.1. AZTEC in a nutshell

To solve the complex problem of finding an adequate balance of all fees, we adopt a *divide-et-impera* approach. We separate the different trade-offs between pairs of cost sources and sequentially solve them in isolation. The overall organization of our proposed framework is outlined in Figure 5.3.

AZTEC performs both long- and short-timescale orchestration. As explained in Section 5.1, the long-timescale orchestrator triggers at the beginning of each extended interval, and is in charge of allocating the dedicated capacities $x_i^d(t)$ and the total shared capacity $x^s(t)$, which will then be preserved over the following T_l interval. This function is realized in our framework by blocks (I) and (II), which operate as follows.

(I) This block performs the forecasting of the long-term dedicated capacity for each network slice $x_i^d(t)$, using as input information about the actual traffic

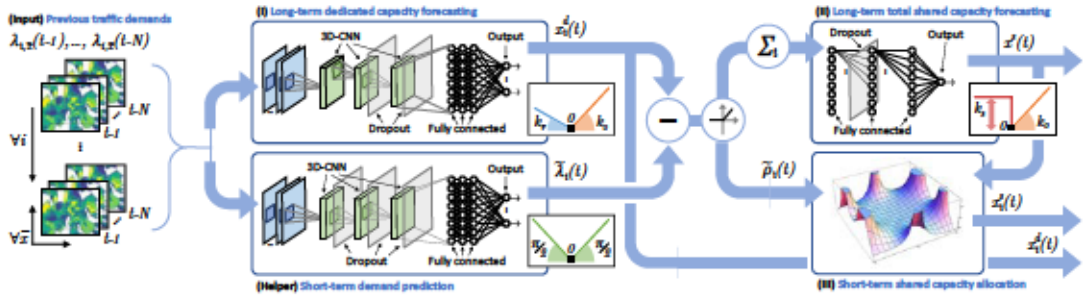


Figure 5.3: Overview of the AZTEC framework. The learning flow proceeds from left to right. The input mobile traffic data is processed by deep neural networks that return, for each slice $i \in \mathbb{S}$, the long-term dedicated capacity $x_i^d(t)$ and the short-term estimated demand $\tilde{\lambda}_i(t)$, respectively. These values are combined to obtain the estimated residual demands $\tilde{\rho}_i(t)$. The aggregate residual demand over all slices is input to a further deep neural network to determine the long-term shared capacity $x^s(t)$. Such capacity is then fed, along with per-slice residuals, to an optimization module that allocates the shared resources $x_i^s(t)$.

generated by each slice during the preceding N re-orchestration opportunities. As discussed for trade-off B in Section 5.2.2, the capacity $x_i^d(t)$ modulates the impact of (i) provisioning unnecessary dedicated resources versus (iv) re-configuring the shared resources needed to serve the residual demand beyond $x_i^d(t)$. Thus, block (I) identifies the $x_i^d(t)$ that minimizes the joint costs (i) and (iv).

(II) This block is in charge of determining the long-term shared capacity $x^s(t)$ available to any slice during the subsequent time interval T_l. The shared capacity is used to serve the residual demands of all slices, hence $x^s(t)$ shall be dimensioned on the aggregate residual traffic $\sum_{i\in\mathbb{S}} \rho_i(t)$. To determine this, block (II) receives as input an estimate of such aggregate during the previous extended interval, *i.e.*, $\sum_{i\in\mathbb{S}} \tilde{\rho}_i(t)$, $t \in [t-(T_l/T_s), t-1]$. The approximate $\tilde{\rho}_i(t) = \max\{0, \tilde{\lambda}_i(t) - x_i^d(t)\}$ come from a forecast $\tilde{\lambda}_i(t)$ returned by a legacy traffic predictor[2] that forecasts per-slice demands over the next re-orchestration opportunity t. Based on such residual demands, block (II) computes $x^s(t)$ such that (i) overprovisioning of shared resources is reduced as much as possible, and (ii) all residual demands can be accommodated within $x^s(t)$. In this way, block (II) addresses trade-off A, jointly minimizing costs (i) and (ii).

Once the long-term capacities $x_i^d(t)$, $\forall i \in \mathbb{S}$, and $x^s(t)$ are set, the short-timescale

[2]Note that we could have employed the real traffic demand $\lambda_i(t)$ observed during the previous interval T_l to determine the actual residuals $\rho_i(t)$. However, the short-timescale orchestrator (presented next) needs to operate on the predicted demand $\tilde{\lambda}_i(t)$ over the future T_s interval. Considering the same estimates in the long-timescale module allows allocating the shared capacity $x^s(t)$ in a way that is conscious of the inaccuracy of the information available during the following short-term resource assignment phase of the framework.

orchestrator assigns portions $x_i^s(t)$ of the total shared resources to each slice. This allocation occurs at every re-orchestration opportunity, spaced by T_s, and is carried out by block (III) of the AZTEC framework as follows.

(III) This block receives as input the long-term shared capacity $x^s(t)$, and the future residual demand $\rho_i(t)$ expected for each slice i during the following T_s. The available total capacity then is allotted to each slice in a way to solve trade-off C in Section 5.2.2. Therefore, block (III) computes $x_i^s(t)$ for each $i \in \mathbb{S}$, by minimizing the combination of costs (ii) and (iv), corresponding to insufficient allocated capacity and additional shared resource reconfiguration, respectively.

Overall, blocks (I)-(III) return a forecast of all capacities $x_i^d(t)$, $x^s(t)$ and $x_i^s(t)$ that the operator shall allocate over both long and short intervals of duration T_l and T_s, respectively. The resulting anticipatory allotment reduces the network management costs associated with the provisioning of exceeding or inadequate resources, and of their reconfiguration over time.

We remark that the penalty of network resource instantiation is not included in this picture. As explained in Section 5.2.2, control on instantiation costs can be achieved by acting on the duration of the extended re-orchestration interval, *i.e.*, T_l: the larger this duration, the lower the costs resulting from resource instantiation in (5.3). AZTEC does not take automated decisions on the value of T_l; instead, it provides via such a parameter an explicit knob that allows the operator to implement any strategy for coping with trade-off D. The rationale is that T_l is a system parameter that is best set by the operator, based on expert knowledge of the underlying virtualization technology.

The implementation of the blocks (I)-(III) above leverages a combination of deep learning architectures and numerical optimization methods, which we detail next.

5.4. Long-timescale orchestration

The long-timescale orchestration is carried out by blocks (I) and (II) of the AZTEC framework, as follows.

5.4.1. Long-term dedicated capacity forecasting

Block (I) is implemented by a Deep Neural Network (DNN) whose structure is inspired by recent breakthroughs in machine learning for image processing [74]. Indeed, block (I) operates on data about the traffic recorded at each base station in the target area over several past time intervals, which can be assimilated to pixels in a time sequence of still images that compose a video. As described in Section 4.3.4, this first requires pre-processing the mobile data traffic, so as to map base station positions into

the matrix form required by DNN, which we do by means of the same correlation-preserving transformation explained in Section 4.3.4. The resulting input is a 4D tensor $\lambda_{i,x}(t-1), \ldots, \lambda_{i,x}(t-N)$, where $\lambda_{i,x}(t)$ is the offered load at the base station associated with matrix element $x = \{m, n\}$, for services running in slice $i \in \mathbb{S}$ and at time t. Following recent advances in machine learning for mobile network traffic analysis, we treat each slice $i \in \mathbb{S}$ in the same way of a color channel in DNN for imaging [101]. This approach lets us process the input along the $\{x, t\}$ dimensions via 3D Convolutional Neural Network (3D-CNN) layers, which are very efficient in extracting spatiotemporal features; at the same time, different slices i are examined in parallel as multiple levels of the same data.

