UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

Gestion Flexible des Ressources dans Les Réseaux de Nouvelle Génération avec SDN

Flexible Resource Management in Next-Generation Networks with SDN

Thèse de doctorat
Specialité: génie électrique

Abderrahime Filali

Sherbrooke (Québec) Canada

Décembre 2021

JURY MEMBERS

Soumaya Cherkaoui

Supervisor

Hossam Hassanein

Examiner

Rodolfo Coutinho

Examiner

Eric Plourde

Examiner

RÉSUMÉ

La 5G et au-delà de la 5G/6G sont censées dessiner la future croissance économique de multiples industries verticales en fournissant l'infrastructure réseau nécessaire pour permettre l'innovation et la création de nouveaux modèles économiques. Elles permettent d'offrir un large spectre de services, à savoir des débits de données plus élevés, une latence ultra-faible et une fiabilité élevée. Pour tenir leurs promesses, la 5G et au-delà de la-5G/6G s'appuient sur le réseau défini par logiciel (SDN), l'informatique en périphérie et le découpage du réseau d'accès (RAN). Dans cette thèse, nous visons à utiliser le SDN en tant qu'outil clé pour améliorer la gestion des ressources dans les réseaux de nouvelle génération.

Le SDN permet une gestion programmable des ressources informatiques en périphérie et une orchestration dynamique de découpage du RAN. Cependant, atteindre une performance efficace en se basant sur le SDN est une tâche difficile due aux fluctuations permanentes du trafic dans les réseaux de nouvelle génération et aux exigences de qualité de service diversifiées des applications émergentes. Pour atteindre notre objectif, nous abordons le problème de l'équilibrage de charge dans les architectures SDN distribuées, et nous optimisons le découpage du RAN des ressources de communication et de calcul à la périphérie du réseau.

Dans la première partie de cette thèse, nous présentons une approche proactive pour équilibrer la charge dans un plan de contrôle SDN distribué en utilisant le mécanisme de migration des composants du plan de données. Tout d'abord, nous proposons des modèles pour prédire la charge des contrôleurs SDN à long terme. En utilisant ces modèles, nous pouvons détecter de manière préemptive si la charge sera déséquilibrée dans le plan de contrôle et, ainsi, programmer des opérations de migration à l'avance. Ensuite, nous améliorons les performances des opérations de migration en optimisant le compromis entre un facteur d'équilibrage de charge et le coût des opérations de migration. Cette approche proactive d'équilibrage de charge permet non seulement d'éviter la surcharge des contrôleurs SDN, mais aussi de choisir judicieusement le composant du plan de données à migrer et l'endroit où la migration devrait avoir lieu.

Dans la deuxième partie de cette thèse, nous proposons deux mécanismes de découpage du RAN qui allouent efficacement les ressources de communication et de calcul à la périphérie des réseaux. Le premier mécanisme de découpage du RAN effectue l'allocation des blocs de ressources radio (RBs) aux utilisateurs finaux en deux échelles de temps, à savoir dans une échelle de temps large et dans une échelle de temps courte. Dans l'échelle de temps large, un contrôleur SDN attribue à chaque station de base un certain nombre de RB à partir d'un pool de RB radio partagé, en fonction de ses besoins en termes de délai et de débit. Dans l'échelle de temps courte, chaque station de base attribue ses ressources disponibles à ses utilisateurs finaux et demande, si nécessaire, des ressources supplémentaires aux stations de base adjacentes. Le deuxième mécanisme de découpage du RAN alloue conjointement les RB et les ressources de calcul disponibles dans les serveurs de l'informatique en périphérie

en se basant sur une architecture RAN ouverte. Nous développons, pour les mécanismes de découpage du RAN proposés, des algorithmes d'apprentissage par renforcement et d'apprentissage par renforcement profond pour allouer dynamiquement les ressources du RAN.

Mots-clés : 5G et au-delà de la-5G/6G, réseau défini par logiciel, informatique en périphérie, découpage du réseau d'accès, apprentissage par renforcement, apprentissage par renforcement profond, haut débit mobile amélioré, communication ultra-fiable à faible latence.

ABSTRACT

5G and beyond-5G/6G are expected to shape the future economic growth of multiple vertical industries by providing the network infrastructure required to enable innovation and new business models. They have the potential to offer a wide spectrum of services, namely higher data rates, ultra-low latency, and high reliability. To achieve their promises, 5G and beyond-5G/6G rely on software-defined networking (SDN), edge computing, and radio access network (RAN) slicing technologies. In this thesis, we aim to use SDN as a key enabler to enhance resource management in next-generation networks.

SDN allows programmable management of edge computing resources and dynamic orchestration of RAN slicing. However, achieving efficient performance based on SDN capabilities is a challenging task due to the permanent fluctuations of traffic in next-generation networks and the diversified quality of service requirements of emerging applications. Toward our objective, we address the load balancing problem in distributed SDN architectures, and we optimize the RAN slicing of communication and computation resources in the edge of the network.

In the first part of this thesis, we present a proactive approach to balance the load in a distributed SDN control plane using the data plane component migration mechanism. First, we propose prediction models that forecast the load of SDN controllers in the long term. By using these models, we can preemptively detect whether the load will be unbalanced in the control plane and, thus, schedule migration operations in advance. Second, we improve the migration operation performance by optimizing the tradeoff between a load balancing factor and the cost of migration operations. This proactive load balancing approach not only avoids SDN controllers from being overloaded, but also allows a judicious selection of which data plane component should be migrated and where the migration should happen.

In the second part of this thesis, we propose two RAN slicing schemes that efficiently allocate the communication and the computation resources in the edge of the network. The first RAN slicing scheme performs the allocation of radio resource blocks (RBs) to end-users in two time-scales, namely in a large time-scale and in a small time-scale. In the large time-scale, an SDN controller allocates to each base station a number of RBs from a shared radio RBs pool, according to its requirements in terms of delay and data rate. In the short time-scale, each base station assigns its available resources to its end-users and requests, if needed, additional resources from adjacent base stations. The second RAN slicing scheme jointly allocates the RBs and computation resources available in edge computing servers based on an open RAN architecture. We develop, for the proposed RAN slicing schemes, reinforcement learning and deep reinforcement learning algorithms to dynamically allocate RAN resources.

Keywords: 5G and beyond-5G/6G, software-defined networking, edge computing, radio access network slicing, reinforcement learning, deep reinforcement learning, enhanced mobile broadband, ultra-reliable low-latency communication.

To my beloved parents, my sister, and my brother for their encouragement and endless support.

ACKNOWLEDGEMENTS

I have been able to complete this thesis with the help and active support of several persons.

First and foremost, I am extremely grateful to my research director, Prof. Soumaya Cherkaoui, for her permanent support and invaluable advice during my Ph.D. Thank you for your constant encouragement and for always being willing and enthusiastic to enrich and complete this research project.

I would like to thank Prof. Abdellatif Kobbane for his mentoring and expert advice. Thank you for the opportunity you offered me, and for accompanying me in my first research steps.

I would like to thank my committee members, not only for their time and extreme patience, but also for their insightful comments and suggestions.

I would also like to thank the co-authors of my papers for their great contributions: Dr. Zoubeir Mlika, thank you for your help, support, and guidance. Dr. Boubakr Nour, thank you for sharing with me your versatile and diversified knowledge. Amine Abouaomar, my first lab mate, thank you for being there from the beginning, for bringing your boundless work energy and the great camaraderie we have shared over these years.

Oussama, Afaf and Hajar! Thank you for being wonderful colleagues and more than just friends. Thank you for the fantastic moments spent together inside and outside the lab, for your incredible spirit of solidarity that helped me through the stressful times. To all my friends outside the lab, thank you for your motivation and for all the fun we had together.

Lastly and most importantly, my deepest and most sincere thanks to my family who deserve endless gratitude. To my father Abdellatif, thank you for your unconditional trust, your timely encouragement, and your wise counsel. To my mother Fatna, who was my first teacher in primary school, thank you for your unwavering emotional support, your countless sacrifices, and for helping me to become the person I am today. To my brother Abdelilah and my sister Salma, thank you for your love, your outstanding support, and your pleasant spirit.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronyme	Définition
3GPP	3rd generation partnership project
ACF	Autocorrelation function
AI	Artificial intelligence
AIC	Akaike information criterion
AMF	Access and mobility management function
AR	Autoregressive
ARIMA	Autoregressive integrated moving average
ARMA	Autoregressive moving average
AWGN	Additive white Gaussian noise
C-RAN	cloud radio access network
DNS	Domain name system
DQL	Deep Q-learning
eMBB	enhanced mobile broadband
ETSI	European telecommunications standards institute
HTTP	Hypertext transfer protocol
IoT	Internet of things
LBDSA	Load balancing for delay sensitive applications
LSTM	Long short-term memory
MA	Moving average
MAE	Mean absolute error
MAPE	Mean absolute percentage error
MCBLB	Migration competency-based load balancing
MDP	Markov decision process
MEC	Multi-access edge computing
MIMO	Multiple input and multiple output
ML	Machine learning
mMTC	massive machine-type communications
NFV	Network function virtualization
NR	New radio
OFDMA	Orthogonal frequency division multiple access
PACF	Partial autocorrelation function
PCF	Policy control function
QoS	Quality of service
RAN	Radio access network
RIC	RAN intelligent controller
RL	Reinforcement Learning
RMSE	Root mean square error
RNN	Recurrent neural network
SDN	Software defined networking
SFC	Service function chaining

SLA	Service-level agreement
SMCLBRT	SDN multiple controller load-balancing strategy based on response time
SMF	Session management function
TCP	Transmission control protocol
UDP	User datagram protocol
UPF	User plane function
URLLC	ultra-reliable low-latency communications
VNF	Virtualized network function
VoIP	Voice over Internet protocol
WSLS	Win-stay, lose-shift

CHAPTER 1

INTRODUCTION

While 5G — the fifth generation of cellular network technology — is undergoing significant deployment growth, beyond-5G and 6G networks are already taking shape on the horizon. These next-generation networks are driving many vertical industries in the midst of a technological transformation. Indeed, analysts from Ericsson report that 5G coverage is expected to reach 60% of the world's population by 2026 [2]. In addition, according to 5G Americas [1], the forecast number of 5G connections worldwide will grow rapidly to 3.4 billion by 2025, where North America, North-East Asia, and Western Europe will have the most 5G subscriptions. We can see in Figure 1.1 that the number of 5G connections will increase roughly five and a half times between 2021 and 2025, from 619 million to 3.4 billion - with an average growth rate of 41%. With their ability to support various cutting-edge applications such as autonomous vehicles, virtual reality, automated factories, and eHealth care, next-generation networks are paving the way for telecom operators to collaborate with new industry sectors as business partners. This partnership forms the foundation to establish smart cities with a new ecosystem where all stakeholders can share the benefits.

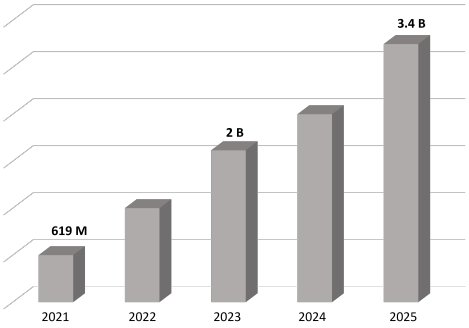


Figure 1.1 5G connections forecast 2021-2025 [1]

To realize a fruitful cooperation with different vertical industries, 5G and beyond-5G/6G networks should provide the required network infrastructure that meets the diversified requirements of their applications. For instance, 5G is expected to support three main heterogeneous services [3], namely ultra-reliable low-latency communications (URLLC), enhanced mobile broadband (eMBB) services, and massive machine-type communication (mMTC) services. The URLLC services support low-latency and high reliability applica-

tions, such as self-driving transport and remote surgery. The eMBB services provide very high data rates for applications such as high-resolution video streaming and augmented reality. The mMTC services enable the connection of a huge number of devices, *i.e.,* Internet of things (IoT) devices.

In order to deliver this wide range of applications in a more cost-effective and high-performance manner, next-generation networks rely on key enablers, including edge computing [4] and network slicing [5] technologies. Edge computing enables data processing, storage, and analysis to be performed at the edge of the network, very close to where the data is generated, to reduce response latency, allow near real-time analysis, and decrease demands on cloud and data center resources. Network slicing allows multiple logical networks, *i.e.,* network slices, to be created and to operate on a common and shared physical infrastructure. Each network slice is tailored to satisfy the specific requirements of an application from the radio access network (RAN) to the core network. In particular, RAN slicing is an essential part of the end-to-end network slicing to enable a more granular differentiation of the services that will be available in the network.

To efficiently exploit the capabilities of these technologies, network operators need to automate the management of network resources and the monitoring of network performance. Software defined networking (SDN) technology has been presented as an innovative paradigm to revolutionize traditional network architecture [6]. The idea behind SDN is to decouple the control plane and the data plane of the network by removing the network intelligence from the data plane components, and then centralizing it within a network entity, known as the SDN controller. The latter maintains a global view of the network to manage the data plane components, such as virtualized switches, base stations, and virtualized network functions (VNFs). In addition, the deployment of artificial intelligence (AI) and machine learning (ML) techniques is easier in the SDN-based network architecture since SDN provides logically centralized control, a global view of the network, and a software-based resource configuration. As a result, next-generation network management decisions will become more intelligent and robust.

Accordingly, SDN technology allows programmable and dynamic orchestration of 5G and beyond-5G/6G networks. However, achieving efficient performance in edge computing resource management and network resource slicing based on SDN capabilities is a challenging task. This is due to the permanent fluctuations of data traffic in next-generation networks and the diversified quality of service (QoS) requirements of their emerging applications.

1.1 Problem Statement

As described previously, SDN enables the automation of network supervision. In particular, it brings agility to network resource management by enabling policy-driven network monitoring. However, some issues related to leveraging SDN in next-generation network resource management continue to attract researchers in both academic and industrial fields.

To avoid a single point of failure and to ensure the availability of SDN controllers, the SDN control plane architecture is designed in a distributed manner [7]. A distributed architecture of the SDN control plane consists of several SDN controllers, where each controller manages a subset of data plane components. This distributed architecture of the SDN control plane is currently being adopted in large-scale networks as a key solution to guarantee scalability in handling the data plane components. However, such an architecture gives rise to the load balancing problem in the control plane. Indeed, since network traffic is highly dynamic, each SDN controller may receive a huge number of requests that it should process, and then make appropriate decisions quickly. These permanent fluctuations in network traffic can make some controllers overloaded while others will be underloaded. When a controller is overloaded, its processing time for received requests increases. Accordingly, the performance of the overloaded controller and the control plane in general decreases. Load balancing between controllers is therefore required to maintain efficient performance of the SDN control plane in managing the entire network.

Network operators leverage the programmability provided by the SDN technology to dynamically manage the allocation of RAN resources to RAN slices. In an SDN-enabled RAN architecture, an SDN controller manages multiple base stations and multi-access edge computing (MEC) servers as data plane components. The base stations provide communication resources, and the MEC servers bring computation resources near the end-users. Under the supervision of the SDN controller, the allocation of RAN resources, namely communication and computation resources, can be performed automatically. However, RAN slicing operations in an SDN control domain are not obvious. Indeed, this is due to several constraints: 1) the RAN traffic is highly dynamic, 2) the transmission conditions on wireless channels vary permanently, 3) and the QoS requirements of emerging applications are diversified. Therefore, when the allocation of RAN resources to end-users is entrusted exclusively to the SDN controller, several problems will occur. These problems can be the increase in the signaling overhead between the SDN controller and the base stations or MEC servers. In addition, the RAN slicing operations can be performed with high latency, which is not sustainable for delay-sensitive applications.

In light of these challenging problems, this thesis raises the following fundamental research question:

> *How to enable efficient and flexible management of next-generation network resources with SDN?*

1.2 Objective

The main objective of this thesis is to leverage SDN as a key enabler to improve the management of next-generation network resources. To achieve this objective, we tackle the two sub-problems discussed previously, namely load balancing in the distributed SDN control plane and RAN slicing of communication and computation resources at the edge of the network. Accordingly, we divide the main objective into the following intermediate objectives.

- Prevent the load imbalance in the SDN control plane
  The first intermediate objective is to avoid load imbalance problems in the distributed SDN control plane. For this reason, we need to define the load of an SDN controller. Then, we have to preemptively detect if the load will be unbalanced in the SDN control plane. When a load imbalance occurs, we need to balance the load between controllers based on the data plane component migration approach. The corresponding research questions for this intermediate objective are as follows:
  - How to define the load of an SDN controller?
  - How to efficiently predict if the load will be unbalanced in the SDN control plane?
- Improve the data plane component migration performance
  The second intermediate objective is to improve the data plane component migration performance. A migration operation of a data plane component involves changing its SDN controller. When the load of the control plane becomes unbalanced, some data plane components under the control of the overloaded SDN controllers migrate to be under the control of the underloaded SDN controllers. The corresponding research questions for this intermediate objective are as follows:
  - Which metrics should be used to define the migration operation?
  - Which data plane components should be migrated?
  - Where should the migration operations happen?
- Enhance the SDN-based RAN slicing performance
  The last intermediate objective is to enhance the slicing performance of the RAN

resources, namely communication and computation resources. We propose RAN slicing approaches that allow allocating resources to RAN slices at multiple levels. These approaches avoid the problems of a centralized RAN slicing mechanism that relies only on the SDN controller and meet the requirements of emerging applications considering the limited RAN resources. The corresponding research questions for this intermediate objective are as follows:

- How to fairly partition the RAN resources between the slices?
- How to ensure that heterogeneous RAN slices are satisfied in terms of quality of service?
- How to adapt the proposed resource allocation approaches to the variable and highly dynamic nature of the RAN environment?