The DNN architecture, illustrated in Figure 5.3, consists of three 3D-CNN layers with 32, 32 and 16 neurons each, and $(3,3,3)$, $(6,6,6)$ and $(6,6,6)$ kernels, respectively. The second and third 3D-CNN layers are followed by dropout layers with probability 0.3, which are known to regularize the network and limit overfitting during the training phase [77]. The convolutional layers constitute the encoder, in charge of extracting meaningful complex features from the data. They are followed by a decoder whose objective is learning global patterns from the feature space. Fully Connected (FC) layers are especially well suited to that purpose, and we leverage three in our implementation, with 64, 32, and $\|\mathbb{S}\|$ neurons, respectively, where operator $\|\cdot\|$ denotes the cardinality of the argument set. Note that the last layer outputs one value per channel, resulting in one value for each slice $i \in \mathbb{S}$. All layers employ Rectified Linear Unit (ReLU) as the neuron activation function, except for the last FC layer, which uses a linear activation function to provide the actual capacity forecast value.

The loss function that drives the DNN training is a custom expression designed to account for the actual management costs incurred by the operator in case of errors in the orchestration of the dedicated capacity. If the operator were able to allocate to a slice i a constant capacity $x_i^d(t)$ that perfectly matched the actual demand $\lambda_i(t)$ over the next T_l, the error and cost would be nil: this is the ideal scenario where all the demand is serviced, without any overprovisioning or re-configuration. However, in practical cases, it is impossible to perfectly predict $\lambda_i(t)$, which is also very unlikely to be constant over the whole T_l. In this case, positive errors $x_i^d(t) - \lambda_i(t)$ lead to overprovisioning, with a cost set by the first term of (5.1) in Section 5.2.1, and negative errors imply that the demand in excess of $x_i^d(t)$ needs to be served by the shared capacity, with (5.4) setting the re-configuration penalty[3].

Positive errors yield $\delta_i(t) = \lambda_i(t)$, while $\rho_i(t) = \lambda_i(t) - x_i^d(t)$ for negative ones. Then, the loss function for $x_i^d(t)$ allocated at t is $\sum_{t \in \mathbb{T}} \ell_i^{(\mathrm{I})}(t)$, where $\mathbb{T}$ is the set of concerned

[3]We remark that the long-term orchestrator is agnostic of the short-timescale resource assignment, and thus cannot take $x_i^s(t)$ into account when computing the cost of negative errors. To deal with this, block (I) assumes a perfect management of shared capacity that always accommodate the residual demand; also, it considers that the residual demand always varies across re-orchestration opportunities. This simplifies (5.4) to $\sum_{i \in \mathbb{S}} f_3(\rho_i(t))$.

re-orchestration opportunities $\{t, \dots, t + (T_l/T_s) - 1\}$, and

$$\ell_i^{(\mathrm{I})}(t) = \begin{cases} f_3\left(\lambda_i(t) - x_i^d(t)\right) & \text{if } x_i^d(t) \leq \lambda_i(t) \\ f_1\left(x_i^d(t) - \lambda_i(t)\right) & \text{otherwise.} \end{cases} \tag{5.5}$$

We stress that (5.5) satisfies the desirable property of having partial derivatives that form a piece-wise constant function, which guarantees robust and fast convergence under popular first-order optimizers like Adam [79].

In order to further improve the quality of the allocation of dedicated resources, we leverage a recent result in neural network design, which allows estimating the uncertainty of the learning outcome. Specifically, adding dropout layers during model testing is mathematically equivalent to generating an approximation of the probabilistic deep Gaussian process [102]. This observation, which holds for DNNs with arbitrary depth and non-linearities, provides a way to return a distribution instead of a single output value. Following this, we activate the dropout layers, adopt a Monte Carlo strategy and perform the forward pass L times for each test input, obtaining outputs $\{x_i^{d,1}(t), \dots, x_i^{d,L}(t)\}$ for, slice i at time t. We then compute the mean $\mu_i^d(t) = 1/L \cdot \sum_{l=1}^{L} x_i^{d,l}(t)$ as well as the variance $\sigma_i^d(t) = 1/L \cdot \sum_{l=1}^{L} (\mu_i^d(t) - x_i^{d,l}(t))^2$, and approximate the model uncertainty as $\mathcal{N}(\mu_i^d(t), \sigma_i^d(t))$.

The knowledge of the model uncertainty can be then leveraged to add a safety margin to the standard DNN outcome. Since the whole support of $\mathcal{N}(\mu_i^d(t), \sigma_i^d(t))$ represents potentially correct values of the dedicated capacity, block (I) returns the 99^{th} percentile of the distribution; this makes it very unlikely to output a $x_i^d(t)$ that is lower than the best one. Thus, when the DNN is confident about the quality of the result, it returns a value close to the mean; vice versa, it adds a substantial safety margin. We provide an example of the advantage of this approach in Section 5.6.1.

5.4.2. Long-term total shared capacity forecasting

Block (II) is also implemented using a dedicated DNN. The structure, in this case, is simpler, as the network operates on a sensibly less rich input than that of Block (I). Specifically, the input is a single time series of the total residual demand in the past extended interval, *i.e.*, $\sum_{i \in \mathcal{S}} \rho_i(t)$, $t \in [t - (T_l/T_s), t - 1]$. The time series is processed by three FC layers with 128, 64 and 1 neurons, respectively; the first two use ReLU activation functions, while the last uses a linear function to produce the final output. A dropout layer with probability 0.2 is present between the first and second FC layers.

The DNN is trained with a different custom loss function that accounts for the correct sources of monetary penalty in case of errors. As with the previous DNN, an ideal case where a fixed long-term shared capacity $x^s(t)$ perfectly matches a constant aggregate residual demand is unrealistic, and errors are unavoidable. Positive errors yield

overprovisioning of $x^s(t)$, whereas negative ones have a cost in terms of denied traffic. The former corresponds to the second and third terms[4] in (5.1), while the latter to the monetary fee[5] for SLA violations in (5.2). By writing these costs jointly for the extended interval starting at t, we obtain a loss function $\sum_{t\in\mathbb{T}} \ell_i^{(\mathrm{II})}(t)$, where

$$\ell_i^{(\mathrm{II})}(t) = \begin{cases} \kappa_s & \text{if } x^s(t) < \sum_{i\in\mathbb{S}} \rho_i(t) \\ f_1\left(x^s(t) - \sum_{i\in\mathbb{S}} \rho_i(t)\right) & \text{otherwise.} \end{cases} \tag{5.6}$$

The expression in (5.6) needs to be slightly modified by adding minimum slopes that make the function differentiable over $\mathbb{R}$. With this, the loss function has the same desirable properties as the ones mentioned for (5.5). Finally, we take advantage of the dropout layer also in this case: we thus approximate the model uncertainty, and return the 99^{th} percentile of the distribution as a safety margin on the correct value of $x^s(t)$.

5.5. Short-timescale orchestration

The short-timescale orchestration consists of block (III) supported by a helper short-term demand predictor, as outlined in Figure 5.3. The predictor uses a DNN architecture that is very similar to that adopted by block (I); indeed, the two DNNs operate on the same input, and produce per-slice forecasts. The main differences between them is in the frequency of operation and, most notably, in the loss function. The helper predictor outputs a prediction at every T_s instead of at every T_l. Furthermore, it uses a traditional Mean Absolute Error (MAE) instead of the cost-aware loss function in (5.5): Mean Absolute Error (MAE) considers identical contributions by the positive and negative errors, thus producing an output $\lambda_i(t)$ that tries to follow as closely as possible the upcoming slice traffic demands.

5.5.1. Short-term shared capacity allocation

Given the total shared capacity $x^s(t)$, and the estimated residual demands $\rho_i(t)$, AZTEC has to decide how to distribute $x^s(t)$ across the requesting slices at every T_s. This is implemented in block (III) with a numerical optimization method.

The primary objective of the shared resource assignment performed by block (III) is avoiding SLA violations due to insufficient available capacity, which would induce a cost

[4]Note that, as per the remark in footnote 3, the shared capacity allotted to individual slices $x_i^s(t)$ used in the original expression in (5.1) has not yet been determined at this stage. The safest option is hence to assume that the whole residual demands will be correctly assigned by the short-term orchestrator. This leads to approximating the allocated shared resources $x_i^s(t)$ by $\rho_i(t)$. Under this hypothesis, the second and third terms in (5.1) reduce to 0 and $f_1\left(x^s(t) - \sum_{i\in\mathbb{S}} \rho_i(t)\right)$, respectively.

[5]The same observation as in footnote 4 applies here. Under an identical assumption of a perfect short-term assignment of $x_i^s(t) = \rho_i(t)$, the only case where some demand can be denied is that of an insufficient total shared capacity $x^s(t)$. Then, (5.2) translates into $\kappa_s \cdot \mathbb{1}_{<\sum_{i\in\mathbb{S}} \rho_i(t)}(x^s(t))$.

modelled in (5.2). At the same time, issuing non-essential resources has a penalty in terms of unnecessary reconfiguration cost, as captured by the expression in (5.4). This corresponds to trade-off C in Section 5.2.2 and can be formulated as

$$\begin{aligned} \min_{x_i^s(t)} \quad & \sum_{i \in \mathbb{S}} \kappa_s P\left(\rho_i(t) > x_i^s(t)\right) \\ \text{subject to} \quad & \sum_{i \in \mathbb{S}} x_i^s(t) \leq x^s(t), \end{aligned} \tag{5.7}$$

The above minimizes the expected value of the expenditure for non-serviced slice demands in (5.2). Note that, by squeezing as many slices as possible within the total capacity, the solution to (5.7) implicitly addresses the challenge of minimizing reconfiguration costs. Also, the formulation in (5.7) deals with probabilities: this is consistent with the probabilistic nature of $\rho_i(t)$ granted by uncertainty approximation via dropout layers.

Due to the empirical nature of the probability distribution $\rho_i(t)$, (5.7) has no closed form and thus we cannot employ classical optimization methods. Furthermore, as we do not have a differentiable objective function, we cannot employ approaches that depend on gradients. Hence, we apply numerical methods that search for the optimal solution within the feasible set of $[x_i^s(t)]$. To simplify this search, we apply the following variable change:

$$p_i(t) = \begin{cases} x_i^s(t) / \left(x_i^s(t) + x_{i+1}^s(t)\right) & \text{if } i \in [1, \|\mathbb{S}\| - 1] \\ \sum_i x_i^s(t) / x^s(t) & \text{if } i = \mathbb{S} \end{cases} \tag{5.8}$$

which yields $p_i(t) \in [0, 1],\ \forall i \in \mathbb{S}$.

The above allows us to search in the N-dimensional variable space with fixed bounds $[0, 1]$ on all variables. Note that the first $\|\mathbb{S}\| - 1$ variables $p_i(t)$ represent the relative amount of shared capacity assigned to slice i with respect to the slice $i + 1$, while the last $p_{\|\mathbb{S}\|}(t)$ represents the overall amount of shared capacity assigned to the services. This variable change serves the following purposes: first, the constraint on the sum of the variables is now enforced through a constraint on each variable; second, we avoid ties between variables, allowing a safe exploration of the solution space where we can focus one variable, and changing its $p_i(t)$ value within the entire range without impacting any of the other variables $p_j(t)$ for $j \neq i$.

Algorithm 4 details `AZTEC` assignment algorithm for shared resources. The algorithm is composed by a main function, which takes as input the total available shared capacity $x^s(t)$ and the empirical distribution of the capacity needed by each service in the next time interval $\rho_i(t)$, and two helper functions: `COST` and `TRANSFORM`. The former computes the cost expected value for a given assignment $p_1(t), \ldots, p_{\|\mathbb{S}\|}(t)$, while the latter transforms $p_i(t)$ back to $x_i^s(t)$. By using the variable change described above, we can use a gradient-free algorithm which works with constrained input variable for the minimization of (5.7). In particular, we chose as numerical optimizer the BOBYQA Algorithm [103], which is a

Algorithm 4 Shared resource assignment algorithm

1: **Function** *TRANSFORM* $(p_1, \ldots, p_{\|\mathbb{S}\|})$:
2: $\quad x^s_{\|\mathbb{S}\|} = \frac{p_{\|\mathbb{S}\|} x^s}{\left(\sum_{i=1}^{\|\mathbb{S}\|-1} \prod_{j=i}^{\|\mathbb{S}\|-1} \frac{p_j}{1-p_j}\right)+1}$
3: $\quad x^s_i = \left(\prod_{j=i}^{\|\mathbb{S}\|-1} \frac{p_j}{1-p_j}\right) x^s_{\|\mathbb{S}\|},\ i \in [1, \|\mathbb{S}\| - 1]$
4: **return** $x^s_1, \ldots, x^s_{\|\mathbb{S}\|}$
5: **Function** *COST* $(p_1, \ldots, p_{\|\mathbb{S}\|})$:
6: $\quad x^s_1, \ldots, x^s_{\|\mathbb{S}\|} \leftarrow$ *TRANSFORM* $(p_1, \ldots, p_{\|\mathbb{S}\|})$
7: $\quad X \leftarrow \sum_i \kappa_s P\left(\rho_i > x^s_i\right)$
8: **return** X
9: **Function** *MAIN* (x^s, ρ_i):
10: $\quad$ Set $p_1 \ldots p_{\|\mathbb{S}\|-1} = 0.5$
11: $\quad$ **Let** $c \leftarrow$ *BOBYQA* $(\mathit{COST}(p_1, \ldots, p_{\|\mathbb{S}\|}))$ *Fixed* $p_1, \ldots, p_{\|\mathbb{S}\|-1}$,
12: $\quad\quad p^0_1, \ldots, p^0_{\|\mathbb{S}\|} \leftarrow$ *GOLDEN* $(c(p_{\|\mathbb{S}\|}))$;
13: $\quad p_1, \ldots, p_{\|\mathbb{S}\|} \leftarrow$ *BOBYQA* $(p^0_1, \ldots, p^0_{\|\mathbb{S}\|})$
14: $\quad x^s_1, \ldots, x^s_{\|\mathbb{S}\|} \leftarrow$ *TRANSFORM* $(p_1, \ldots, p_{\|\mathbb{S}\|})$
15: **return** $x^s_1, \ldots, x^s_{\|\mathbb{S}\|}$

gradient-free optimizer that allows constrained variables.