1.3 Contributions and Originality

Load balancing in the SDN control plane and RAN slicing of communication and computational resources are important aspects for efficient utilization of SDN as a key enabler to improve the management of next-generation network resources. Therefore, this thesis proposes three approaches to deal with these aspects. In the first approach, we present an efficient load balancing scheme of the distributed SDN control plane based on the data plane component migration mechanism. In the second approach, we propose a new RAN slicing scheme that optimizes the allocation of RAN communication resources to meet the requirements of emerging applications. In the last approach, we design a RAN slicing scheme that jointly optimizes the allocation of communication and computation resources based on the open radio access network (O-RAN) architecture.

The first approach is divided into two main parts. In the first part, we propose prediction models that forecast the load of SDN controllers in long term. These long-term predictions of the controller load enable preemptive detection of whether the load will be unbalanced in a distributed SDN control plane. Then, data plane migration operations can be scheduled in advance. This proactive load balancing mechanism not only avoids SDN controllers from being overloaded, but also allows a judicious selection of which data plane component should be migrated and where the migration should happen. Accordingly, in the second part, we solve the load balancing problem of the distributed SDN control plane by optimizing the tradeoff between a load balancing factor and the cost of migration operations. The load balancing factor is defined based on the response time of controllers, and it measures how fairly the load is partitioned among the SDN controllers. On the other hand, the migration cost of a data plane migration operation is defined based on the time

required to execute a migration protocol. The main contributions of this approach can be summarized as follows:

- We build and evaluate two prediction models. The first model is a traditional stochastic model called autoregressive integrated moving average (ARIMA). The second model is a machine learning prediction model known as long short-term memory (LSTM).
- We provide a performance analysis comparing the accuracy of the short and long-term SDN controllers' load predictions of the ARIMA and LSTM models.
- We use mathematical programming techniques to formulate the SDN load balancing problem as an optimization program where the objective is to minimize, through migration operations, the load balancing factor considering the migration operation cost.
- We prove that the SDN load balancing optimization problem is NP-complete by reducing the partition problem to it.
- We propose a reinforcement learning (RL) algorithm to solve the formulated optimization problem.
- We assess the performance of the proposed RL algorithm against the optimal solution and load balancing algorithms from the literature.

The second approach proposes an SDN-based RAN slicing mechanism that allocates the communication resources to end-users. The communication resources considered in this approach is the radio resource blocks (RBs). The proposed mechanism performs the allocation of RBs to end-users in two time-scales, namely a large time-scale and a small time-scale. In a large time-scale, an SDN controller allocates to each base station a number of RBs from a shared radio RBs pool, based on its requirements. In a short time-scale, each base station assigns, based on the pre-allocated RBs, to each of its associated end-users the needed RBs to meet their QoS requirements. If the RBs allocated by the SDN controller are not sufficient for a base station, the latter can request additional RBs from other base stations instead of waiting for the next large time-scale RB reservation update. In this approach, we consider two types of RAN slices, including the eMBB service and the URLLC service. We solve this RAN slicing problem by optimizing the achieved data rate of eMBB and URLLC end-users. The key contributions of this approach can be resumed as follows:

- We formulate the RB allocation problem as an optimization program where the objective is to maximize the total achievable data rate of eMBB and URLLC end-users.

- We prove the NP-Hardness of the formulated optimization problem by reducing the 0-1 knapsack problem to it.
- We model each RB allocation time-scale level as a Markov decision process (MDP). Specifically, we model the large time-scale RB allocation level as a single-agent MDP and the short time-scale RB allocation level as multi-agent MDP.
- We propose an RL algorithm and a deep Q-learning algorithm to solve the single-MDP and the multi-agent MDP, respectively.
- We evaluate the performance of the proposed RAN slicing mechanism against a benchmark algorithm.

The last approach presents a RAN slicing mechanism to jointly allocate the communication and computation resources in an O-RAN architecture. In this approach, the communication and computation resources considered are the RBs of base stations and the CPU cores of the MEC server, respectively. The proposed mechanism performs the RAN slicing into two levels. In the first RAN slicing level, each base station allocates its RBs to its end-devices that need to offload their computational tasks to more resourceful units, i.e., MEC servers. In the second RAN slicing level, we allocate the computation resources to compute the tasks offloaded by the end devices. In this approach, we leverage the O-RAN architecture to provide an implementation of the proposed RAN slicing mechanism. O-RAN architecture introduces the hierarchical RAN intelligent controller (RIC), including non-real-time RIC (non-RT) and near-real-time RIC (near-RT) where ML/AI algorithms are integrated to enable RAN programmability. The non-RT RIC handles the heaviest RAN functions while the near-RT RIC executes critical RAN functions. Therefore, non-RT and near-RT RICs can be leveraged to dynamically allocate the RAN communication and computation resources using ML/AI capabilities. The main contributions of this approach can be summarized as follows:

- We model, for each RAN slicing level, the resource slicing problem as a single-agent MDP.
- We develop, for each RAN slicing level, a DQL algorithm to solve each single-agent MDP.
- We define the role of the non-RT and near-RT RICs of the O-RAN architecture in performing slicing operations.
- We describe where and how the training phase and the implementation phase of each DQL can be performed in an O-RAN architecture.
- We conduct extensive simulations to demonstrate the efficiency of the proposed RAN slicing mechanism in meeting the desired QoS requirements.

1.3.1 List of Papers

The research conducted during this doctoral project resulted in high-quality papers, which are published and submitted to renowned and highly respected conferences and journals. The list of the papers is as follows:

Published papers:

1. **A. Filali**, S. Cherkaoui and A. Kobbane, "Prediction-Based Switch Migration Scheduling for SDN Load Balancing," ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019, pp. 1-6, 2019 (Chapter 3)
2. **A. Filali**, A. Abouaomar, S. Cherkaoui, A. Kobbane and M. Guizani, "Multi-Access Edge Computing: A Survey," in IEEE Access, vol. 8, pp. 197017-197046, 2020.
3. **A. Filali**, Z. Mlika, S. Cherkaoui and A. Kobbane, "Preemptive SDN Load Balancing With Machine Learning for Delay Sensitive Applications," in IEEE Transactions on Vehicular Technology, vol. 69, no. 12, pp. 15947-15963, Dec. 2020. (Chapter 4)

Papers under revision:

1. **A. Filali**, Z. Mlika, S. Cherkaoui and A. Kobbane, "Dynamic SDN-based Radio Access Network Slicing with Deep Reinforcement Learning forURLLC and eMBB Services," IEEE Transactions on Network Science and Engineering (Chapter 5)
2. **A. Filali**, B. Nour, S. Cherkaoui, and A. Kobbane, "Joint Slicing of Communication and Computation RAN Resources Using Deep Reinforcement Learning for URLLC Services," in IEEE Communications Standards Magazine. (Chapter 6)

1.4 Thesis Plan

This thesis is based on manuscripts (Article-based). Chapter 2 conducts a literature review describing recent advances related to our research project. Chapter 3 and chapter 4 present our contribution related to the load balancing in the distributed SDN control plane. In chapter 3, we propose a long-term prediction model for the SDN controller load based on ARIMA, an optimization model to balance the load in the control plane, and a heuristic algorithm to solve the optimization problem. In chapter 4, we propose two long-term prediction models based on ARIMA and LSTM respectively, perform a comparative study to evaluate their accuracy, formulate an optimization model to balance the load in the control plane, and develop an algorithm to solve the optimization problem. Chapter 5 and chapter 6 present our contribution to improving the performance of RAN slicing. In chapter 5, we describe the proposed scheme for allocating communication resources, the optimization model formulated to allocate communication resources, and the algorithms developed to solve the optimization problem. In chapter 6, we present the proposed

scheme for the joint allocation of communication and computation resources, the RAN architecture adapted for the proposed scheme, and the algorithms developed to perform the allocation of communication and computation resources.

CHAPTER 2

State of the Art

SDN technology is a key component of the current 5G network architecture as well as future telecommunication network architectures (*i.e.,* beyond-5G and 6G networks). It represents a promising technology that provides more programmability to operators in managing their resources, especially at the edge of the network. In addition, SDN is being leveraged as an enabler of other pillar technologies for next-generation networks, namely edge computing and network slicing. Therefore, improving next-generation resource management using SDN technology is increasingly attracting researchers in both academic and industrial fields.

In this chapter, we conduct a literature review describing recent advances related to our research project. We start by providing, in section 2.1, an overview of SDN technology and its architecture, and highlight the potential opportunities that SDN brings to 5G and beyond-5G/6G networks. In section 2.2, we initially introduce the edge computing paradigm and elaborate on its role in 5G and beyond-5G/6G networks. Then, we emphasize the capabilities of SDN that enhance the edge computing operation. In section 2.3, we review recent approaches proposed to address the load balancing problem in the SDN control plane. In section 2.4, we provide an overview of the network slicing paradigm, its main use cases in 5G and beyond-5G/6G networks, and the role of SDN in improving its performances, with a particular focus on RAN slicing. Finally, in section 2.5, we present recent approaches proposed to overcome the challenges of RAN slicing.

2.1 Software-defined networking

SDN was born with the idea of bringing the concepts of programmability and virtualization to networks. Indeed, the static nature of the traditional network architecture model fails to support the exponential increase in traffic volume generated by modern services - mobile broadband, massive IoT, and critical business - that 5G and beyond-5G/6G networks is expected to provide to verticals. Therefore, operators need to adapt their networks automatically and in real-time to respond quickly to changing business requirements. Achieving this requires technology such as SDN, which provides operators with the flexibility and agility to manage their networks. SDN consists in decoupling the control plane and the data plane, and then centralizing the network control functions in an

external entity called an SDN controller. The latter interprets the policies received from an application layer and then enforces them by configuring and controlling the data plane components. In SDN, the application program interfaces (APIs) used to enable communication between the SDN controller and the control applications running in a higher-layer are known as northbound interfaces. On the other hand, the APIs that allow communication between the SDN controller and the data plane components are called the southbound interfaces [6].

2.1.1 SDN control plane architecture

Although SDN enables dynamic and programmable network configuration, the centralized control logic in a single SDN controller results in a lack of scalability, a single point of failure, and performance bottlenecks, especially when managing an increasing number of data plane components. Therefore, to overcome these issues and meet the real-world network requirements, *e.g.,* 5G, in terms of scalability, reliability, and performance, the SDN control plane architecture is designed to be physically distributed while maintaining a logically centralized network view [8]. In this architecture, multiple SDN controllers are used, which share the control load and synchronize information with each other to improve the performance and consistency of the entire network. There are two types of physically distributed and logically centralized architectures: i) hierarchical architecture and ii) flat architecture.

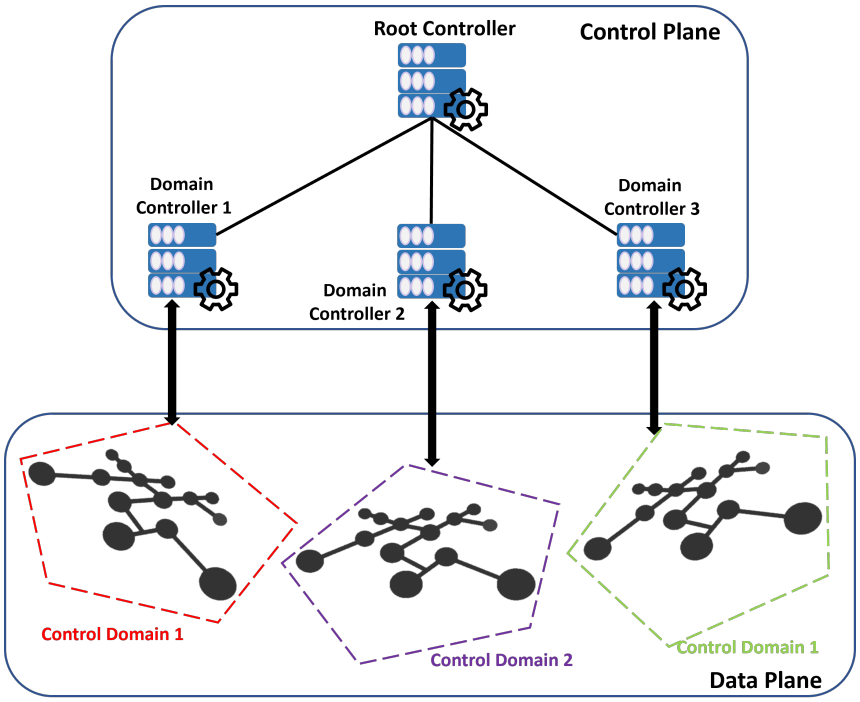


Figure 2.1 SDN control plane hierarchical architecture

In a hierarchical architecture, as shown in figure 2.1, controllers are arranged in a tree structure, in which there exist two types of controllers: the root controller and the domain controller. The domain controller maintains a local view of the network since it manages only a part of the data plane components that form an area called the control domain. The root controller manages the domain controllers. It has a global view of the network since the domain controllers communicate with it all the necessary information about their control domains. Note that in a hierarchical architecture, there is no communication between the domain controllers. B4 [9] is an example of SDN control plane hierarchical architecture.

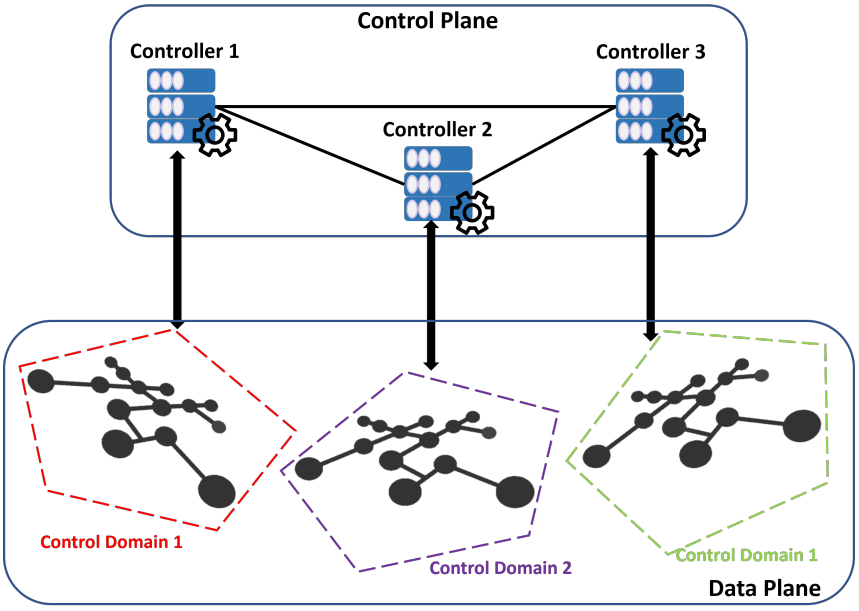


Figure 2.2 SDN control plane flat architecture

In a flat architecture, figure 2.2, controllers are at the same level, and each controller manages a single control domain. In order to preserve consistency, the controllers exchange network information with each other using east-west interfaces. The most popular examples of SDN control plane flat architecture are OpenDaylight (ODL) [10], Open Network Operating System (ONOS) [11], HyperFlow [12], and ONIX [13].

2.1.2 OpenFlow protocol

The OpenFlow protocol is a standard southbound communication interface between the control plane and the data plane in an SDN environment [14]. Indeed, the SDN controller uses the OpenFlow protocol to manage the data plane elements by 1) pushing down the directives received from the application layer to the data plane components, and 2) collecting information about the data plane network. The OpenFlow protocol defines three

types of exchanged messages between the two planes, including controller to data plane component messages, asynchronous messages, and symmetric messages.

- *Controller to data plane component messages:* are initiated by the controller and used to configure and program the data plane components, collect statistical information about the network, and inspect the state of data plane components.
- *Asynchronous messages:* are initiated by the data plane component without soliciting the controller and used to notify the latter of network events and any change in the data plane component state.
- *Symmetric messages:* are initiated by the controller or the data plane component without soliciting each other and used to maintain the communication channel active between the controller and the data plane component.

2.1.3 SDN for next generation networks

Next generation networks are expected to provide network operators with the ability to deliver faster and more responsive network services to their users and partners. However, the exponential increase in the size and complexity of networks, and the volume of data they are intended to convey have accelerated the adoption of SDN solutions. Designing the network according to the SDN architecture, *i.e.,* separating the network control plane from the data plane, brings several benefits that stem from its programmable management, easy reconfiguration, and on-demand resource allocation. To achieve this, several efforts have been conducted to incorporate the SDN concepts into the RAN and core networks.