When running the above numerical method, we have to set a starting point. We set the relative ratios of shared capacity across services as 0.5, *i.e.*, all services receive the same amount of resources (see line 10). Given the possibly high number of services that can be included in the system, there may be different local minima and there exists the chance of getting stuck in a local minima that does not deliver a good performance. To reduce the probability that this happens, we perform a search for the best starting $p_{\|\mathbb{S}\|}(t)$. In particular, we run the Golden Section method [104] considering $p_{\|\mathbb{S}\|}(t)$ as the only variable for the cost. After this step, the initial value of $p_{\|\mathbb{S}\|}(t)$ is set to the value resulting from the Golden Section search. With this, we obtain the starting p^0 set $p^0_1(t), \ldots, p^0_{\|\mathbb{S}\|}(t)$. Then, we perform a BOBYQA minimization using it as the starting point. This provides the values that minimize the overall expected cost.

5.6. AZTEC Performance Evaluation

In this section we evaluate the performance of AZTEC which framework has been detailed in Section 5.3. We evaluate AZTEC with an extensive dataset of mobile data traffic collected at 470 eNodeBs of a real-world network serving a large metropolitan region in Europe, the same utilized to assess DeepCog's performance in Section 4.5. Similarly, the measurement data concerns a set of five popular and heterogeneous mobile services, namely Youtube, Facebook, Instagram, Snapchat, and iTunes, whose traffic flows were classified by the operator using proprietary Deep Packet Inspection (DPI) techniques.

We assume that each mobile service is associated to one dedicated slice in $\mathbb{S}$, and

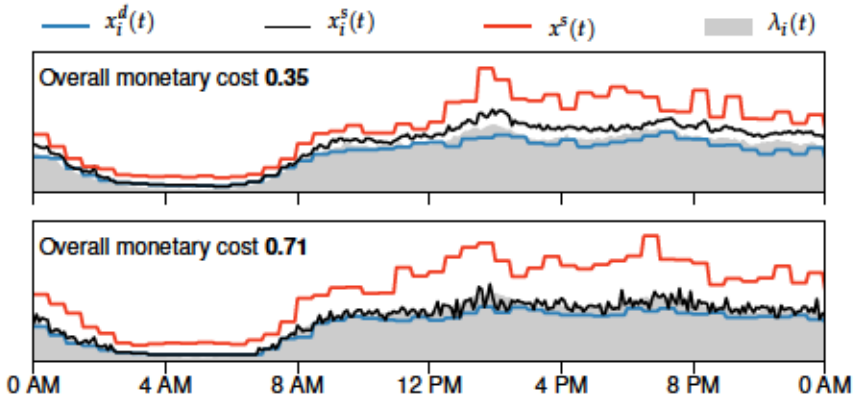


Figure 5.4: Time series of sample resource allocations. Top: AZTEC framework. Bottom: equivalent framework where no uncertainty estimates are used.

investigate the anticipatory allocation of resources at a single datacenter that runs virtualized core network functions over the mobile data traffic generated in the city under study. To this end, we train the AZTEC framework on eight weeks of data, use two additional weeks for validation, and finally run experiments on another two weeks. The three time periods do not overlap, and all results refer to the test phase only. Upon extensive appraisal of the system performance, we have observed that the DNNs in AZTEC best operate with data from the previous $N = 6$ intervals of duration T_s, hence we employ a $[6 \times 47 \times 10 \times 5]$ 4D tensor input. Unless stated otherwise, our default settings are $\kappa_o = \kappa_s = \kappa_i = 1$ and $\kappa_r = 0.5$, so as to account for the typically relatively lower cost per Mbps of resource reconfiguration; also, we set $T_s = 5$ minutes and $T_l = 30$ minutes to align with the capabilities of current Virtual Infrastructure Manager (VIM) [89]. In all our tests, AZTEC returned capacity allocations within one second, which is suitable for real-time operation in practical systems.

5.6.1. Harnessing the forecast uncertainty

As presented in Section 5.3, we leverage a recent result on the approximation of uncertainty in DNN to include a safety margin in the model forecast. In a preliminary step for our evaluation of AZTEC, we investigate the impact of including the knowledge of the estimated uncertainty in the forecast produced by the different DNNs that are part of the framework.

Figure 5.4 visually shows the benefit of this design choice, by comparing the AZTEC with and without uncertainty; in the second case, dropout layers are deactivated during test, and all DNNs produce a legacy single-value output. In each plot, we report the anticipatory allocation of capacities $x_i^d(t)$ and $x_i^s(t)$ to the target slice i, as well as that of the total shared capacity $x^s(t)$; the actual demand is on the background. The top plot

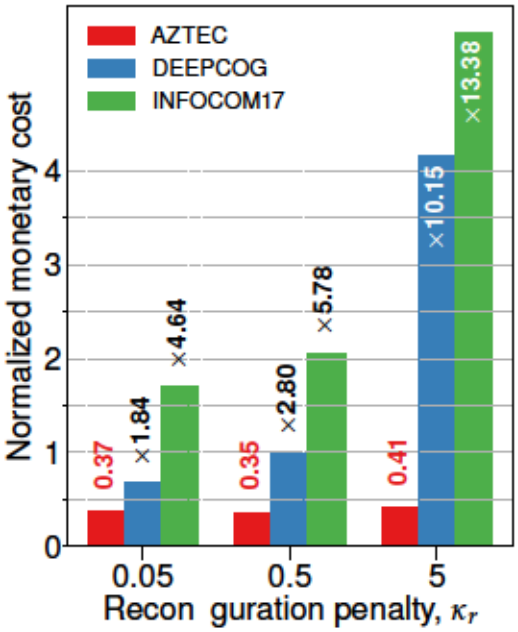


Figure 5.5: Normalized monetary cost of AZTEC and two benchmarks versus the reconfiguration cost scaling factor κ_r. Numbers denote the exact cost for AZTEC, and the added cost factor for the benchmarks.

provides in fact an illustrative example of the typical time-varying resource allocation achieved by AZTEC, where capacities follow the fluctuations of the slice traffic.

More interestingly, AZTEC achieves a more reliable assignment of slice resources by accounting for the level of uncertainty of the predictions. The total capacity $x_i^d(t) + x_i^s(t)$ is smoother and avoids situations where the demand cannot be serviced. Conversely, the framework not accounting for uncertainties yields a capacity allocation that is noisy and incurs in substantial SLA violations due to unsatisfied demands. In addition, AZTEC achieves such a result while saving on the amount of allocated resources (note the lower $x^s(t)$ curve), which ultimately results in an overall monetary cost that is half of that incurred by the framework without uncertainties. While this is an excerpt from a specific test, we recorded similar gains for all different settings explored in our analysis.