- *SDN in RAN:* The work in [15] presents a software-defined RAN solution, which abstracts all the control operations in a logically centralized SDN controller. The latter manages the RAN infrastructure entities, *e.g.,* base stations, and orchestrates their operations, such as resource block allocation. Authors in [16] propose an implementation of SDN architecture for heterogeneous 5G RANs. Two main components are used, a centralized controller that maintains the network abstraction and a set of agents, each running on a RAN element where its actions are managed by the controller. OpenRAN 5G NR project [17] introduces a programmable and softwarized RAN, which is aligned with SDN principles by adopting the RAN functional split and proposing a hierarchical controller architecture known as RAN intelligent controllers (RICs). The split concept consists in defining which RAN functions should be centralized and those that should be distributed. RICs control, configure and optimize the RAN infrastructure with embedded machine learning (ML) and artificial intelligence (AI) tools. Authors in [18] leverage a hierarchical controller architecture to perform dynamic and automatic optimization of RAN resources, *e.g.,* spectrum,

computation, and transmission power. A central controller maintains a global view of the RAN, generates optimization problems based on the operator policies, and performs problem decomposition. Then, multiple local controllers, each controlling a base station, receive the programs from the central controller and solve them using appropriate algorithms.

- *SDN in core network:* In [19], SDN controllers are deployed in the core network to handle control plane signaling procedures, such as user attachment and mobility. For an optimal SDN-based core network design, three optimization models are proposed to place SDN controllers and map them with the required virtualized network functions (VNFs). The proposed optimization models take into account the latency requirements of the data plane and the control plane. The work in [20] focuses on separating the functions of the control plane and the data plane of some entities in the core network entities. Then, it proposes an approach for optimal placement of SDN controllers to reduce the communication cost between controllers as well as between data plane components and controllers. Authors in [21] propose an SDN-based core network architecture that relocates all core network control functions such as Access and Mobility Management Function (AMF), Session Management Function (SMF) and Policy Control Function (PCF) as applications on top of an SDN controller, which controls the core network data plane components such as User Plane Function (UPF). The proposed architecture reduces the end-to-end delay during two important control plane procedures, namely registration and handover, compared to the traditional core network architecture.

2.2 SDN for edge computing

2.2.1 Edge computing

With the massive deployment of IoT devices and the emergence of new applications that require a powerful real-time computing resource, the edge computing concept continues to surge in popularity as an innovative information and communication technology. Edge computing refers to the processing, analysis, and storage of data in proximity to where it is generated, such as IoT devices or edge servers. In other words, rather than sending the data to be processed on centralized cloud servers, which is costly in terms of latency and bandwidth, computing tasks take place closer to the end-devices. Therefore, edge computing improves network performance by reducing bandwidth utilization and latency, providing network scalability, and enhancing data security. Bandwidth utilization is reduced since most of the generated data by end-devices does not need to travel to the data centers to be processed, analyzed, and stored. Since data can be processed at the edge

of the network, low latency can be achieved. Therefore, delay-sensitive applications can receive notifications, decisions, and actions in near real-time. Network scalability is due to the ability to expand computing capacity at the network edge through a combination of edge-devices and edge servers, which is less expensive than dedicated data centers. Edge computing distributes data processing, analysis, storage among a wide range of end-device and edge servers, which avoid the unsecured transmission of data over long distances and through multi-hop paths to centralized servers.

The European Telecommunications Standards Institute (ETSI) is leading the definition of most technical and architectural specifications for the MEC concept through the MEC Industry Specification Group (ISG) [22]. The latter represents a collection of standards that help service providers to implement and deliver edge computing capabilities to their users. MEC enables communication service providers and third parties to have an open environment to flexibly develop and deploy a new range of services and applications. Indeed, being closer to end-devices and providing a fast and efficient way to act on data can benefit many use cases including industrial IoT [23], augmented and virtual reality [24], autonomous vehicles [25], and e-health [26].

2.2.2 Edge computing in next-generation networks

One of the main promises of next-generation networks is to reduce network round-trip times to provide low-latency communication services to delay-sensitive applications. To achieve this, they rely on MEC technology, which is considered the key enabler to meet the latency targets specified by the Third Generation Partnership Project (3GPP) [27]. With MEC, 5G and beyond-5G/6G are able to create new business opportunities in many vertical industries, such as private and public transportation, smart city management, agriculture, healthcare, manufacturing. In fact, Gartner, Inc. estimates that by 2025, 75% of enterprise-generated data will be created and processed outside a traditional centralized data center or cloud when 5G networks become widely deployed [28]. Accordingly, combining 5G and beyond-5G/6G with the MEC capabilities results in two major benefits:

- *Scalability:* involves adapting the network resources to the requirements of the connected devices to prevent the network from being overwhelmed, which decreases the quality of the provided services. MEC technology can ensure the network scalability since it supports network function virtualization (NFV). Therefore, additional network functions can be deployed quickly and flexibly, allowing multiple services to be initiated just in time. Through virtualization, the scalability of the 5G and beyond-5G/6G networks can be improved, making them more adept in handling the growth of new application and service requirements. In addition, it enables network

operators to efficiently and seamlessly provide new network services and applications on demand, without requiring additional hardware resources.

- *Innovation:* refers to the ability to develop and deploy new applications and services in a flexible manner due to the open and interoperable nature of the MEC environment. MEC avoids the restrictions imposed by propriety solutions that are driven by particular providers. It enables efficient integration of applications developed by various participants, namely traditional network operators or other service providers. For instance, network operators, with the support of third-party developers can develop and implement a variety of applications within an open and global standard, which is adopted by different verticals.

2.2.3 SDN for edge computing

In the 5G and beyond-5G/6G networks, edge computing is accelerating the creation of new applications with varying requirements, *e.g.,* latency-sensitive applications and bandwidth-intensive applications. These requirements dictate a joint management of edge computing resources, namely computing, communication, and storage resources. SDN technology promotes efficient utilization of edge computing resources by supporting policy-driven network supervision and implementing automatic orchestration of network resources [29]. Indeed, the logically centralized view of the SDN control plane allows operators to easily define the appropriate policies to manage all the available resources at the edge of the network. Then, they leverage the programmability of SDN to automate the enforcement, adaptation, and reconfiguration of these policies to quickly fulfill application requirements for computing, communication, and storage resources. Furthermore, SDN can manage the interaction between the edge and cloud computing environments [30]. The SDN controller can decide, based on its global view of the available resources at the edge of the network, whether a required service is suitable for edge computing resources or it should be sent to a more resourceful environment such as cloud computing. Accordingly, SDN improves the availability of edge computing by simplifying the management of its resources.

The cooperation between SDN and edge computing technologies has proven to enable more dynamism and operational efficiency in the 5G and beyond-5G/6G networks. Several works in the literature have shown the importance of SDN as a key enabler to improve the availability, flexibility, and resilience of the edge computing environment.

- *Availability:* Authors in [31] propose a MEC-based architecture to support the increased volume of data traffic and computing/storage tasks in autonomous vehicular networks. The proposed architecture incorporates a hierarchical SDN control plane, composed of edge controllers and a cloud controller, to improve resource utilization

and availability. An edge SDN controller allocates local computing/storage resources in MEC servers to connected autonomous vehicles, routes traffic, and performs radio resource pooling and slicing. The cloud SDN controller uses its global view of the network to manage the global traffic steering among cloud servers and make decisions about task migration between MEC servers based on resource availability at the edge. In [32], an SDN-based edge-cloud interaction is presented to handle the scheduling and routing of big data flows in an industrial IoT environment. The SDN control plane executes a flow scheduling and routing algorithm that optimizes the trade-offs between energy efficiency and bandwidth, and energy efficiency and latency. The results obtained show that the proposed SDN-based architecture supports the availability of the bandwidth resource. The work in [33] introduces an SDN-based management framework to manage the utilization of MEC computing resources, which are distributed across the RAN. An SDN orchestration layer composed of multiple hierarchically organized controllers ensures the availability of the MEC's computing resources. It instantiates, implements, monitors, and terminates network service nodes (virtual machines) on demand by users. In addition, it coordinates the cooperation between multiple network operators to share different network services.

- *Flexibility:* The work in [34] designs a MEC-based RAN managed by a hierarchical structure of SDN controllers to make efficient resource allocation decisions in an industrial IoT environment. It proposes two resource control schemes, namely a centralized and a distributed scheme. In the former, for each task offloaded from an industrial IoT device, a high-level controller is responsible for designating a MEC server and allocating the appropriate computing and transmission resources to execute and transmit this task. Then, it delivers commands to the local controllers, who ensure the fulfillment of these commands by managing their local resources. In the distributed scheme, the high-level controller only designates where a task should be executed, and each local controller allocates the computing and transmission resources to execute and transmit each task. Therefore, the hierarchical SDN control plane provides more flexibility in managing the MEC environment's resources. Authors in [35] propose an offloading and resource allocation mechanism in an SDN-based MEC vehicular network. The proposed mechanism runs on an SDN controller. The latter uses this mechanism to decide, based on an execution delay threshold, whether a task should be offloaded to a MEC server or executed locally in the vehicle. If a task is offloaded, the SDN controller jointly optimizes the transmission power of the vehicle, subchannel assignment, and computing resource allocation. By

leveraging the features of SDN technology, *i.e.,* centralized logic control and programmability, MEC resources can be flexibly allocated and jointly optimized with other network resources such as radio sub-channels and transmission power. In [36], an SDN-enabled model is introduced to place the virtual network functions in a MEC environment and allocate the computing and storage resources to them. The SDN controller decides how many virtual network functions are needed to meet the end users' requirements, where each virtual network function should be placed, and how many computing and storage resources each VNF should have. These decisions are subject to the availability of resources in the MEC servers. The SDN controller facilitates the placement of virtual network functions, which reduces the cost of allocating MEC resources.

- *Resilience:* Authors in [37] leverages SDN to design a resilient IoT data exchange approach in an edge computing infrastructure. To maintain services provided to IoT devices in the event of connectivity issues, this approach uses SDN to configure edge computing resources as a backup to cloud resources. The SDN controller uses its global view of the network to collect information about the network state to be aware of any link or node failures. When a failure occurs, the SDN controller redirects the data flow to an available edge node or to the cloud using the appropriate paths. SDN enables dynamic adaptation of IoT data exchanges without changing already defined requirements by collecting and maintaining accurate network conditions. In [38], a distributed software-defined edge computing system is implemented to manage the mobile access of IoT devices. Each SDN controller uses an efficient algorithm to match the available edge access point to IoT devices based on its local view of the network. Also, SDN controllers exchange information to ensure seamless switching of IoT devices between access points located in different geographical areas. A distributed SDN control plane supports resilient access control to IoT devices over heterogeneous edge access points. In [39], SDN is used to support fault tolerance in the task processing of delay-sensitive applications within an edge computing Internet of vehicles network. When a road-side unit receives a request from a vehicle, it notifies the SDN controller that defines an optimal data forwarding path and allocate computing resources. In the process of allocating computing resources and data transmission links, the SDN controller considers the failure rates of links and edge nodes, which improves the resilience of the edge environment.

2.3 SDN control plane load balancing

SDN technology is currently indispensable for managing edge computing resources, and its role is becoming more prominent with the deployment of the 5G network. In particular, the distributed architecture of the SDN control plane is widely adopted in large-scale networks since it provides more scalability in managing network resources, which efficiently increases network performance. In a distributed SDN control plane architecture, each SDN controller has a partial view of the network and manages only a part of the network elements and resources. To maintain consistency in the network management by several SDN controllers, they communicate with each other and exchange necessary information. Although such a distributed architecture meets the requirements of large-scale real-world network deployments in terms of scalability and reliability, it raises a serious challenge, namely load balancing between SDN controllers.

2.3.1 Load balancing problem

The load balancing problem in the SDN control plane can be defined as an uneven distribution of the control load between controllers. This problem is due to permanent and unpredictable traffic fluctuations at the data plane level, notably in large-scale networks when the number of applications is huge. To support this traffic behavior, each SDN controller must react quickly by communicating with multiple network elements and making appropriate decisions to better manage the resources under its control. Thus, an SDN controller is exposed to a tremendous number of requests received, either from the data plane components that it is managing or from other network elements, to which it should respond as fast as possible. In a distributed SDN control plane architecture, such a scenario makes the traffic sizes that should be handled by different controllers become disparate, where some controllers will be too loaded or even overloaded, while others will be underloaded. When an SDN controller becomes overloaded, its performance in processing received requests and managing its local network decreases.

2.3.2 Load balancing approaches

To balance the load between SDN controllers, various approaches have been proposed, including data plane component migration, traffic redirection [40], and workload sharing [41]. The data plane component migration approach is widely used to fairly balance the load of the control plane since it dynamically adjusts the load of the controllers. It involves changing the control domains to which the data plane components belong. In other words, when the load of the control plane becomes unbalanced, some data plane components under the control of overloaded SDN controllers migrate to be under the control of underloaded SDN controllers. Accordingly, the overloaded SDN controllers can

be alleviated and, at the same time, exploit the processing capacity of the underloaded SDN controllers. To migrate a data plane component from an overloaded controller to an underloaded controller, Dixit *et al.* defined a migration protocol in [42].

In recent years, a large variety of algorithms and schemes based on data plane component migration have been proposed in the literature. To improve the load balancing performance in the SDN control plane using the data plane component migration technique, different parameters have been considered. Therefore, we classify these works based on three main parameters: migration efficiency, response time, and resource utilization.

- *Migration efficiency:* is defined as the trade-off between the load balancing rate and the data plane component migration cost. The load balancing rate indicates the degree of load balancing in the control plane. To complete a data plane component migration operation, the protocol [42] consists in exchanging several messages between the concerned entities, *i.e.,* overloaded and underloaded controllers, and data plane components. Thus, the migration cost can refer to the time required to exchange all these messages, the network overhead generated by exchanging these messages, or the load change of the control plane. The authors of [43] define the load balancing rate as the difference between the control plane load variances before and after performing migration operations. To improve the migration efficiency, they propose a three-stage load balancing scheme. First, it divides controllers into overloaded and underloaded controllers using a load diversity factor. This factor measures the load difference between controllers and when it exceeds a defined threshold, migration operations of the data plane components must be performed. Second, it selects which data plane components should be migrated from the overloaded controllers according to a distribution probability. Indeed, the data plane components with a small load and located far from overloaded controllers, have a high probability of being migrated. Third, it chooses a destination controller for each migrated data plane component that maximizes the migration efficiency. Similarly, the works [44] and [45] propose the same approach. However, the authors of [44] define the load balancing rate using the response time of the controllers instead of their load. On the other hand, the authors in [45] define the load of a controller by considering the requests received from the data plane components, as well as the synchronization and routing overheads. The authors of [46] improve the migration efficiency by defining two types of data plane migration operations, namely shift and swap moves. A shift move is a classical operation of migrating a data plane component from an overloaded controller to an underloaded controller. A swap move is when

two data plane components simultaneously exchange their control domains through migration operations. A swap move is performed when the load is unbalanced in the control plane and shift moves fail for certain reasons such as overloading the target controller.

- *Resource utilization:* Refers to the network resources, such as bandwidth status, computing load, or memory usage. In [47], the proposed load balancing framework selects the data plane components to be migrated and where the migration should occur based on the CPU, memory, and bandwidth status of the controllers. Specifically, this framework migrates, from the control domains of overloaded controllers, the data plane components whose load consumes less controller resources. In [48], it is stated that the more the data plane traffic crosses several control domains, the more the communication between the control plane and the data plane increases. Consequently, the resource consumption of the control plane increases. To achieve an efficient allocation of control resources, the association between the data plane components and the controllers is adjusted based on the flow path characteristics. For this reason, the data plane components are assigned to controllers with the aim of reducing the number of control domains through which the data plane traffic should pass. In [49], a control plane resource utilization model is designed to measure the resources requested by the data plane components in terms of CPU, memory, and bandwidth. This model is used to detect overloaded controllers. Then, the data plane components to be migrated are chosen based on the improvement in resource utilization balancing. In addition, the delay required to complete migration operation is considered to avoid costly and time-consuming migration operations.

- *Response time:* Represents the time required for an SDN controller to respond to a request received from a data plane component. The authors of [50] design a load balancing mechanism that identifies overloaded and underloaded controllers based on their response time. The latter is used to measure the load of controllers and to define an overload threshold at which migration operations should be triggered. The authors of [51] propose a two-phase algorithm that minimizes the average response time of the control plane. In the first phase, a stable matching between controllers and data plane components is performed while ensuring a worst-case controller response time for each data plane component. In the second phase, the previous matching is adjusted to achieve a near-optimal load balancing among controllers. The authors of [52] target the same objective, *i.e.,* minimizing the response time of the controllers, by balancing the control plane load. To ensure this objective, they define a minimum threshold of processing resource utilization that each controller

should achieve. When performing migration operations, this threshold prevents migrating a large number of data plane components to controllers with high-processing capacity.

2.3.3 Conclusion

Nevertheless, despite the potential advantages of the previously discussed work, we identify a significant research gap in their SDN control plane load balancing approaches. All approaches act in a reactive manner when performing migration operations of the data plane components. Indeed, a reactive mechanism triggers migration operations of the data plane components after detecting that the load is unbalanced in the control plane. Accordingly, the overloaded controller remains congested until the migration operations are completed, which increases its response time to requests received from the data plane components. Therefore, when the performance of an SDN controller decreases, it affects the overall operation of the network.