5.6.2. Comparative evaluation

We next assess the performance of AZTEC against two recent benchmarks: INFOCOM17, a state-of-the-art mobile network traffic predictor [57], and DeepCog, the capacity forecasting model for network resource allocation described in Section 4.3; both solutions are based on custom-built DNNs. INFOCOM17 is a traditional demand predictor, agnostic of all resource management costs, whereas DeepCog takes anticipatory decisions on capacity allocation that aim exclusively at minimizing the trade-off A of overprovisioning and

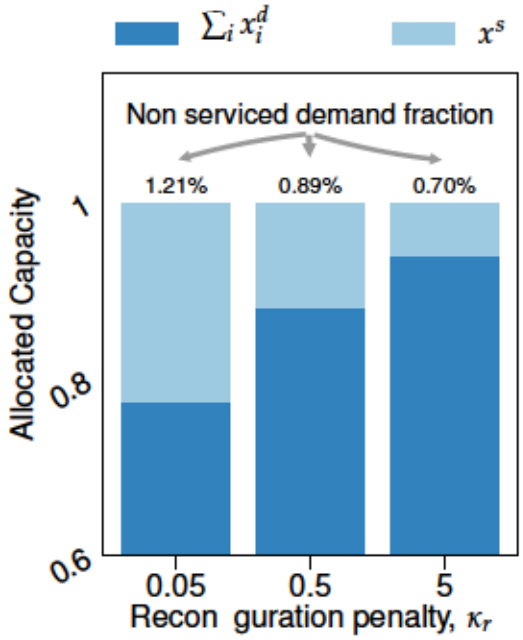


Figure 5.6: Total dedicated capacity $\sum_{i \in \mathbb{S}} x_i^d(t)$ and shared capacity $x^s(t)$ allocated by AZTEC versus κ_r. Numbers denote the fraction of re-orchestration opportunities with insufficient allocated resources.

non-serviced demands.

Figure 5.5 summarizes the result of the comparative evaluation, showing the overall normalized[6] monetary cost of the anticipatory resource management against reconfiguration prices spanning two orders of magnitude. The gain of AZTEC over the benchmarks is clear, as even a state-of-the-art capacity predictor like INFOCOM19 increments the cost for the operator by a factor that ranges from 1.7 to beyond 10. As expected, the benchmark solutions inherently suffer more when reconfiguration costs –which they neglect– grow; however, even in a situation favorable to reconfiguration-agnostic approaches where such costs are small (*e.g.*, for $\kappa_r = 0.05$), AZTEC still yields a lower economic fee.

Finally, not only the relative performance is promising, but also the absolute (normalized) values show the potential advantage of a zero-touch network slicing paradigm. Indeed, by automatically and dynamically allocating capacity in advance, AZTEC can cut management costs down to 35-41% of those incurred with an optimal

[6]Due to confidentiality reasons, we cannot make the actual economic cost explicit, as it would reveal the operator's mobile data traffic volumes. We thus normalize all results by the cost of an optimal but completely static resource allocation that dimensions the dedicated capacity to the traffic peak of each slice during the test period. This optimal allocation only incurs into overprovisioning costs, computed as $\sum_{i \in \mathbb{S}} \sum_t f_1 \left(\max_t\{\lambda_i(t)\} - \lambda_i(t)\right)$, which represents the normalization term.

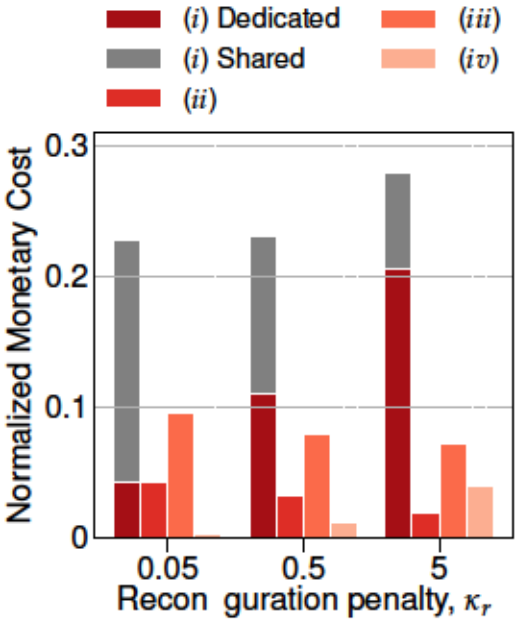


Figure 5.7: Breakdown of the normalized monetary cost by penalty type. Tags refer to cost definitions in Section 5.2.1, with overprovisioning cost (i) separated into contributions of the dedicated capacity, *i.e.*, the first term of (5.1), and of the shared capacity, *i.e.*, the second and third terms of (5.1).

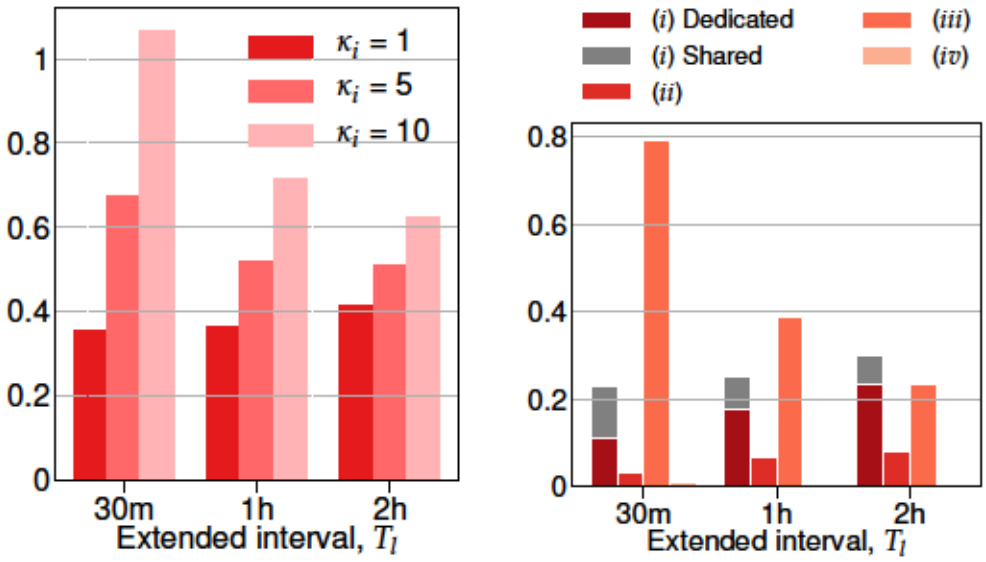
static provisioning of resources[7].

5.6.3. Monetary cost breakdown

The above results prove that AZTEC maintains the combined costs described in Section 5.2.1 under control across a wide range of reconfiguration penalties, by properly adapting the portion of capacity allocated as dedicated, *i.e.*, $x_i^d(t)$, and as shared, *i.e.*, $x^s(i)_t$, to each slice i. To gain further insights on this, Figure 5.6 illustrates the capacity breakdown for tests in Figure 5.5: a wider fraction of traffic is allocated to the more flexible shared capacity when the reconfiguration fee is low; instead, moving traffic to the dedicated capacity becomes more cost-efficient as κ_r grows. An important remark is that in all cases AZTEC incurs into SLA violations due to insufficient available resources in 0.7-1.21% of the re-orchestration opportunities: as a term of comparison, DeepCog causes violations in at least 5.80% of the system observation time.

The ductility of the AZTEC orchestration results in a relative contribution of each cost source that stays fairly constant for different values of κ_r, as shown in Figure 5.7. Here, the most notable trend is in the split of the overprovisioning cost, which grows for the dedicated capacity and decreases for the shared capacity. Indeed, and consistently

[7]See footnote 6.



(a) (b)

Figure 5.8: (a) Normalized monetary cost vs the duration of the extended re-orchestration interval T_l, for different scaling factors κ_i of the resource instantiation cost. (b) Breakdown of the normalized monetary cost by penalty type, for $\kappa_i = 10$.

with Figure 5.6, provisioning dedicated resources in excess becomes more convenient than reconfiguring shared resources as κ_r becomes larger.

Overall, these results demonstrate the capability of AZTEC to properly solve the multiple trade-offs among resource management costs outlined in Section 5.2.2, helping the operator to drastically reduce operation expenses in an automated way.

5.6.4. Controlling resource instantiation costs

Although AZTEC does not take autonomous decisions on trade-offs involving the instantiation of resources, it still offers direct control on the associated cost (*iii*) by means of the T_l parameter. Indeed, as operators incur into this type of fee once every extended re-orchestration interval, making this longer allows limiting the penalty. In Figure 5.8a we investigate the effectiveness of such a lever to control the instantiation cost, where T_l varies from 30 minutes to 2 hours.

The results show that T_l has an impact on the total cost when it is actually needed. In other words, when the scaling factor κ_i is small, the influence of the resource instantiation penalty on the total cost is negligible, and varying T_l has little effect. However, as κ_i grows, the duration of the extended interval becomes a functional handle to control the (now substantial) resource instantiation fee: increasing T_l from 30 minutes to 2 hours can reduce the overall cost by 40% when $\kappa_i = 10$.

Further detail is provided in Figure 5.8b, where the contribution of the different management costs are told apart, for $\kappa_i = 10$. We observe that, as expected, the cost of (i) overprovisioning and (ii) non-serviced demands grow with T_l. Indeed, if capacities $x_i^d(t)$ and $x^s(t)$ remain fixed over a longer interval, this limits the flexibility of the orchestrator to follow fluctuations in the demand, and forces a more challenging forecast over a longer time horizon. However, and more importantly, longer T_l have a clear positive impact on the instantiation cost, which they can reduce by a factor 4 in this specific case study. Ultimately, these results demonstrate the effectiveness of AZTEC in offering the operator with a means to control resource instantiation costs.

6 Conclusions

Network slicing will be one of the pillars of future 5G networks. In this thesis, we investigated the problems for Admission Control and Resource Orchestration in Network Slicing, two critical functions of the next mobile network generation. We have identified the main research problems and by integrating Artificial Intelligence (AI) in the network, designed novel algorithms and solutions to enable sliced 5G networks.

Network Slicing will bring new players to the 5G business model: the Infrastructure Providers (InPs) will sell their resources to tenants which, in turn, provide a service to their users. An open problem within this model is how to admit requests from the tenants, ensuring that the corresponding SLAs will be satisfied while maximizing the monetization of the InP. There is the need of new resource allocation mechanisms that take into account the relationship between the various players.

In the first part of this thesis, we have addressed this issue by designing an admission control algorithm to be executed by the InP when receiving slice requests from the tenants. We first present a model based on Semi-Markov Decision Process (SMDP) for the decision-making process and formulate the optimal revenue problem built on Value Iteration. While this is very useful in order to obtain a benchmark for comparison, the algorithm itself has a very high computational cost, which makes it impracticable for real scenarios. Building on this model, we have first designed an adaptive online algorithm based on Q-learning that aims at maximizing revenue by learning from the outcome resulting from the previous decisions. Even though the Q-learning algorithm provides close to optimal performance and reduces the computational cost compared with Value Iteration, it becomes infeasible when the state space grows. In Section 3.4.4, we have then designed an algorithm based on Neural Network (NN): the Network-slicing Neural Network Admission Control (N3AC) algorithm. The performed evaluation shows that N3AC (*i*) performs close to optimal performance, (*ii*) substantially outperforms naive approaches as well as smart heuristics, and (*iii*) only requires few hundreds iterations to converge to optimal performance. Furthermore, the proposed AI solution scales to large scenarios and it is flexible enough to adapt to real scenarios.

In Section 4.3, we have presented DeepCog, an original data analytics tool for the cognitive management of resources in sliced 5G networks. DeepCog tackles the novel problem of capacity forecasting, whose solution is key to the sustainable operation of future multi-tenant mobile networks. Indeed, once slices are admitted into the network, the InP has to allocate them enough resources for their services delivering with a predefined quality while minimizing its operational costs. Inspired by recent advances in deep learning for image and video processing, DeepCog hinges upon a deep neural network structure, which analyzes antenna-level demand snapshots for different services in order to provide a prediction of the resources that the operator has to allocate to accommodate the future load. The operation is performed for individual mobile services separately, and over a configurable time horizon. At the core of DeepCog there is α-OMC, a new and customized loss function that drives the deep neural network training so as to minimize the monetary cost contributed by two main deployment fees, *i.e.*, overprovisioning and SLA violation. Ours is, to the best of our knowledge, the only work to date where a deep learning architecture is explicitly tailored to the problem of anticipatory resource orchestration in mobile networks. The solution presented in this thesis thus represents a first attempt to integrate data analytics based on machine learning into an overall cognitive management framework. Thorough empirical evaluations with real-world metropolitan-scale data show the substantial advantages granted by DeepCog over state-of-the-art predictors and other automated orchestration strategies, providing a first analysis of the practical costs of heterogeneous network slice management across a variety of case studies.

Assuming that resource instantiation and reconfiguration occur at no costs in a sliced network is not generally valid. For this reason in Chapter 5 we have designed `AZTEC`, a practical multi-timescale orchestration approach for slicing-capable networks. This approach combines deep learning tools with classic optimization algorithms to provide a zero-touch anticipatory capacity forecasting for slices. By separating the long-term assignment of slice-dedicated resources from the short-term re-orchestration of shared resources, it manages to (i) minimize resource instantiation costs, while (ii) ensuring that the service demands of all slices always are met, by timely reconfiguring the assignment of shared resources. In `AZTEC` the long-term slice-dedicated resources and the short-term orchestration of shared resources are performed for all the slices at the same time. Moreover, leveraging on recent result on the approximation of uncertainty in DNN, `AZTEC` includes in the forecast process a safety margin that further reduces the monetary cost. The evaluation results, performed on extensive real-world data show that `AZTEC` significantly improves the performance of state-of-the-art solutions, while providing operators with a fine-level control on the underlying system.

In summary, in this thesis we have discussed the potentially critical role of AI for the management and operation of mobile networks that implement network slicing. AI-based

solutions can address the different and very complex problems that emerge at multiple levels, including admission control of new network slices and resource allocation to slices in the Radio Access Network (RAN), Mobile Edge Computing (MEC) and network core. We outlined practical deep learning architectures that can solve such problems and designed three different AI-based approaches, illustrating the high typical gain that one can expect from integrating AI in network slicing.

The research conducted in this thesis sets a first step through the integration of AI in the next mobile network generation. Two main outcomes from this work that could drive future research directions are briefly described next.

1. We have demonstrated that AI-based approaches can be very effective for different problems around network slicing as this involves large traffic volumes and relatively relaxed timescales. Future extensions of the proposed approaches can look at the trade-off between the complexity required by applying these algorithms and QOS requirements in case of reduced timescales.

2. In this thesis, we have carefully described a methodology to apply AI to networking problems. Detailed reasoning behind all the choices involving the input/output arrangement and architectural aspects have been provided. This methodology could potentially be leveraged to design algorithms for other networking problems.

References

[1] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G Infrastructure Markets: The Business of Network Slicing," in *Proc. IEEE INFOCOM'17*, Atlanta, USA, May 2017, pp. 910–918.

[2] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[3] P. Rost *et al.*, "Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[4] D. Bega, M. Gramaglia, C. J. Bernardos Cano, A. Banchs, and X. Costa-Perez, "Toward the network of the future: From enabling technologies to 5G concepts," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 8, 2017.

[5] D. S. Michalopoulos, M. Doll, V. Sciancalepore, D. Bega, P. Schneider, and P. Rost, "Network slicing via function decomposition and flexible network design," in *in Proc. of IEEE PIMRC*, 2017, pp. 1–6.