2.4 SDN for network slicing

2.4.1 Network slicing

To accurately design network services that meet the end-user requirements and deliver them in a more cost-effective manner, network operators need better control over the characteristics and quality parameters of these services. To achieve this goal, network slicing technology enables network operators a significant opportunity to create and deliver diversified network services compatible with the varying requirements of end-users while reducing operators' capital (CAPEX) and operating (OPEX) expenditures [5]. With network slicing, operators can segment the network to deploy multiple logical networks on top of a common and shared physical infrastructure. Each logical network, called a network slice, is tailored and dedicated to serving a specific service or application by providing the appropriate resources that guarantee the required quality of service. Operators can flexibly deploy, optimize, and retire network slices according to their needs, allowing a much more granular level of control over how they allocate their network resources. A network slice can span across multiple network domains, namely the RAN, transport network, and core network to provide end-to-end service. RAN slicing consists in customizing and managing the virtualized base stations functions and sharing radio resources between the created network slices. Core network slicing involves partitioning the main resources of the core network, including the computing and storage resources, as well as assigning VNFs operating in the core network to the crated network slices. Transport network slicing is

an abstract network topology that seeks to connect different endpoints with appropriate isolation and a specific service-level agreement (SLA).

Network slicing standardization efforts are conducted in several working groups and projects such as 3GPP and global system for mobile communications (GSMA). In 3GPP, several working groups are actively involved in defining the network slicing specifications. For instance, the 3GPP SA1 group identifies the potential service requirements to enhance network slicing support in different use cases and scenarios [53]. The 3GPP SA2 group defines the fundamental system architecture to support network slicing technology [54]. The 3GPP RAN group describes the use cases and solutions to improve the RAN slicing [55]. The GSMA network slicing taskforce project harmonizes the network slicing standardization ecosystem to ensure that operators, vendors, and service providers can consider common network slicing solutions [56].

2.4.2 Network slicing in next-generation networks

5G and beyond-5G/6G networks are enabling a new world of applications and services for a wide range of industries. Indeed, 5G supports three main services that are classified into enhanced mobile broadband (eMBB), massive machine-type communications (mMTC), and ultra-reliable low-latency communications (URLLC) services [3]. The eMBB services accommodate the applications that require stable connection with a very high data rate, such as Virtual Reality (VR) and high-definition streaming. The mMTC services support the connection of a massive number of devices, e.g., IoT devices, which generate small data and have sporadic traffic. The URLLC services target small payload applications that require low latency transmissions with very high reliability, such as manufacturing automation, autonomous driving, and remote surgery. Therefore, each application has a distinct set of requirements to accommodate a specific use case, which presents a serious challenge for network operators. Network slicing technology overcomes this problem since it allows network operators to create very specific network slices for very specific applications over a single physical network [57]. In addition, network slicing not only meets the requirements of new applications, but also unlocks new revenue for network operators by enabling SLA and performance-based pricing.

RAN slicing is an essential part of end-to-end network slicing as it allows for a much more granular degree to differentiate the services that will be available in the network [58]. As a result, innovation can be stimulated in various sectors, which presents a major opportunity for service providers to better monetize the services offered to their customers. The RAN slicing also affects the number of users who can access to the provided services, which in turn has a direct impact on the revenues of the service providers. Note that with

edge computing capabilities, the RAN slicing involves the management and control of two main resources, namely communication resources and computation resources. RAN communication resource slicing involves sharing the radio resources such as bandwidth and transmission time interval among the deployed RAN slices. On the other hand, RAN computation resource slicing consists in allocating the available computing resources in the MEC servers, which are connected to the base stations, between the deployed RAN slices. Therefore, when RAN slicing is performed in an appropriate manner, it maximizes network resource utilization and revenue while meeting customer requirements.

2.4.3 SDN-based RAN slicing

SDN provides the ingredients required for RAN slicing [59]. The SDN controller can seamlessly manage and orchestrate the RAN slices using its global view of the network. Indeed, it can dynamically manage the lifecycle of a RAN slice, which includes the preparation phase, the commissioning phase, the operation phase, and the decommissioning phase [60]. The preparation phase consists in designing and preparing the environment and the resources required to deploy the slice. In the commissioning phase, the slice is instantiated by assigning to it the reserved resources. The operation phase involves supervising the slice operation and performing resource updates if needed. When the slice is no more needed, the decommissioning phase is activated to remove the crated slice.

Network operators can leverage the programmability provided by SDN controllers to automate the allocation of RAN resources, such as communication and computation resources, to RAN slices. Although SDN enables RAN slicing with great agility and efficiency to deliver the required services, defining the appropriate RAN slicing policies that will be used by the SDN controllers is a challenging task. This challenge is due to the limited resources of the RAN that should meet the diversified requirements of a large number of next-generation network applications.

Several SDN-based approaches have been proposed in the literature to address the resource-constrained challenges of RAN slicing, including how to ensure that heterogeneous RAN slices are satisfied in terms of quality of service, and how to adapt the resource allocation mechanism to the variable and highly dynamic nature of the RAN environment. We classify these approaches according to the RAN resources considered in the RAN slicing. Note that we focus on the two main RAN resources, i.e., the communication resources and the computation resources. Therefore, to perform the RAN slicing, some approaches consider only the RAN communication resource, while others jointly consider the RAN communication resource and the RAN computation resource.

- Communication resources slicing: The authors of [61] propose a resource scheduling strategy for RAN slicing, where the spectrum resources are shared among a number of RAN slices to meet their required throughput. The proposed strategy performs the resource allocation in two time-scales, namely large time-scale and small time-scale. For large time-scale resource allocation, a deep learning algorithm is used to predict the periodic traffic of RAN slices. The obtained traffic prediction results are used to perform the large time-scale resource allocation for each RAN slice. To deal with inaccurate predictions and unexpected network states, a reinforcement learning algorithm is used to perform online resource allocation of RAN slices on a small time-scale. The authors of [62] design two schemes to allocate radio resources to users considering the requirements of delay-sensitive and delay-elastic applications. The first scheme assigns radio resources to RAN slices based on their preferences and importance in different base stations. The second scheme adjusts the previously allocated resource to balance QoS satisfaction and resource utilization between the RAN slices. The authors of [63] propose a heuristic algorithm to partition the radio spectrum resources into different bandwidth slices and allocate them to heterogeneous base stations. The proposed algorithm considers the QoS requirements of two types of end-devices, including end-devices requiring high transmission reliability and end-devices demanding high throughput. It seeks to maximize the achievable downlink rates of the end-devices.The author of [64] design a hierarchical RAN slicing framework to support eMBB and URLLC services with various QoS requirements. In the upper level, an SDN controller manages a pool of radio resources that are shared among the base stations. According to instantaneous traffic demands, the SDN controller allocates a number of radio resource blocks to each base station. In the lower level, each base station allocates to each of its associated users a number of radio resource blocks based on the pre-allocated resources by the controller. The authors of [65] develop a dynamic radio resource allocation scheme based on a genetic algorithm. To meet the data rate and latency requirements, the proposed algorithm allocates radio resources to eMBB, URLLC, and mMTC users considering their location and distribution.

- Communication and computation resources slicing: The authors of [66] present a RAN slicing framework for the Internet of vehicle services to dynamically allocate radio spectrum and computation resources among RAN slices. They propose a reinforcement learning algorithm that first makes the resource allocation decision, which includes the allocation of radio spectrum and computation resources for the slices in the base stations. Then, it distributes the workload offloaded by vehicles

between the slices to balance the load between the base stations. The authors of [67] study the multi-tenant cross-slice resource allocation problem, where multiple service providers compete with each other to provide their subscribers access to the virtual computation and communication slices. When the auction bids from service providers are received by the infrastructure provider, an SDN controller assigns channels to the users according to the computation and communication resources required by the subscribers. A DRL algorithm is used to learn the optimal computation offloading and packet scheduling policies. The authors of [68] propose a RAN slicing and computation task scheduling framework that intends to jointly maximize the communication and computation resource utilization while guaranteeing the QoS for autonomous driving tasks. They leverage a multi-agent DRL algorithm to slice the pooled radio resources between the base stations and allocate the computing resources to the offloaded tasks from vehicles based on network traffic load conditions. The authors of [69] propose a novel dynamic wireless and computation resource allocation scheme based on a deep deterministic policy gradient algorithm. The proposed scheme considers the task offloading cost and provides both resources to eMBB, URLLC, and mMTC slices with the purpose of maximizing the revenue of the network operator. The authors of [70] design an online method that optimizes the RAN resource consumption, including computation and bandwidth resources while ensuring an isolation level between the slices. For achieving this, a DRL algorithm is exploited to explore the near-optimal RAN slicing decisions.

2.4.4 Conclusion

These approaches perform the RAN slicing either centrally or hierarchically. In centralized resource allocation solutions, the SDN controller makes all decisions. On the other hand, in hierarchical solutions, the allocation of RAN resources is performed in several levels, such as in the SDN controller level and the base stations level. However, the centralized RAN slicing approaches suffer from bottleneck and single point of failure issues. In hierarchical RAN slicing solutions, the resource allocation operation in the lower levels can fail when the resources pre-allocated by the upper levels are not sufficient since it should wait for the next resource reservation update.

Chapitre 3: Avant-propos

Auteurs et affiliation:

Abderrahime Filali: étudiant au doctorat, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Soumaya Cherkaoui: Professeure, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Abdellatif Kobbane: Professeur, Université Mohammed-V, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS), Rabat-Maroc.

Date d'acceptation: avril 2019.

État de l'acceptation: version finale publiée.

Revue: IEEE International Conference on Communications (ICC).

Titre français: Planification de la migration des commutateurs basée sur la prédiction pour l'équilibrage de charge SDN.

Résumé français:

Les architectures distribuées du plan de contrôle du réseau défini par logiciel (en anglais software-defined networking, SDN) nécessitent une conception judicieuse pour équilibrer la charge entre les contrôleurs. Les solutions proposées pour l'équilibrage de charge SDN utilisent généralement des opérations de migration de commutateurs. Cependant, une migration efficace des commutateurs nécessite de déclencher l'opération au bon moment, et de choisir judicieusement le commutateur migré et le contrôleur de destination. Dans cet article, nous proposons un algorithme de planification de la

migration des commutateurs pour améliorer l'efficacité de la migration et assurer l'équilibrage de charge entre les contrôleurs. Notre algorithme utilise un modèle de prévision auto-régressif intégré à moyenne mobile (en anglais auto regressive integrated moving average, ARIMA) ARIMA à plusieurs étapes pour prédire la charge du contrôleur à long terme. Lorsqu'une surcharge est prédite, une opération de migration des commutateurs est programmée à l'avance. Après avoir validé la précision du modèle de prévision ARIMA, nous avons évalué la performance de l'algorithme en analysant le temps de réponse des contrôleurs. Les résultats numériques confirment les performances de l'algorithme proposé.

Chapitre 3: Foreword

Authors and affiliation:

Abderrahime Filali: Ph.D. Student, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Soumaya Cherkaoui: Professor, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Abdellatif Kobbane: Professor, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS) Mohammed V University in Rabat, Morocco

Date of acceptance: april 2019.

Acceptance status: final version published.

Conference: IEEE International Conference on Communications (ICC).

Title: Prediction-based Switch Migration Scheduling for SDN Load Balancing

CHAPTER 3

Prediction-Based Switch Migration Scheduling for SDN Load Balancing

3.1 Abstract

Distributed architectures of the SDN control plane require a careful design for balancing the load among controllers. Solutions proposed for SDN load balancing usually use switch migration operations. However, an efficient switch migration means triggering the operation at the right moment, and judiciously choosing the migrated switch and the destination controller. Here, we propose a switch migration scheduling algorithm to improve the migration efficiency, and ensure load balancing between controllers. Our algorithm uses a multi-step ARIMA forecasting model to predict the long-term controllers load. When an overload is predicted, a switch migration operation is scheduled in advance. After validating the accuracy of the ARIMA forecasting model, we evaluated the performance of the algorithm by analyzing the response time of controllers. Numerical results confirm the performance of the proposed algorithm.

3.2 Introduction

In large-scale networks, a distributed architecture of SDN (Software Defined Networking) control plane is highly recommended because of its flexibility and ability to handle huge amounts of traffic [7]. A distributed SDN architecture is, indeed, an effective way to provide more scalability to the network, regardless of the target applications in the data plane (*i.e.,* IoT, connected vehicles or future 5G NR-enabled applications [71, 72, 73]). However, several load distribution issues can make the exploitation of the distributed controllers'

processing capacity inefficient. Traffic fluctuations in large-scale networks (*e.g.,* commercial data centers), can translate directly in hugely disparate traffic sizes to be handled by different controllers in the control plane. In fact, each controller can receive, from the data plane equipment in its control domain, thousands of packets to process as quickly as possible. When there is an overload, the controller's response time will increase, thus affecting the traffic flows in the data plane. Some load balancing between SDN controllers is, therefore, required to minimize latency and optimize processing resources utilization at different controllers.

Switch migration is a solution that is usually used to fairly distribute load among controllers. Switch migration consists in a migration operation which is initiated when the controller load reaches a defined threshold, where some switches are migrated to less-loaded controllers. Changing the master controller of a specific switch happens by exchanging role request messages between the two involved controllers (*i.e.,* the source controller and the destination controller) [42]. Switch migration has been intensively studied and a large variety of schemes and algorithms have been proposed over the previous years [74, 43, 75, 76, 77, 78, 79, 80, 52]. However, most of these works propose either reactive mechanisms for switch migration or apply simple load prediction models to decide whether a migration is necessary. Reactive mechanisms start a switch migration after a controller overload has already been detected. Therefore, the overloaded controller remains congested for a certain period of time, during which the response time of the controller increases. Also, using simple prediction models may yield less accurate load estimations, leading to delayed switch migrations or to unnecessary migration. Such approaches may overlook a significant number of messages sent by some switches, which can bias the load estimation.

In this paper, we define a long-term switch load prediction model, based on the autoregressive integrated moving average (ARIMA) time series approach [81]. ARIMA is a flexible class of forecasting models that utilize historical information (in this case controller load information) to make predictions. Our model uses previous observations of the switch's load to predict its future loads. Forecasting switch load allows to find out if a controller will be overloaded and, thus, schedule a switch migration in advance. Accordingly, we propose a switch migration scheduling algorithm to decide which switch should be migrated, and where the migration should happen. Knowing in advance that a controller could be overloaded can improve switch migration efficiency and help avoid the processing congestion in the control plane.

Our contributions in this work can be summarized as follows: (1) we define an optimization problem minimizing the load difference between controllers; (2) we use the ARIMA model to forecast switch load and prevent controllers' overload; and (3) based on ARIMA, we propose a switch migration scheduling algorithm to transfer some switches from overloaded controllers to underutilized ones. We performed extensive simulations to evaluate the forecasting model and the switch migration scheduling algorithm. The results show that the prediction model is accurate and parsimonious (*i.e.,* our model uses fewer parameters to adjust the data). Also, the small difference between the response of the controllers confirms the performance of the proposed algorithm.

The remainder of this paper is structured as follows: Section II discusses some proposed algorithms and schemes for switch migration. In Section III, we define the network architecture model, and formulate the optimization problem. Section IV introduces the multi-step prediction model using ARIMA and describes the proposed algorithm. Section V discusses the obtained results and highlights the performances of the proposed algorithm. Finally, Section VI concludes the paper.

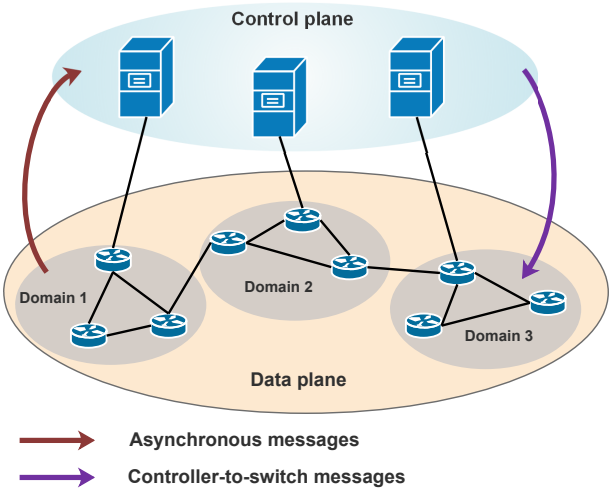


Figure 3.1 A distributed architecture of controllers with their respective control domains

3.3 Related Work

Switch migration is considered as an interesting solution for load balancing in the control plane. Several algorithms and schemes have been proposed in the literature to improve switch migration operations. G. Cheng et al. [74] formulated the switch migration problem as an optimization problem maximizing processing resources' utilization. For solving the problem, they designed a distributed hopping algorithm to reach the desired service level agreement. However, the load estimation algorithm predicts the controller load by calculating the average arrival rate of the requests from the switches, which is less accurate than a forecasting model like ARIMA. Furthermore, migrated switches and destination controllers are randomly chosen by their algorithm. In [43], a switch migration plan was proposed by making a trade-off between load balancing rates and migration costs. In the switch migration operation, the phase of election of a destination controller consists of choosing the controller according to migration efficiency, which is defined by the ratio of load balance variation to migration cost. Therefore, the same controller may be chosen as a target controller for several migrated switches, thus potentially increasing its load. Moreover, to reach the destination controller, a

transit controller can be added in the proposed switch migration plan, which complicates the migration operation. The authors in [75] addressed the controller load balancing problem as a zero-sum game problem, where underutilized controllers are the players and migrated switches are considered to be commodities. To maximize their payoffs, the players (cold spot controllers) compete to be elected as a master controller. In this game, however, the communication overhead between controllers is high since several messages will be exchanged in the network (*e.g.,* invitation requests to the neighbor controllers to participate in the game, information about migrated switches).

The authors in [76] proposed an algorithm for switch cluster migration from congested SDN controllers. The algorithm forms a cluster of switches characterized by a strong communication between them and assigns the formed cluster to one controller. However, the algorithm increases the control traffic overhead considerably because it migrates a cluster of switches. To decrease the load of an overloaded controller as fast as possible, [77] defined a dynamic scheme to choose the migrated switch with largest flow arrival rate from edge switches. However, choosing the migrated switch only from the boundary switches can decrease the migration efficiency if no boundary switch migration offloads the controller. In [78], a two-phase algorithm is proposed to dynamically assign switches to controllers. The first phase is based on the matching game concept and in the second phase a coalition game is used to ensure load balancing between controllers. In the second phase, switches can change their coalitions (*i.e.,* migration operation) according to their utilities to achieve Nash equilibrium. The presented approach produces a high switch migration frequency during the first and second phases. Thus, the migration cost will be important. Using the standard deviation of the controllers' load, the authors in [79] developed an f-approximation algorithm. The first step of the algorithm initializes the partitions of the whole network. In the second step, each controller evaluates its state with regard to a local threshold to see

whether a transfer of heavily loaded switch to nearby controllers is necessary. The migration decision-making is based on the calculated threshold considering only nearby controllers, which can impact when the effectiveness of the migration in balancing the load.

3.4 Problem formulation

A distributed Software Defined Networking architecture adopts a finite set of controllers, $C = \{c_1, c_2, ..., c_n\}, |C| = n$. Each controller has a processing capacity in terms of the number of packets that it can handle per second, $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$. The forwarding plane is composed by a finite set of switches denoted by $S = \{s_1, s_2, ..., s_m\}, |S| = m$. In such architecture each controller manages several switches in its domain. In a master/slave SDN model, we define a Boolean variable $x_i^k \in \{0, 1\}$, where $x_i^k = 1$ if c_k is the master controller of the switch s_i. Otherwise $x_i^k = 0$.

In a switch-controller communication based on Open Flow protocol, there are three types of exchanged messages: Controller-to-switch (*e.g., switch_ configuration, modify_ state*), Asynchronous (*e.g., Packet_ in, Port_ status*) and Symmetric (*e.g., Hello and Echo*). To measure the load of a controller, *Packet_ in* messages processing is generally considered the most significant load. A *Packet_ in* message is generated from the switch to its controller when a new flow of packets arrives to the switch and there is no matching rule in the flow table. Let $l_i(t)$ be the generated load by the switch $s_i \in S$, *i.e.,* the number of *Packet_ in* messages sent to the controller at time t. Thus, the load of the k^{th} controller $L_k(t)$ at time t is defined by equation 3.1.

$$L_k(t) = \sum_{i=1}^{m} l_i(t) * x_i^k \tag{3.1}$$

Load balancing between controllers is ensured by minimizing the load difference between them. Accordingly, we have formulated the following optimiza-

tion problem:

$$\underset{x}{\text{minimize}} \quad \frac{1}{n}\sum_{k=1}^{n}|L_k(t) - L^*| \tag{3.2a}$$

$$\text{subject to} \quad L^* = \frac{1}{n}\sum_{k=1}^{n} L_k(t), \tag{3.2b}$$

$$\sum_{k=1}^{n} x_i^k = 1, \forall\, 1 \leq i \leq m, \tag{3.2c}$$

$$x_i^k \in \{0,1\}, \forall\, 1 \leq i \leq m, \forall\, 1 \leq k \leq n. \tag{3.2d}$$

Constraint (3.2b) calculates the average load of controllers. Constraint (3.2c) ensures that each switch maintains connectivity with a single master controller. Constraint (3.2d) guarantees that the variable x_k^i is binary.

Keeping the network performance in a good condition requires each controller to have almost the same load. However, a controller load can be unbalanced compared to the other if the traffic flow variation is important. In the case of an imbalanced load, some switches need to be migrated. Nevertheless, waiting for the controller load to reach its maximum capacity to migrate switches can lead to a degradation of system performance and a decrease in traffic flow processing capability. To avoid these issues a primitive switch migration, *i.e.,* before an overload happens, is needed. Of course, unnecessary switch migration can also degrade system performance, so a suitable load prediction model is necessary.

3.5 Workload prediction model

3.5.1 Time Series and Forecasting

Predicting the future load of an SDN controller L_t at time t is possible through the analysis of the historical data using a time series approach. A time series is defined as a sequence of observed data over a fixed-length time interval. Modeling a time series aims to understand the past observations for devel-

Algorithm 1 Scheduling Switcher Migration

Inputs: $\forall c_k \in C$, $\forall s_i \in S$, $\forall \alpha_k \in \alpha$

Outputs: MigrationSchedule

Initialize:
C_M : The set of controllers with migration actions.
C_A : The set of controllers with no migration actions.
T_{ol} : The controller overload threshold.
$migrationList = \emptyset$.
for all $c_k \in C_A$ **do**
  $W_{ol}^{c_k} = overload\_Prediction(c_k, W, T_{ol})$ % Algorithm2
  if $W_{ol}^{c_k} \neq -1$ **then**
    $flag = 0$
    for all $s_i \in S_{c_k}$ **do**
      if $L_k(t + W_{ol}) - l_i(t + W_{ol}) \leq T_{ol}$ && $flag = 0$ **then**
        add s_i to $migrationList$
        $flag = 1$
      end if
    end for
    if $flag == 0$ **then**
      choose the switch with the smallest load and add it to $migrateList$
    end if
  end if
end for
while $migrationList \neq \emptyset$ **do**
  choose the switch s_i with the smallest W_{ol}
  sort the C_A controllers in an ascending according to their load $L_k(t)$
  for all $c_k \in C_A$ **do**
    if $l_i(t) + L_k(t) < \alpha_k$ **then**
      add $(s_i \longrightarrow c_k)$ into MigrationSchedule
      move c_k to C_M
      delete s_i from $migrationList$
    end if
  end for
end while

oping a prediction model used to generate future values for the series (*i.e.,* make forecasts). The choice of an appropriate model is decisive for a successful time series forecasting. Among several forecasting methods, we preferred exploiting the autoregressive integrated moving average model. The ARIMA process was chosen for the implementation of our model because it is able to represent non-stationary time series, it is the most appropriate general model for treating problems with a single variable (switch's load in our case) and, extreme values have a less impact on the model. Depending on the use case, forecasts can be used for short or long term horizons. Often, time-series models are used to predict the next step, called one-step forecasting. In some cases, like ours, where decisions must be made based on long-term predictions, a multi-step forecast is required for planning strategies.

3.5.2 One-step prediction

Knowing the last p observations of $l(t)$, *i.e.,* $\{l(t-p+1), \ldots, l(t)\}$, the aim is to predict $l(t+1)$, which is the expected load that the switch s_i will transmit to its controller at time *t+1*. The time series $l(t)$ follows an ARMA (p,q) model if it is stationary and for each instant t:

$$l(t) = E(t) + \sum_{i=1}^{p} \phi_i B^i l(t) + \sum_{j=1}^{q} \theta_j B^j E(t) \tag{3.3}$$

Where ϕ_i $(i = 1, 2, \ldots, p)$ and θ_j $(j = 1, 2, \ldots, q)$ are the model parameters estimated from available data. E_t terms are random errors, *i.e.,* white noise process with zero-mean and constant variance σ^2. p and q are the number of lags used by the model, *i.e.,* the autoregressive order and moving average order, respectively. B is the backshift operator defined as follows: $B^i * l(t) = l(t-i)$. The ARMA model previously described can only be used for stationary time series. However, in our case, the time series shows a non-stationary behavior. Thus, to make our model stationary, we apply a finite differentiation using the backshift operator so, the differentiation of

degree d is defined by: $(1 - B^d) * l(t) = l(t) - l(t - d)$. Hence, an ARIMA (p,d,q) model is an ARMA (p,q) model that has been differentiated d times and can be given by:

$$(1 - \sum_{i=1}^{p} \phi_i B^i)(1 - B)^d l(t) = (1 + \sum_{j=1}^{q} \theta_j B^j) E(t) \tag{3.4}$$

3.5.3 Multi-step prediction

In our model, we aim to schedule migration operations over a time window $W \in \mathbb{N}^+$. For this purpose, the multi-step forecast is an appropriate approach to predict the future load few seconds ahead. There are several multi-step techniques, we opt for the recursive multi-step forecast strategy. This method involves using a one-step model multiple times where, the prediction for the previous time step is used as an input for making a prediction on the following step. Let $l(t + w)$ denotes the w^{th} step prediction of $l(t)$, the prediction of $\{l(t+1), l(t+2), \ldots, l(t+w)\}$ is obtained by iterating the one-step prediction w times. The w^{th} step prediction $l(t + w)$ is given by:

$$l(t + w) = ARIMA(l(t + w - p), ..., l(t + w - 1)) \tag{3.5}$$

3.5.4 Switch migration scheduling algorithm

We present a switch migration scheduling algorithm inspired from [82]. Based on the ARIMA time series model described above, the algorithm makes a multi-step prediction of the load sent by each switch to predict a possible controller overload. The algorithm then schedules a migration operation after choosing the migrated switch and the destination controller. Algorithm 1 takes as input the set of controllers, their processing capacities, and the set of switches handled by each controller. To avoid complexity and to reduce the communication overhead between controllers in the control plane, each overloaded controller will migrate only one switch. We divide the controllers into

two sets, a set of controllers performing a migration operation denoted by C_M, and the other set presents the controllers without any migration operation denoted by C_A. We set an overload threshold T_{ol} and, the ***migrationList*** is empty in the beginning. The first stage of Algorithm 1 is to predict in which step, from a time window W, a controller may be overloaded. Thus, for each controller in set C_A, Algorithm 2 is called (step 3 in Algorithm 1) to predict its load multi-steps ahead (w steps) depending on the desired time window $W \in \mathbb{N}^+$. The output of the Algorithm 2 is the predicted overload step of the controller $c_k \in C_A$, denoted by $W_{ol}^{c_k}$. Using ARIMA model (steps 5-9 in Algorithm 2) the load $l_i(t+w)$ of each managed switch s_i by the controller c_k will be forecasted for each step $w \in W$. Then (steps 10-15 in Algorithm 2), if the predicted load $L_k(t+w)$ of the controller c_k, which is the sum of the predicted loads of switches that it manages (equation 3.1), exceeds the fixed threshold, the Algorithm 2 returns in which step $W_{ol}^{c_k}$ the controller will be overloaded.

The next stage of Algorithm 1 defines the switch to migrate (steps 4-15 in Algorithm 1). For that, if the controller c_k will be overloaded in step $W_{ol}^{c_k}$, among all switches in S_{c_k} (*i.e.,* the set of switches managed by the controller c_k), the algorithm will choose the one which, after its migration, the controller load will be lower than the fixed threshold T_{ol}. If switch satisfies this constraint, it will be added to the ***migrationList***. Otherwise, the switch with the smallest load will be chosen to quickly offload the overloaded controller. The final stage of Algorithm 1 is to specify destination controllers for all migrated switches in ***migrationList*** (steps 17-27 in Algorithm 1). First, all switches in ***migrationList*** are sorted in an ascending order according to the overload step W_{ol} (*i.e.,* the switch that its master controller is the closest to be overloaded, will be migrated first). Then, sort the set C_A in an ascending order according to the load $L_k(t)$ and chose the destination controller for each switch, respecting not to violate the processing capacity α of the controller.

Finally, add each migration action to the ***migrationSchedule*** and move the included controllers to the set C_M.

Algorithm 2 Overload prediction

Inputs: $c_k \in C_A$.
W : The number of steps to predict.
T_{ol}: The controller overload threshold.
Outputs: $W_{ol}^{c_k}$: The predicted overload step of the controller c_k.

Initialize:
$W_{ol}^{c_k} = -1$
Let S_{c_k} denotes the set of switches handled by the controller c_k.
Let $l_i(t)$ denotes the generated load by the switch s_i at time t.
Let $l_i(t+w)$ denotes the predicted generated load at the w^{th} step.
for all $s_i \in S_{c_k}$ **do**
  for $w = 1, ..., W$ **do**
    $l_i(t+w) = ARIMA(l_i(t+w-p), ..., l_i(t+w-1))$
  end for
end for
for $w = 1, ..., W$ **do**
  $L_k(t+w) = \sum_{i \in S_{c_k}} l_i(t+w)$
  if $L_k(t+w) \geq T_{ol}$ **then**
    $W_{ol}^{c_k} = w$
    break
  end if
end for

3.6 Numerical results

3.6.1 Prediction performance evaluation

Forecasts accuracy can be verified by analyzing the forecasting model performance on new data that were not used during the model fitting phase. Thus, the generated dataset was divided in two portions, training and test data. The training data are used to estimate the parameters of the forecasting model and the test data are used to evaluate its accuracy. Using MATLAB, we generated 300 observations for 100 switches, *i.e.,* we have 100 time series and each one has a size equal to 300. An observation presents the load that a switch can generate (*Packet_in* requests). As mentioned above the observed data were divided into two parts, 250 observations were reserved for

training phase and 50 observations for testing the model. To determine the autoregressive and moving average order of the ARIMA model, the analysis of autocorrelation functions and partial autocorrelation functions allows to have an idea about each parameter. Moreover, we used Akaike Information Criterion (AIC) to choose the best model by varying p and q between [0-1]. AIC criterion (equation 3.6) has as input likelihood value L generated by the estimator and the number of parameters to be estimated K.

$$AIC = 2K - 2log(L) \tag{3.6}$$

In each time series, which presents the observed switch's load, we calculated the AIC for each forecasting step (50 steps). Figure 3.2 illustrates the average AIC value of the 100 time series in each forecasting step. A good model is the one with the smallest AIC values, according to the figure, the model with $p = 0$ and $q = 1$ is the best one. Moreover, evaluating a forecast model could be done through the residual analysis. A good forecasting model will yield residuals without autocorrelations. Thus, figure 3.3 depicts the residuals' correlogram of the $1st$ and the $50th$ forecast step and it can be seen that the residuals are uncorrelated. We choose to plot the autocorrelation function of the residuals in the 1^{st} and the 50^{th} prediction step, in order to check that there is no correlation between residuals in short and long-term predictions.

3.6.2 Proposed algorithm performance evaluation

Let us consider an SDN architecture composed of a set of controllers and switches. In a large-scale network, an important number of new flows can be received by a given switch within 10 μs [83]. Therefore, under such conditions, we assumed that a switch can generate a number of $Packet\_in$ messages ranging between 50000 and 100000. The simulation inputs of the algorithm are: $n = 6$ controllers, $m = 100$ switches and each controller has a processing capacity $\alpha = 1800000$. For ARIMA prediction model, we define 300 seconds

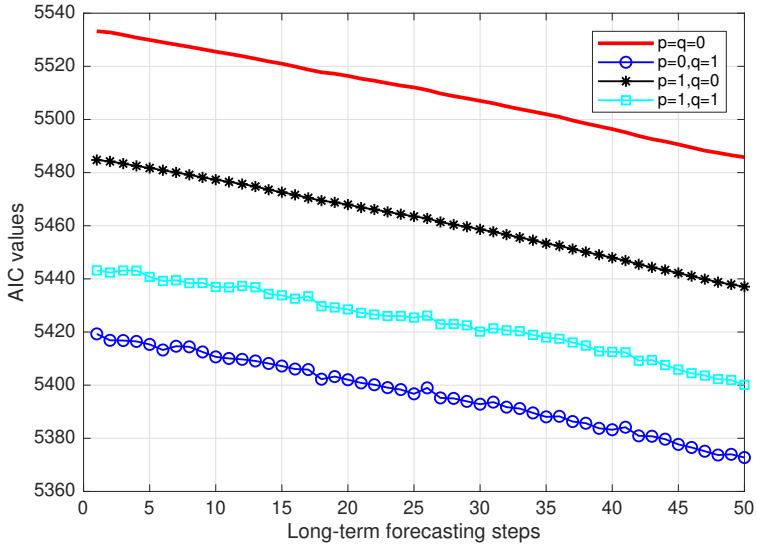


Figure 3.2 Akaike information criterion

of previous observations. The number of steps for the long-term forecasting is 15. The switch migration scheduling algorithm is used to forecast the controller load for the next 15 steps and schedule a migration operation in case of overload. We evaluate the algorithm performance by analyzing each controller's response time. Assuming that a controller can be considered as queue system and the arrival time of requests follows the Poisson process, the controller's response time T_{r_k} (equation 3.7) at time t is calculated by applying Little's law where, α_k and $L_k(t)$ are respectively the controller processing capacity and load at time t.

$$T_{r_k} = \frac{1}{\alpha_k - L_k(t)} \tag{3.7}$$

Figures 3.4 and 3.5 show the response time of the six controllers with and without migration. Using the algorithm, the load of each controller was forecasted 15 steps ahead. Figure 3.4 shows a big difference in the response time of the controllers due to the unbalanced load between them. Hence, scheduling a switch migration operation in case of overloading or significant load unbalance is required. Figure 3.5 illustrates the difference between the

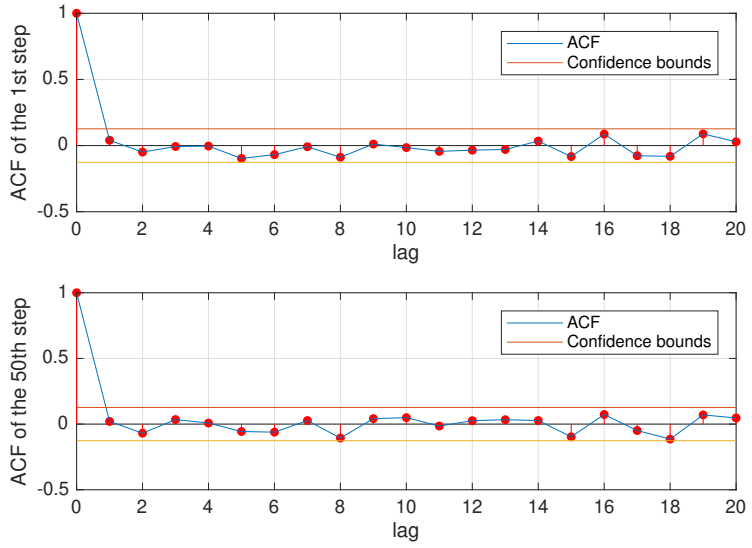


Figure 3.3 Residues autocorrelation function of the 1^{st} and 50^{th} forecast step

response times of controllers c_2, c_3, c_4 and c_5 is slight during the long-term prediction (15 steps). The controller c_1 is too overloaded so, the algorithm tries to offload it by migrating the switch with the smallest load.