[6] P. Serrano, M. Gramaglia, D. Bega, D. Gutierrez-Estevez, G. Garcia-Aviles, and A. Banchs, "The path toward a cloud-aware mobile network protocol stack," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 5, p. e3312, 2018.

[7] D. Bega, A. Banchs, M. Gramaglia, X. Costa-Pérez, and P. Rost, "CARES: Computation-Aware Scheduling in Virtualized Radio Access Networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 7993–8006, Oct. 2018.

[8] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, and X. Costa-Perez, "A machine learning approach to 5G infrastructure market optimization," *IEEE Transactions on Mobile Computing*, 2019.

[9] D. Bega *et al.*, "Deepcog: Cognitive network management in sliced 5g networks with deep learning," in *IEEE INFOCOM 2019*, Apr. 2019.

[10] D. Bega *et al.*, "Alfa-OMC: cost-aware deep learning for mobile network resource orchestration," in *Proc. of IEEE INFOCOM NI Workshop*, Paris, France, May 2019, pp. 1–9.

[11] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Optimizing Resource Provisioning in Network Slicing with AI-based Capacity Forecasting," *IEEE JSAC SI-MLinSDNNFV*, Dec. 2019.

[12] ——, "AZTEC: Anticipatory Capacity Allocation for Zero-Touch Network Slicing."

[13] D. Bega, M. Gramaglia, A. Garcia-Saavedra, M. Fiore, A. Banchs, and X. Costa-Perez, "Network Slicing Meets Artificial Intelligence: an AI-based Framework for Slice Management."

[14] NGMN Alliance, "Description of Network Slicing Concept," Public Deliverable, 2016.

[15] ITU-R, "Minimum requirements related to technical performance for imt-2020 radio interface(s)," Technical Report, 2017.

[16] C. Zhang *et al.*, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, Mar. 2019.

[17] 3GPP, "Telecommunication management;Study on management and orchestration of network slicing for next generation network," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 28.801, 2018. [Online]. Available: http://www.3gpp.org/DynaReport/28801.htm

[18] NGMN Alliance, "NGMN Use Cases related to Self Organising Network, Overall Description," White Paper, May 2007.

[19] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[20] NGMN Alliance, "5G White Paper," Public Deliverable, Feb. 2015.

[21] 5G-PPP Architecture Working Group, "View on 5G Architecture," Public Deliverable, Jul. 2016.

[22] 3GPP, "Technical Specification Group Services and System Aspects; Study on Architecture for Next Generation System," TR 23.799, v0.7.0, Aug. 2016.

[23] A. Gudipati, L. Li, and S. Katti, "RadioVisor: A Slicing Plane for Radio Access Networks," in *Proc. of ACM HotSDN*, Chicago, Illinois, Aug. 2014, pp. 237–238.

[24] I. Malanchini, S. Valentin, and O. Aydin, "Generalized resource sharing for multiple operators in cellular wireless networks," in *Proc. IEEE IWCMC*, Nicosia, Cyprus, Aug. 2014, pp. 803–808.

[25] R. Mahindra, M. A. Khojastepour, H. Zhang, and S. Rangarajan, "Radio Access Network sharing in cellular networks," in *Proc. IEEE ICNP*, Göttingen, Germany, Oct. 2013, pp. 1–10.

[26] S. Rathinakumar and M. Marina, "GAVEL: Strategy-proof Ascending Bid Auction for Dynamic Licensed Shared Access," in *Proc. ACM MobiHoc*, Paderborn, Germany, Jul. 2016, pp. 121–130.

[27] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: enabling enterprises' own software-defined cellular networks," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 146–153, Jul. 2016.

[28] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 32–39, Jul. 2016.

[29] M. A. Habibi, B. Han, and H. D. Schotten, "Network Slicing in 5G Mobile Communication Architecture, Profit Modeling, and Challenges," *arXiv:1707.00852*, 2017.

[30] B. Han, S. Tayade, and H. D. Schotten, "Modeling profit of sliced 5G networks for advanced network resource management and slice implementation," in *Proc. of IEEE ISCC*, Heraklion, Greece, Jul. 2017, pp. 576–581.

[31] B. Han, D. Feng, L. Ji, and H. D. Schotten, "A Profit-Maximizing Strategy of Network Resource Management for 5G Tenant Slices," *a arXiv:1709.09229*, 2017.

[32] S. Shenker, "Fundamental design issues for the future Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 7, pp. 1176–1188, 1995.

[33] M. Berenson, D. Levine, K. A. Szabat, and T. C. Krehbiel, *Basic business statistics: Concepts and applications.* Pearson higher education AU, 2014.

[34] J. L. Devore, *Probability and Statistics for Engineering and the Sciences.* Cengage learning, 2011.

[35] ITU-R, "Guidelines for evaluation of radio interface technologies for imt-advanced," Report ITU-R M.2135-1, 2016.

[36] R. Bellman, "A Markovian decision process," DTIC, Tech. Rep., 1957.

[37] R. Howard, *Dynamic Programming and Markov Processes.* Technology Press-Wiley, 1960.

[38] S. Lippman, "Applying a New Device in the Optimization of Exponential Queuing Systems," *Operation Research*, vol. 23, no. 4, pp. 687–710, Aug. 1975.

[39] H. Tijms, *A First Course in Stochastic Models.* J. Wiley & Sons, 2003.

[40] C. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[41] E. Even-Dar and Y. Mansour, "Learning rates for Q-learning," *Journal of Machine Learning Research*, vol. 5, pp. 1–25, Dec. 2003.

[42] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv:1312.5602*, 2013.

[43] ——, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, Feb. 2015.

[44] C. Jiang, H. Zhang, Y. Ren, Z. Han, K.-C. Chen, and L. Hanzo, "Machine Learning Paradigms for Next-Generation Wireless Networks," *IEEE Wireless Commun.*, vol. 24, no. 2, pp. 98–105, Dec. 2017.

[45] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial Intelligence: a Modern Approach.* Prentice hall Upper Saddle River, 2003, vol. 2, no. 9.

[46] F. S. Melo and M. I. Ribeiro, "Q-learning with linear function approximation," in *Proc. COLT*, San Diego, USA, Jun. 2007, pp. 308–322.

[47] F. Chollet, *Deep learning with Python.* Manning Publications Co, 2018.

[48] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[49] Tieleman, Tijmen and Hinton, Geoffery, "RMSprop gradient optimization," http://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf, 2014.

[50] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, May 2015.

[51] 3GPP, "Technical Specification Group Services ans System Aspects; Policy and charging control architecture," TS 23.203, v15.1.0, Dec. 2017.

[52] A. de la Oliva *et al.*, "5G-TRANSFORMER: Slicing and Orchestrating Transport Networks for Industry Verticals," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 78–84, Aug. 2018.

[53] 5G-PPP, "The 5G Infrastructure Association. Pre-structuring Model, version 2.0," Nov. 2017.

[54] ETSI, NFVGS, "Network Function Virtualization (NFV) Management and Orchestration," *NFV-MAN*, vol. 1, Dec. 2014.

[55] ETSI, "OSM release FOUR technical overview," May 2018.

[56] The Linux Foundation, "ONAP, Open Network Automation Framework." [Online]. Available: https://www.onap.org

[57] J. Wang *et al.*, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *Proc. of IEEE INFOCOM*, May 2017, pp. 1–9.