3.7 Conclusion

The major concern of switch migration operations in SDN networks, is choosing the right time to migrate switches. Here, we proposed a prediction-based switch migration scheduling algorithm to balance the load between SDN controllers. Based on ARIMA multi-step forecasting model, the algorithm predicts when a controller can be overloaded. As a result, a switch migration is scheduled by choosing the migrated switch and the destination controller while respecting the controllers' processing capacity constraints. After validating the selected ARIMA model, using the AIC criterion and the residuals autocorrelation functions, the model performance by analyzing the controllers' response time. The simulation results show that switch migrations during the prediction interval allow avoiding controllers' congestion while reducing the difference between controllers' response time.

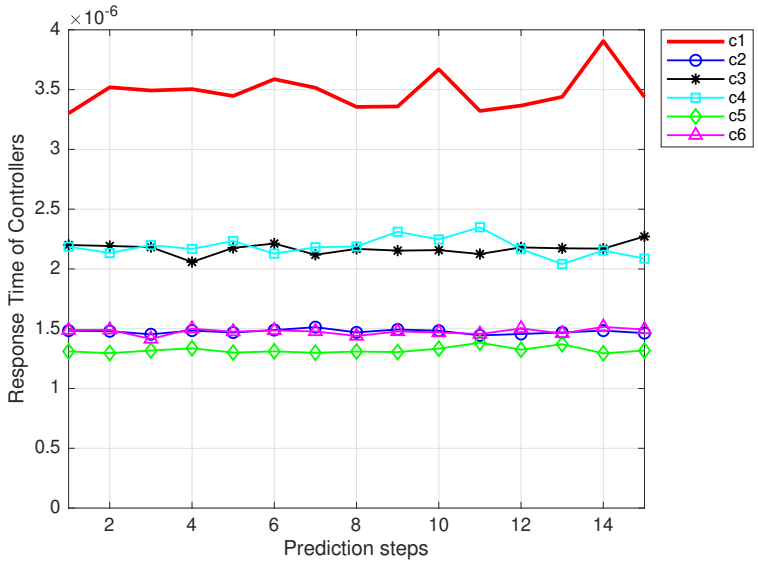


Figure 3.4 Response time of controllers without migration

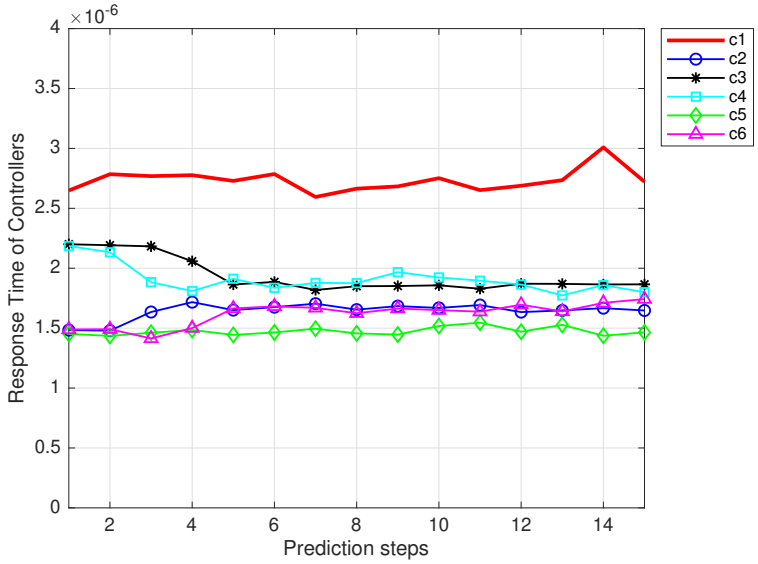


Figure 3.5 Response time of controllers with migration

Chapitre 4: Avant-propos

Auteurs et affiliation:

Abderrahime Filali: étudiant au doctorat, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Zoubeir Mlika: Chercheur post-doctoral, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Soumaya Cherkaoui: Professeure, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Abdellatif Kobbane: Professeur, Université Mohammed-V, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Rabat-Maroc.

Date d'acceptation: novembre 2020.

État de l'acceptation: version finale publiée.

Revue: IEEE Transactions on Vehicular Technology (TVT).

Titre français: Équilibrage de charge SDN préemptif avec apprentissage automatique pour les applications sensibles au délai.

Résumé français:

Le réseau défini par logiciel (en anglais software-defined networking, SDN) est un élément clé pour assurer l'évolutivité des réseaux 5G et des réseaux de périphérie à accès multiple. Pour équilibrer la charge entre les contrôleurs SDN distribués, la migration des composants du plan de données a été proposée. Contrairement à la plupart des travaux précédents qui utilisent des mécanismes réactifs, nous proposons d'équilibrer, de manière préemptive, la

charge dans le plan de contrôle SDN afin de supporter les flux réseau qui nécessitent des communications à faible latence. Tout d'abord, nous prédisons la charge des contrôleurs SDN pour éviter les déséquilibres de charge et planifier à l'avance la migration des composants du plan de données. Ensuite, nous optimisons les opérations de migration pour obtenir un meilleur équilibrage de charge en respectant les contraintes de délai. Plus précisément, dans la première étape, nous construisons deux modèles de prédiction basés sur les approches auto-régressive intégrée à moyenne mobile (en anglais auto regressive integrated moving average, ARIMA) et récurrente à mémoire court-terme et long terme (en anglais long short-term memory, LSTM) pour prédire la charge des contrôleurs SDN. Puis, nous réalisons une étude comparative entre ces deux modèles et calculons leurs précisions et leurs erreurs de prévision. Les résultats montrent que, dans les prédictions à long terme, la précision du modèle LSTM surpasse celle du modèle ARIMA par 55% en termes d'erreurs de prédiction. Dans la deuxième étape, pour sélectionner les composants du plan de données à migrer et où la migration devrait avoir lieu sous des contraintes de délai, nous formulons le problème comme un programme binaire non linéaire, prouvons sa NP-complétude et proposons un algorithme d'apprentissage par renforcement pour le résoudre. Les simulations révèlent que l'algorithme proposé est proche de l'optimal et surpasse les algorithmes de référence récemment publiés dans la littérature.

Chapitre 4: Foreword

Authors and affiliation:

Abderrahime Filali: Ph.D. Student, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Zoubeir Mlika: Postdoctoral Research Fellow, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Soumaya Cherkaoui: Professor, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Abdellatif Kobbane: Professor, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS) Mohammed V University in Rabat, Morocco

Date of acceptance: november 2020.

Acceptance status: final version published.

Journal: IEEE Transactions on Vehicular Technology (TVT).

Title: Preemptive SDN Load Balancing With Machine Learning for Delay Sensitive Applications

CHAPTER 4

Preemptive SDN Load Balancing With Machine Learning for Delay Sensitive Applications

4.1 Abstract

SDN is a key-enabler to achieve scalability in 5G and Multi-access Edge Computing networks. To balance the load between distributed SDN controllers, the migration of the data plane components has been proposed. Different from most previous works which use reactive mechanisms, we propose to preemptively balance the load in the SDN control plane to support network flows that require low latency communications. First, we forecast the load of SDN controllers to prevent load imbalances and schedule data plane migrations in advance. Second, we optimize the migration operations to achieve better load balancing under delay constraints. Specifically, in the first step, we construct two prediction models based on Auto Regressive Integrated Moving Average (ARIMA) and Long Short-Term Memory (LSTM) approaches to forecast SDN controllers' load. Then, we conduct a comparative study between these two models and calculate their accuracies and forecast errors. The results show that, in long-term predictions, the accuracy of LSTM model outperforms that of ARIMA by 55% in terms of prediction errors. In the second step, to select which data plane components to migrate and where the migration should happen under delay constraints, we formulate the problem as a non-linear binary program, prove its NP-completeness and propose a reinforcement learning algorithm to solve it. The simulations show that the

proposed algorithm performs close to optimal and outperforms recent benchmark algorithms from the literature.

4.2 Introduction

A distributed architecture of the SDN control plane is the appropriate solution to overcome the present issues of the centralized architecture, especially in large scale networks [71]. Specifically, a distributed architecture provides high scalability and higher processing capacity for mobile applications at the control plane, while avoiding traffic congestion, high delays and single point of failure (*i.e.,* bottleneck) issues [84]. In a distributed control plane architecture, see figure 4.1, the network is horizontally partitioned into several disjoint areas called control domains, where each control domain is managed by a single controller. This distributed architecture is currently coupled with several standards of existing and emerging technologies, notably Network Function Virtualization (NFV), Service Function Chaining (SFC) and Multi-Access Edge Computing (MEC) [85]. Moreover, it is a key component in the 5G network [86]. Being part of all these technologies, the distributed SDN architecture has, indeed, the ability to sustainably ensure the performance required by the target applications in the data plane, notably delay-sensitive applications [87, 72, 88]. However, such an architecture gives rise to load distribution problems which can seriously affect the scalability of the control plane and decrease the exploitation of its resources (*e.g.,* the controllers' processing capacity) [7]. In large-scale networks, where the number of applications is huge and traffic fluctuations are permanent, each controller should process and respond, as fast as possible, to thousands of requests received from the data plane components. Accordingly, this dynamic network traffic can lead to a load imbalance among controllers, *i.e.,* some controllers will be overloaded while others will be underloaded. When a controller is overloaded, its response time to data plane components requests increases. Therefore,

load balancing between SDN controllers is necessary to minimize latency and to efficiently exploit the control plane resources.

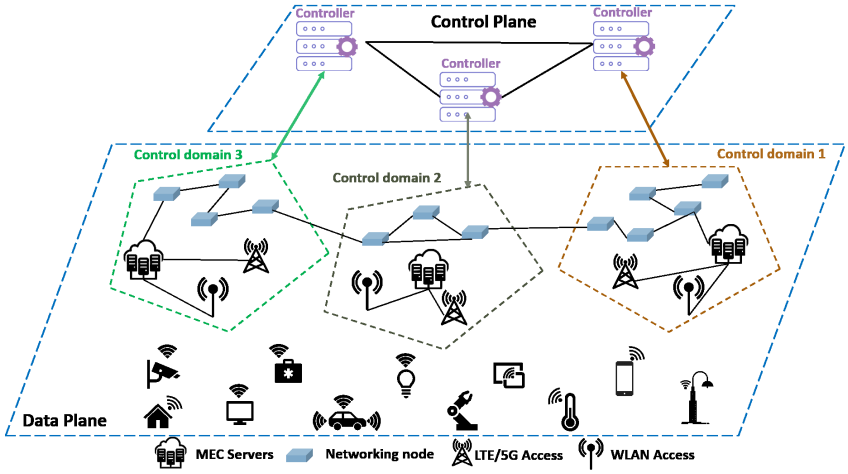


Figure 4.1 A distributed SDN Mobile Networks Architecture

The data plane component migration is an efficient solution that is widely used to balance the load between controllers [89]. Indeed, when the load on controllers becomes disparate, some data plane components are migrated from overloaded controllers to underloaded controllers, which can fairly balance the load between controllers. A migration operation of a data plane component consists in changing its control domain through a four-phase mechanism [42]. Although the data plane component migration is a key-enabler to achieve load balancing in the control plane, it presents several challenges such as where the migration should happen (*i.e.,* determining the overloaded and the underloaded controllers) and which data plane component should be migrated. To overcome these challenges, a large variety of algorithms and schemes have been proposed in recent years [50, 90, 91, 48, 51, 92, 46, 44, 45, 47, 52]. However, we identify two important research gaps in these works. First, all of them propose reactive mechanisms for data plane component migration. A reactive mechanism triggers a migration operation after detecting a controller overload, which makes it congested for a certain period of time. Consequently, the response time of the overloaded controller increases during this period.

Second, the cost of a migration operation has not been defined in terms of delay, which can decrease the migration efficiency.

In this paper, we aim to preemptively balance the load between controllers for delay sensitive applications on the edge. Therefore, in order to fill the aforementioned research gaps, we: (1) propose a long-term prediction model that forecasts the controllers' load; and (2) solve the load balancing of the distributed SDN architecture by optimizing the trade-off between load balancing and migration operation costs that are based on the response times of SDN controllers. Long-term predictions allow to detect a load unbalanced in the control plane and, thus, react proactively by scheduling migration operations in advance. This proactive mechanism not only prevents controllers from being overloaded, but also allows a careful selection of which data plane components should be migrated and where the migration should happen. Consequently, we formulate the SDN load balancing problem, called SDN Load Balancing for Delay Sensitive Applications (LBDSA), as an optimization program where the objective is to minimize, through migration operations, a load balancing factor combined with a migration operation cost. Due to this proactive mechanism as well as the design of LBDSA model, the delay sensitive application requirements are respected.

To summarize, the novelty of this work lies in two main parts. The first part is a comparative study between Autoregressive Integrated Moving Average (ARIMA) [93]—a traditional stochastic model—and a machine learning prediction model—Long Short-Term Memory (LSTM) [94]. In the second part, we define the LBDSA problem and propose a reinforcement learning algorithm to solve it. The key contributions of this work are summarized as follows:

- We build and evaluate two prediction models, one based on ARIMA and the other on LSTM.

- We provide a performance analysis comparing the accuracy of the short and long term SDN controller load predictions of the ARIMA and LSTM models.
- We model LBDSA as a non-linear binary program and study its NP-completeness.
- Due to the NP-completeness result, obtaining an optimal solution to LBDSA is computationally expensive. Thus, we propose a reinforcement learning algorithm, called 2WSLS, that is based on the well-known Win-Stay-Lose-Shift (WSLS) learning policy [95, 96].
- We evaluate the performance of 2WSLS against the optimal solution and two state-of-art SDN load balancing algorithms [46] and [50]. We show that 2WSLS is close to optimal and outperforms the benchmark algorithms.

The rest of the paper is organized as follows. Section II discusses recent works on load balancing in the SDN control plane based on data plane components migration operations. Section III introduces the ARIMA and LSTM prediction models. Sections IV constructs the ARIMA and LSTM models used for long-term predictions and evaluates the obtained models. Section V compares the performance of the constructed models in terms of accuracy for short-term and long-term predictions. Section VI and Section VII present, respectively, the system model for the SDN load balancing optimization and the mathematical programming formulation of LBDSA and its NP-completeness. Section VIII presents the proposed learning algorithm, 2WSLS, and explain its operations. Section IX highlights the performances of 2WSLS and discusses the obtained results. Finally, section X concludes the paper.

4.3 Related Work

The works presented in this section can be divided into those that: (i) do not consider any costs related to data plane component migration operations

[50, 90, 91, 48, 51, 92] and (ii) consider data plane component migration costs [46, 44, 45, 47].

In [50], the authors propose a load balancing mechanism based on the real response time of controllers. They also use this response time to define an appropriate threshold to decide whether a controller is overloaded or not. Overloaded controllers simultaneously migrate the heaviest switches to underloaded controllers. However, the proposed mechanism can increase the control traffic overhead since migration operations are performed simultaneously. The authors of [90] use a load diversity factor, which is the ratio of their loads, to find overloaded controllers and underloaded controllers. When the diversity is caused by an overloaded controller, some switches should be migrated from its domain of control, and when it is caused by an underloaded controller, all switch under its control should be migrated. To solve this switch migration problem, they use a non-cooperative game in which the players are the immigration controllers. In [91], a modified version of the Hungarian algorithm is used to assign switches to controllers. In the assignment process, the proposed algorithm considers the round-trip time between the switches and the controllers as well as the current load of the controllers. The work in [48] solves the switch-controller assignment problem in a two-phase manner. First, based on the flow paths, a greedy set coverage algorithm is used to form a minimum number of control domains that contain the most switches on the paths. The obtained assignment is used in the second phase by a coalition game strategy to improve the load balancing between the controllers. In this phase, switches can change their coalitions to achieve a Nash equilibrium. However, the proposed approach results in a higher number of migration operations. To solve the dynamic controller-switch assignment problem in a long-term horizon, authors in [51] decompose it into a series of one time-slot assignment problems using the Randomized Fixed Horizon Control (RFHC) framework. In each time slot, given the request rate of each

switch, a two-phase algorithm is used to assign switches to controllers. In the first phase, the assignment problem is modeled as many-to-one matching game. In the second phase, the obtained result from the first phase is used as an input for a coalition game to achieve a Nash equilibrium. To maximize the efficiency of the flow setup between controllers and switches, the authors in [92] define a long-term optimization problem for controller placement and controller-switch assignment. The placement of controllers is planned considering eventual future switch migration operations. After the controllers are placed, switch migration operations can be performed by choosing switches with the highest load from overloaded controllers. To sum up, the works in [50, 90, 91, 48, 51, 92] do not consider any migration cost to evaluate the effectiveness of migration operations.

The authors of [46] deal with the switch migration problem by performing two types of switch movements. They propose a heuristic algorithm that uses shift and swap moves to migrate switches from overloaded controllers to underloaded controllers. A shift move is the classical migration operation of a switch while a swap move is when two switches exchange their master controllers through migration operations. The cost of migrating a switch depends on the latency between (i) the controllers involved in the migration operation and (ii) the migrated switch and the controllers involved in the migration operation. The swap move is very useful for load balancing between controllers when the shift move is impossible. However, performing swap moves hugely increase the migration cost in terms of time and control plane overhead. The authors of [44] define a load diversity factor to divide controllers into overloaded controllers and underloaded controllers. The load diversity factor between two controllers is the ratio of their loads and when this ratio exceeds a predetermined threshold, switch migration is performed. The overloaded controller chooses to migrate the switch which has a large latency to it and consumes less resources. A destination controller is selected

to maximize the migration efficiency which is defined as the ratio of load balancing variation to migration cost. Similarly, the authors of [45] use the same load diversity factor but they consider the synchronization and routing overheads when calculating the load of a controller. In [47], a multi-criteria decision method called Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS) is used to choose the target controller. These criteria are based on resource consumption and hop distance. Also, they use a probabilistic model in which the switch with a low resource consumption is selected for migration. The cost of a migration operation is mainly defined, in [44, 45, 47], by the load added to the control traffic overhead without considering the time required for this migration operation, which can decrease the migration efficiency.

4.4 SDN controller load prediction models

In this section, we define the SDN controller load considered in this work and the basic concepts of time series. Then, we review the essential mathematical background on how the predictions are made using ARIMA and LSTM models.

4.4.1 SDN controller load

The load of an SDN controller at time t, denoted by $L(t)$, can be defined as the sum of the requests received from (i) the managed data plane components, (ii) other SDN controllers, or (iii) any network entity which can communicate with this controller. In this work, we consider the requests sent by the data plane components as the most important load of an SDN controller [97], in particular, the $Packet\_In$ messages of the OpenFlow protocol.

To predict the load of each SDN controller in a distributed control plane architecture, we assume that there exists a root controller that has a global view of the network [98]. Indeed, the root controller is connected to the other controllers and does not manage any data plane component. In such an

architecture, all controllers update the root controller of their control domain state. Thus, the root controller knows all information about each controller, including their load history. Based on the load history of the SDN controllers, the root controller can make load predictions for each controller.

4.4.2 Time series and predictions

Time series is an ordered sequence of measured data points (*i.e.,* observations) over a period of time, usually at regular time intervals. The importance of the order lies in a possible existence of dependencies between the observations, thus, changing the order could change the meaning of the data. In our case, we are working on a single variable, which is the SDN controller load $L(t)$. Therefore, the studied time series is termed univariate. Moreover, the controller load is measured every second, which makes it a discrete time series. The major benefit of analyzing the load of SDN controllers as a time series is to predict its future values. Make predictions is about forecasting the future with no error or as little error as possible. For this reason, a suitable prediction model must be trained by using the time series data. Also, the parameters of the model should be fitted to extract patterns from the data. Depending on the studied problem, predictions can be performed for short-term or long-term prospects. For the purpose of our study, which is the prediction of any load imbalance in the control plane to schedule data plane component migration operations in advance, the long-term forecast is the appropriate model.

4.4.3 Multi-step load prediction

Our objective through load prediction is to detect any load imbalance in the control plane. Hence, it is possible to schedule migration operations for data plane components in advance. To perform effective migration operations, several tasks need to be executed such as choosing the data plane components to migrate and their new control domains. For this reason, long-term prediction is the appropriate model that allows a careful selection of which data plane

components need to migrate and where the migration should happen. Long-term prediction refers to perform multi-step forecasting of SDN controller load. In this work, we used Multiple Input and Multiple Output (MIMO) strategy to predict multi-step of the SDN controller load. MIMO strategy uses the same past observations to predict the entire forecast sequence in one shot manner with the same fitted model. Let $L(t+w)$ be the predicted SDN controller load in step w (*i.e.,* at time $t+w$), the prediction of the sequence $\{L(t+1), L(t+2), ..., L(t+w)\}$ can be defined by $F(L(t), ..., L(t-p+1))$, where F is vector-valued function and p is the number of observations.

4.4.4 Load prediction using ARIMA and LSTM

In order to investigate the effectiveness of time series models to make predictions with higher accuracy and lower forecast errors, our study aims to compare a traditional stochastic and a machine learning models, namely ARIMA and LSTM, respectively. The main reasons that led us to choose ARIMA as a representative of stochastic forecasting models are the non-stationary property of our dataset and that ARIMA is considered to be the most general model among linear models. As a representative of the machine learning models, we preferred LSTM for many reasons, namely our data can be non-linear, it is dynamic and can comprise high autocorrelations across different periods of time. Also, LSTM can preserve and train the features of data for a longer period of time. Moreover, ARIMA and LSTM have been widely exploited in different forecasting domains [99, 100].

1) Load prediction using ARIMA

In an Autoregressive Integrated Moving Average (ARIMA) model, the future values of a variable are supposed to be a linear combination of several past values (*i.e.,* observations) and random errors. In time series analysis and prediction applications, the ARIMA model is the general model of the Autoregressive (AR) model, the Moving Average (MA)

model, and the combination of AR and MA (ARMA) models. Indeed, an autoregressive model of order p, denoted by AR(p), is based on the idea that the current value $L(t)$ of the time series at time t, *i.e.,* the SDN controller load in our case, can be expressed as a linear combination of p past values, with a random error $E(t)$. Rather than using past observations of the forecast variable, a moving average model of order q, denoted by MA(q), uses the previous errors as predictors for future outcomes. ARMA(p, q) model combines the two previous models as follows : $L(t) = E(t) + \sum_{i=1}^{p} \phi_i B^i L(t) + \sum_{j=1}^{q} \theta_j B^j E(t)$, where ϕ_i and θ_j are the parameters of the model and $E(t)$ is a random error with zero mean and constant variance. $B^i * \ L(t) = L(t-i)$ is the backshift operator.

The ARMA model can only be used for stationary time series. However, in many situations, the statistical properties (*e.g.,* mean, variance) of a time series change over time, making it non-stationary. For this reason, the ARMA model has been generalized by the ARIMA model to integrate the case of non-stationary time series. A non-stationary time series becomes unpredictable and cannot be modeled or predicted. Thus, working with stationary time series is easier because they can be analyzed and forecasted. In an ARIMA (p, d, q) model, a non-stationary time series can be transformed to a stationary series through the differentiation process. The ARIMA (p, d, q) model is formulated as $\left(1 - \sum_{i=1}^{p} \phi_i B^i\right)(1-B)^d L(t) = \left(1 + \sum_{j=1}^{q} \theta_j B^j\right) E(t)$, where d is the degree of differencing and $(1-B)^d L(t) = L(t) - L(t-d)$.

2) Load prediction using LSTM

As a type of Recurrent Neural Network (RNN), the LSTM network has a powerful ability to remember information from earlier stages in order to make predictions over an extended horizon. In fact, the main idea behind these networks is to give the model visibility about the previous

stages while generating predictions for the current input. Accordingly, previous states need to be remembered, which allows the network to learn and exploit the sequential observations when forecasting next steps. The LSTM network has the particularity to selectively remember pattern for long sequences. An LSTM network is a set of memory blocks called cells which are responsible for filtering, eliminating and adding information through three gates, namely the input gate, the forget gate and the output gate. The forget gate is used to decide whether the information from the previous cell should be thrown. The decision of which part should be memorized from the current input and hidden state is made by the input gate. The output gate acts as a filter to determine what will be conducted out of the cell.

4.5 Construction of prediction models

This section introduces the source of the dataset used in this work and how it is prepared before being used by prediction models. Next, we describe the process of constructing the ARIMA and LSTM models and evaluate the obtained models.

4.5.1 Simulation setup

All simulations and experiments, namely dataset generation, ARIMA and LSTM modeling, training process and predictions, were performed on a laptop with an Intel Core i7-8750H processor, 16 GB of RAM and NVIDIA GeForce GTX 1070 graphic card. The software environment used in this work includes Keras version 2.2.4 with TensorFlow 1.14.0 backend.

4.5.2 Dataset description and preparation

The dataset used in this work is the SDN controller load, in other words, the sum of the number of requests sent by the data plane components to the controller. In order to have this type of data, we created an SDN network using the Mininet emulator. We used RYU [101] as an SDN controller and

the OpenFlow protocol version 1.3 as a southbound interface between the data plane components and the SDN controller. As data plane components, we used virtual switches that support the OpenFlow protocol. We built a data plane layer that contains 38 virtual switches. The virtual switches are randomly connected, and each switch is connected to 3 hosts. To avoid network architecture dependency, different data plane topologies are created by regularly and randomly changing the connection between switches throughout the generation period of the dataset. In order to have dynamic traffic as in a real network, we used Iperf to generate UDP and TCP traffic, the D-ITG generator to produce VoIP and DNS traffic and Wget to generate HTTP traffic. Also, the generated VOIP and DNS traffics follow an exponential and uniform distribution, respectively. These setups enable the dynamicity in the generated traffic of the constructed network. All these traffics are generated by randomly selecting hosts. The traffic generators are executed for 6 hours while capturing the exchanged messages between the SDN controller and the virtual switches. Then, the captured traffic was filtered to keep only the requests received by the SDN controller (*e.g.,* $Packet\_In$ messages).

Dataset preparation is crucial to achieve better forecasting performance. First, we dealt with noisy and inconsistent data by replacing outliers with the average of the sequence data. Since the processing capacity of an SDN controller is defined by the number of packets that it can handle per second, the obtained traffic is sampled every second. Thus, our data is transformed into time series with a time step size equal to 1 second. Each value in the time series represents an observation of the SDN controller load. Then, the data is divided into two partitions, train and test subsets, while maintaining the temporal order of observations. Training data is used to fit the model by adjusting its parameters. Test data is used to evaluate the obtained model after the training phase and is not used in learning phase. Our data is large

enough, thus, using 80% for training and 20% for test keeps both subsets highly representative.

4.5.3 ARIMA modeling

An ARIMA model, denoted by ARIMA(p, d, q), is composed of integrated I(d), autoregressive AR(p) and moving average MA(q) components, where d is the number of differencing required to make the time series stationary; p is the number of lag observations included in the model; and q is the number of lagged forecast errors in the prediction equation. In order to build an effective forecasting model, the three parameters d, p and q must be carefully determined.

1) Differencing Order

The purpose of differencing a time series is to make it stationary (*i.e.,* the properties of the time series doesn't depend on the time). In order to study the stationarity and define the right degree of differencing, we analyzed the AutoCorrelation Function (ACF). A time series is said to be stationary if the ACF plot quickly reaches zero, while the ACF of a non-stationary time series slowly decreases to zero. In figure 4.2, we observed that the ACF of the original time series has positive autocorrelations for a large number of lags (more than 10 lags), thus, the time series needs to be differentiated. Also, if the autocorrelation of lag 1 is too negative, less than -0.5, the time series is over-differentiated. Therefore, looking at the ACF plot of the 1st differencing, see figure 4.3, the lag 1 autocorrelation is greater than -0.5, which indicates that the time series is not over-differentiated. Accordingly, the optimal degree of differencing for this time series is $d = 1$. To remove all traces of autocorrelations in the residuals, autoregressive and moving average terms should be added.

2) Autoregressive and moving average order

The next step in building the ARIMA model for our time series is the identification of the autoregressive degree p and the moving average de-

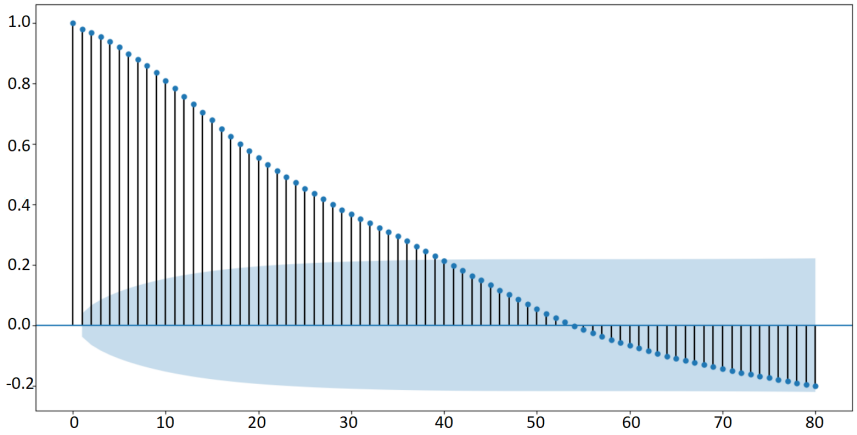


Figure 4.2 Autocorrelation function of the original series

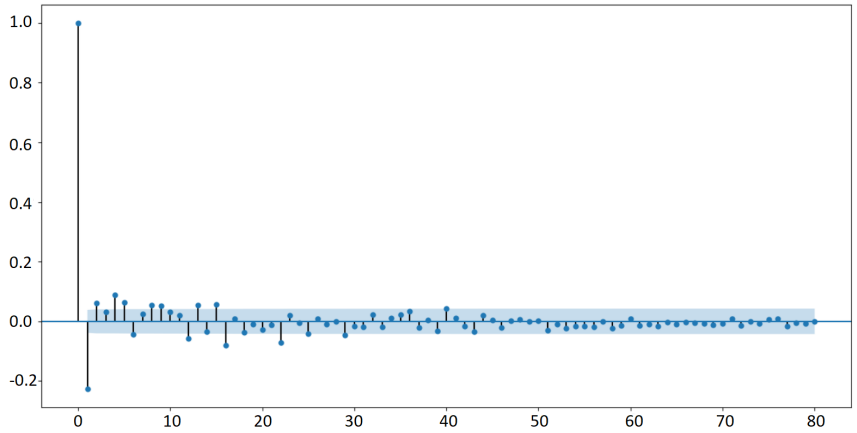


Figure 4.3 Autocorrelation function of 1st differencing

gree q. ARIMA model selection requires fitting multiple models on the prepared dataset (*i.e.,* estimating the performance of several models) and then choosing the best one of them. A model is considered the best if its performance on the training dataset is good and its complexity is low. We can use the Partial AutoCorrelation Function (PACF) and the ACF to approximately figure out how many autoregressive and moving average terms, respectively, are required to remove any autocorrelation in the stationary time series. However, inspecting the PACF and ACF plots may not lead us to identify the optimal AR and MA degrees. In order to obtain an optimal ARIMA model we employed *Akaike Information Criterium* (AIC) method to select AR and MA degrees. AIC is an

effective way to choose a model that has a good fit but few parameters (*i.e.,* p and q in our case) and is given by $AIC = 2K - 2ln(L)$, where K is the number of parameters in the model (*e.g.,* p, q degrees) and L is the likelihood of the data. Since we determined before the degree of differencing, $d = 1$, we varied the values of p and q to find the smallest AIC result. Table 4.1 summarizes the obtained results. Accordingly, the best model is ARIMA (4,1,3).

Table 4.1 AIC results.