[58] M. C. Valenti, S. Talarico, and P. Rost, "The role of computational outage in dense cloud-based centralized radio access networks," in *Proc. of IEEE GLOBECOM*, Dec. 2014, pp. 1466–1472.

[59] N. Nikaein *et al.*, "OpenAirInterface: A Flexible Platform for 5G Research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, Oct. 2014.

[60] C. Zhang *et al.*, "Deep Learning in Mobile and Wireless Networking: A Survey," *arXiv:1803.04311 [cs.NI]*, Mar. 2018.

[61] M. Wang *et al.*, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, vol. 32, no. 2, pp. 92–99, March 2018.

[62] J. X. Salvat *et al.*, "Overbooking Network Slices Through Yield-driven End-to-end Orchestration," in *Proc. ACM CoNEXT*, 2018, pp. 353–365.

[63] J. J. Ayala *et al.*, "vrAIn: A Deep Learning Approach Tailoring Computing and Radio Resources in Virtualized RANs." Los Cabos, Mexico: ACM MobiCom, Nov. 2019.

[64] C. Gutterman *et al.*, "RAN resource usage prediction for a 5G slice broker," in *Proc. of ACM MOBIHOC*, New York, NY, USA, 2019.

[65] M. Joshi and T. H. Hadi, "A Review of Network Traffic Analysis and Prediction Techniques," *arXiv:1507.05722 [cs.NI]*, Jul. 2015.

[66] F. Xu *et al.*, "Big Data Driven Mobile Traffic Understanding and Forecasting: A Time Series Approach," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 796–805, Sep. 2016.

[67] M. Zhang *et al.*, "Understanding Urban Dynamics From Massive Mobile Traffic Data," *IEEE Transactions on Big Data*, pp. 1–1, 2017.

[68] S. T. Au *et al.*, "Automatic forecasting of double seasonal time series with applications on mobility network traffic prediction," *JSM Proceedings, Business and Economic Statistics Section*, Jul. 2011.

[69] R. Li *et al.*, "The prediction analysis of cellular radio access network traffic: From entropy theory to networking practice," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 234–240, Jun. 2014.

[70] M. Z. Shafiq *et al.*, "Characterizing and modeling internet traffic dynamics of cellular devices," in *ACM SIGMETRICS*, San Jose, California, USA, Jun. 2011, p. 305.

[71] A. Y. Nikravesh *et al.*, "An Experimental Investigation of Mobile Network Traffic Prediction Accuracy," *Services Transactions on Big Data*, vol. 3, no. 1, pp. 1–16, Jan. 2016.

[72] C. Zhang and P. Patras, "Long-Term Mobile Traffic Forecasting Using Deep Spatio-Temporal Neural Networks," in *ACM MobiHoc*, Los Angeles, CA, USA, Jun. 2018, pp. 231–240.

[73] S. Ntalampiras and M. Fiore, "Forecasting mobile service demands for anticipatory MEC," in *IEEE WoWMoM*, Chania, Greece, Jun. 2018, pp. 14–19.

[74] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, Sep. 2014.

[75] Y. LeCun, "Generalization and network design strategies," *Connectionism in perspective*, pp. 143–155, Jun. 1989.

[76] C. Szegedy *et al.*, "Going deeper with convolutions," in *IEEE CVPR*, Jun. 2015, pp. 1–9.

[77] G. E. Dahl *et al.*, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *IEEE ICASSP*, May 2013.

[78] M. **W**. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron) - A review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, Aug. 1998.

[79] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, Dec. 2014.

[80] A. Furno *et al.*, "A Tale of Ten Cities: Characterizing Signatures of Mobile Traffic in Urban Areas," *IEEE Transactions on Mobile Computing*, vol. 16, no. 10, pp. 2682–2696, Oct. 2017.

[81] J. Paparrizos and L. Gravano, "k-Shape: Efficient and Accurate Clustering of Time Series," in *ACM SIGMOD*, Jun. 2015, pp. 1855–1870.

[82] I. Borg and P. Groenen, "Modern Multidimensional Scaling: Theory and Applications," *Journal of Educational Measurement*, vol. 40, no. 3, pp. 277–280, Sep. 2003.

[83] H. **W**. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, Mar. 1955.

[84] D. K. Krishnappa *et al.*, "DASHing YouTube: An analysis of using DASH in YouTube video service," in *Proc. of IEEE LCN*. Sydney, NSW, Australia: IEEE, 2013, pp. 407–415.

[85] G. Dimopoulos *et al.*, "Measuring video QoE from encrypted traffic," in *Proc. of ACM IMC*. Santa Monica, CA, USA: ACM, 2016, pp. 513–526.

[86] P. Casas *et al.*, "Qomosn-on the analysis of traffic and quality of experience in mobile online social networks," in *Proc. of IEEE EuCNC*, Paris, France, 2015, pp. 471–475.

[87] C. Marquez *et al.*, "How Should I Slice My Network?: A Multi-Service Empirical Evaluation of Resource Sharing Efficiency." New Delhi, India: ACM MobiCom, Nov. 2018, pp. 191–206.

[88] ——, "Not All Apps Are Created Equal: Analysis of Spatiotemporal Heterogeneity in Nationwide Mobile Service Usage," in *ACM CoNEXT*. ACM, 2017, pp. 180–186.

[89] J. Gil Herrera *et al.*, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[90] J.-J. Kuo *et al.*, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture," in *IEEE INFOCOM*, Atlanta, GA, USA, May 2017, pp. 1–9.

[91] P. Caballero *et al.*, "Network slicing games: Enabling customization in multi-tenant networks," in *IEEE INFOCOM 2017*, May 2017.

[92] B. Koley, "The zero touch network," in *IEEE CNSM*, 2016.

[93] European Telecommunications Standards Institute (ETSI), "ZSM Scenarios and key requirements," ETSI ISG ZSM 001, Oct. 2018.

[94] A. Garcia-Saavedra and others, "FluidRAN: Optimized vRAN/MEC Orchestration," in *IEEE INFOCOM 2018*, Apr. 2018.

[95] J. Kim *et al.*, "3GPP SA2 architecture and functions for 5G mobile communication system," *ICT Express*, vol. 3, no. 1, pp. 1–8, 2017.

[96] L. Peterson, "Cord: Central office re-architected as a datacenter," in *IEEE Softw. Defined Netw. Newslett.*, 2015.

[97] M. Mao and M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud," in *IEEE CLOUD 2012*, Jun. 2012.

[98] 5G-CORAL, "Refined design of 5G-CORAL orchestration and control system and future directions," D3.2, May 2019.

[99] S. Gonzalez *et al.*, "Towards a Resilient Openflow Channel Through MPTCP," in *IEEE BSMB 2018*, Jun. 2018.

[100] L. Zanzi and others, "OVNES: Demonstrating 5G network slicing overbooking on real deployments," in *Proc. of IEEE INFOCOM WKSHPS*, Honolulu, HI, USA, Apr. 2018, pp. 1–2.

[101] C. Zhang and others, "Multi-Service Mobile Traffic Forecasting via Convolutional Long Short-Term Memories," in *Proc. of IEEE M&N 2019*, Catania, Italy, Jun. 2019, pp. 1–6.

[102] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Proc. of ICML*, New York, NY, USA, Jun. 2016, pp. 1050–1059.

[103] M. J. Powell, "The BOBYQA algorithm for bound constrained optimization without derivatives," *Technical report*, 2009.

[104] W. H. Press and others, *Numerical Recipes in FORTRAN; The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1993.