Model	**AIC**
ARIMA(1,1,1)	29075.185
ARIMA(0,1,0)	29207.288
ARIMA(1,1,0)	29073.316
ARIMA(0,1,1)	29081.751
ARIMA(2,1,1)	29058.922
ARIMA(3,1,0)	29047.015
ARIMA(3,1,2)	29016.949
ARIMA(4,1,3)	28997.957
ARIMA(4,1,2)	29007.532

3) Evaluation of the obtained ARIMA model

All steps previously followed to build an adequate ARIMA model with the time series, namely the identification of the differentiation degree, the AR degree and the MA degree, aim to build an effective and parsimonious model. In order to evaluate the obtained model, *i.e.,* ARIMA (4,1,3), we examined the residuals plot to ensure there are no patterns. Figure 4.4 and figure 4.5 illustrate the residual errors plot and the density plot, respectively. The residual errors seem to be stationary, *i.e.,* fluctuation around a mean of zero, and the density plot shows a Gaussian distribution with mean zero. Figure 4.6 depicts the ACF plot of the residual errors in which we can observe that all autocorrelations are within the threshold limits, *i.e.,* no autocorrelations between the residuals. Therefore, the fitted ARIMA(4,1,3) model is excellent.

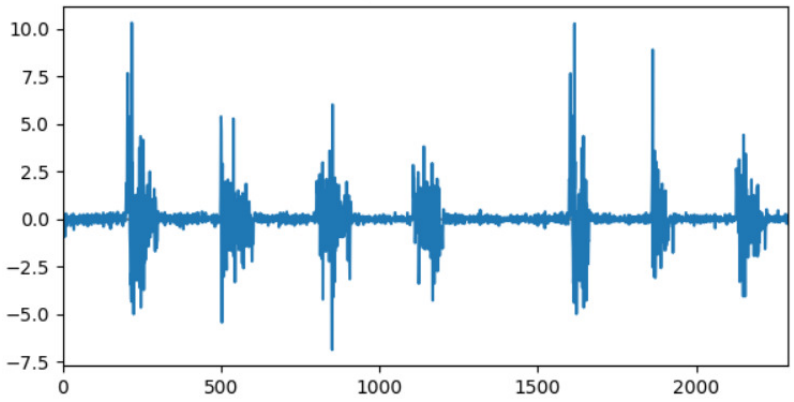


Figure 4.4 Residual errors of the fitted ARIMA(4,1,3) model.

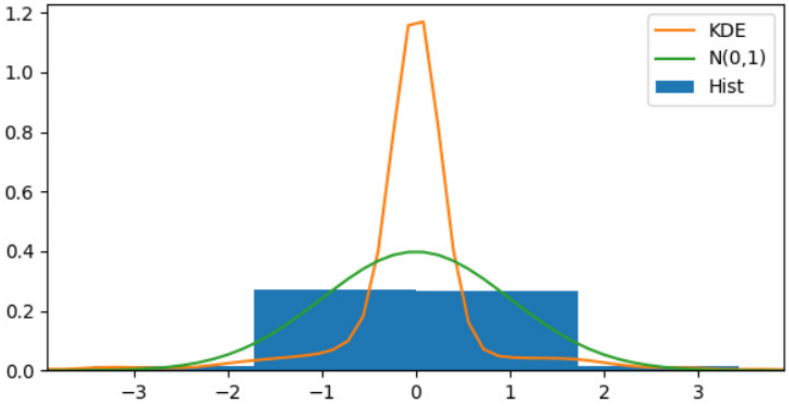


Figure 4.5 Residual errors density of the fitted ARIMA(4,1,3) model.

4.5.4 LSTM network modeling

To select an LSTM network that can make predictions with high accuracy, several hyperparameters, such as the number of hidden layers, the number of neurons of each layer, the loss function, the optimizer and the number of previous observations, need to be tuned. The hyperparameter tuning process refers to find the best combination of hyperparameters to obtain a better performance. Before tuning the hyperparameters of the LSTM network, we first differentiated the time series to make it stationary. According to the construction process of the ARIMA model, performing first order differencing is sufficient to make the time series stationary. Then, we transformed the time series values to be on a scaled between 0 and 1. Data scaling is important

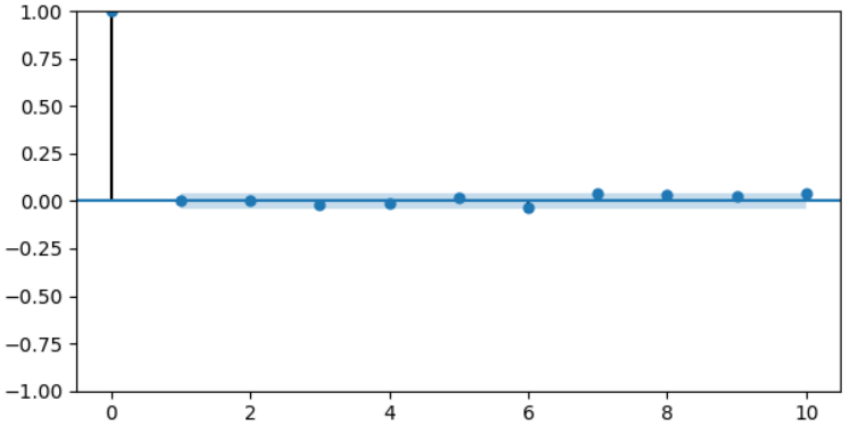


Figure 4.6 Autocorrelation function of residual errors.

because it avoids large input values that could slow down the leaning and convergence of the LSTM network. To provide an unbiased evaluation of the fitted model on the training dataset, we used the validation dataset. The model consults the validation dataset to validate its performance and not to learn from it.

In this work we performed the random search method to find the best solution for the built LSTM model. Based on this method, random combinations of the hyperparameters are used to train the LSTM model. The utilization of random search method allows to explicitly control the used parameters and the number of attempted combinations. After training and tuning multiple combinations, the most important hyperparameters of the retained LSTM model are summarized in table 4.2.

Table 4.2 Retained Hyperparameters for LSTM network.

Hyperparameter	**Value**
Number of hidden layers	3
Number of neurons in each layer	200
Activation function in all layers	Relu
Optimizer	Adam
Loss function	Mean Squared Error
Number of previous observations	200

4.6 Prediction performance evaluation

To compare the performance of ARIMA and LSTM in terms of short-term and long-term prediction, we present, in this section, the metrics used to evaluate the studied models. Then, we show the prediction results of each model and compare their performance in SDN controller load prediction.

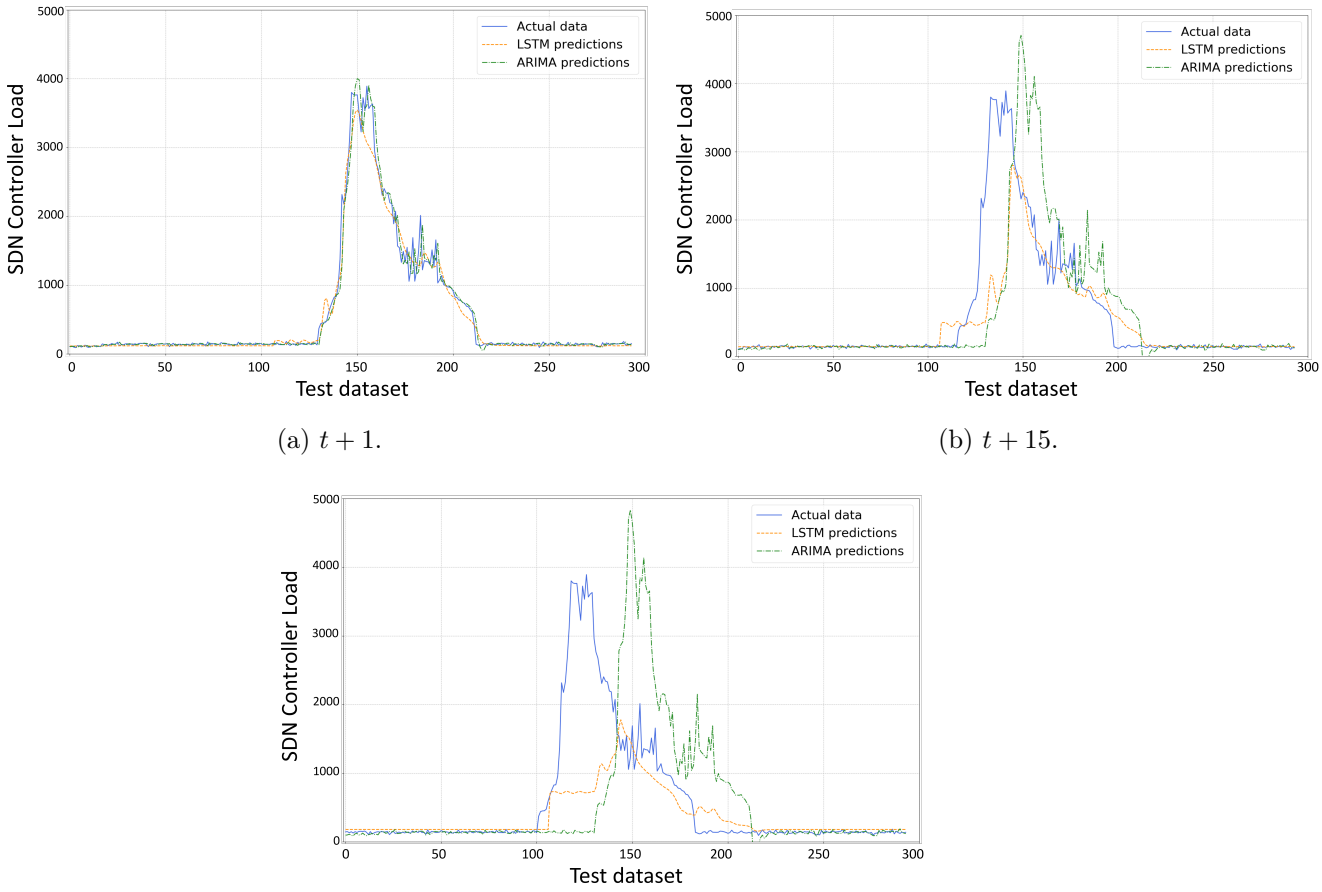
4.6.1 Evaluation metrics

To assess the performance of the two prediction models, *i.e.,* ARIMA and LSTM, in terms of accuracy, we used the Root Mean Square Error (RMSE), the Coefficient of Determination R^2, the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE) as evaluation metrics. For each step w (*i.e.,* at time $t + w$). In RMSE formula the errors are squared before they are averaged, which penalizes large errors. R^2 the square of the correlation between the actual and predicted values. MAE is the average of all absolute errors of predictions and it is a linear score, which means that all the individual differences are weighted equally in the average. MAPE measures the accuracy of a model as a percentage.

4.6.2 Prediction results

Using ARIMA and LSTM models, we performed multi-step predictions. The experiment involved forecasting the controller load from 1 to 30 seconds into the future. As mentioned before, we used the 20% of the dataset (*i.e.,* dataset of tests) to compare ARIMA and LSTM in terms of prediction accuracy.

Figure 4.7a, 4.7b and 4.7c illustrate the prediction results of SDN controller load for step 1, 15 and 30, respectively. From figure 4.7a, we observe that both models achieved good forecast performance for short-term predictions t+1. In fact, ARIMA and LSTM predictions for step $t + 1$ closely follow the actual (*i.e.,* real values) SDN controller load, displaying similar patterns. On the one hand, ARIMA prediction results slightly outperform those of LSTM in step $t + 1$ with 4.91% and 13.78 of difference in terms of MAPE



(a) $t + 1$.

(b) $t + 15$.

(c) $t + 30$.

Figure 4.7 SDN controller load prediction results of ARIMA and LSTM models for different steps.

and MAE, respectively. On the other hand, from table 4.3, LSTM model has the minimum score in terms of MAE and MAPE for step t+15 and t+30. We can note that LSTM model can follow some peaks of SDN controller load fluctuations, figure 4.7b and figure 4.7c. The percentage error (*i.e.*, MAPE) of the ARIMA model predictions in step 30 reached 99.99% while the MAPE of the LSTM model stopped at 45%. With this difference of 55%, the performance of LSTM is much better in long-term predictions than the ARIMA model. In addition, ARIMA predictions results for the three steps, namely $t + 1$, $t + 15$ and $t + 30$, have roughly the same outline, which is explained by the fact that an ARIMA model try always to follow the mean

value. Furthermore, to compare the accuracy of the ARIMA and LSTM

Table 4.3 MAE and MAPE scores of ARIMA and LSTM.

Metrics	**Forecasting step**	**ARIMA**	**LSTM**
MAE	t+1	64.04	77.82
	t+15	353.94	204.74
	t+30	573.71	296.95
MAPE(%)	t+1	12.49	17.40
	t+15	50.02	32.16
	t+30	99.99	45.32

models, the RMSE score and the coefficient of determination R^2 were plotted from step t+1 to step t+30. Figure 4.8 shows that the RMSE score increases proportionally with the prediction steps. In fact, the errors in the prediction results of the two models, *i.e.,* ARIMA and LSTM, become high where the forecasts are made for farther stages in the future. However, the LSTM model has better results in long-term predictions than ARIMA with a difference of 400.31 in terms of RMSE score for t+30, while ARIMA is still lightly good in short-term predictions. Similarly, we plotted and analyzed the R^2 scores in figure 4.9. R^2 varies between 0 and 1, *i.e.,* between a weak prediction results when R^2 is close to 0 and a good prediction results when R^2 is close to 1. From figure 4.9, we can see that the R^2 scores of both models decrease tend to 0 as the prediction step increases. However, the R^2 of ARIMA model quickly reaches 0, starting from step 18, while the R^2 of LSTM achieves 0.35 at step t+30.

Using the predefined evaluation metrics, we compared the accuracy of ARIMA and LSTM to predict, in short-term and long-term, the load of SDN controllers. When a load imbalance is predicted in a distributed SDN control plane, we need to judiciously choose which data plane component to migrate and where the migration should happen to guarantee an efficient migration operation. Thus, the next step is to properly design a migration-based model to balance the load between controllers while considering a migration cost.

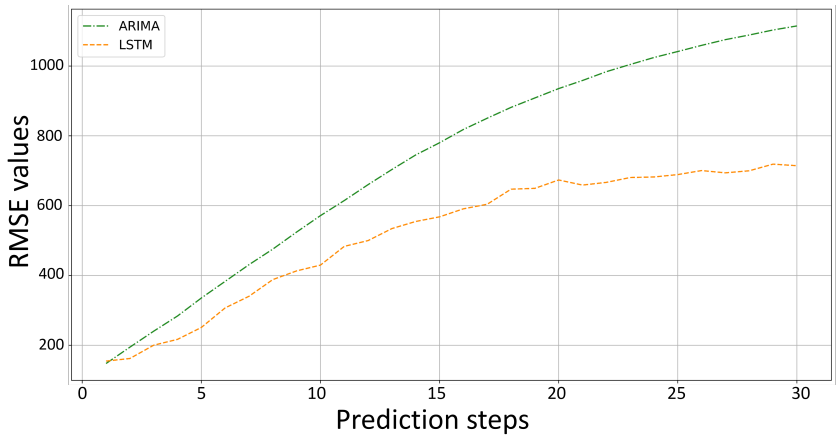


Figure 4.8 RMSE of ARIMA and LSTM from t+1 to t+30.

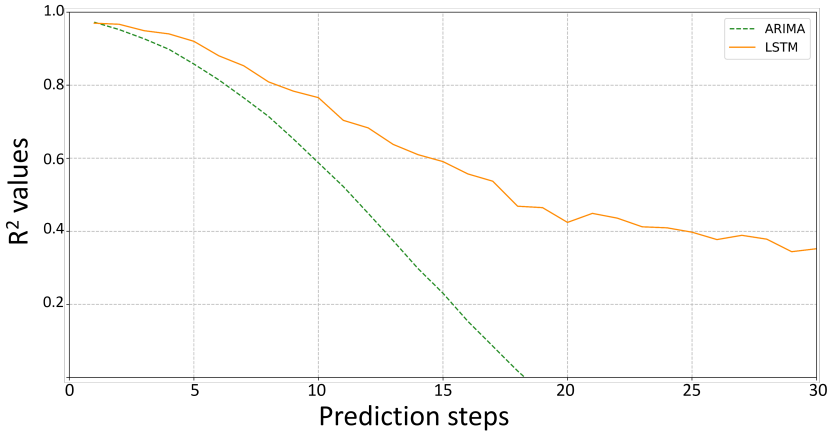


Figure 4.9 R^2 of ARIMA and LSTM from t+1 to t+30.

4.7 System model

4.7.1 Controller response time

We consider a distributed SDN control plane architecture composed of a finite set of SDN controllers $C = \{c_1, c_2, c_3, ..., c_n\}$, with $|C| = n$. Each controller c_k has a limited processing capacity α_k defined as the possible number of requests that can be processed per time unit,and we denote by $\alpha = \{\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n\}$ the set of capacities of the controllers. The data plane contains a finite set of components $S = \{s_1, s_2, s_3, ..., s_m\}$, with $|S| = m$. In a distributed control plane architecture, each controller manages a sub-set of data plane components. Accordingly, we define the association between controllers and data plane components as a binary matrix $X_{n \times m}$,

where $x_{ki} = 1$ if and only if c_k manages the data plane component s_i. As explained before, the load of an SDN controller is the sum of the *Packet_In* messages sent by the data plane components. Therefore, the load of the k^{th} controller can be calculated as $L_k = \sum_{i=1}^{m} x_{ki} * \ l_i$, where l_i is the number of requests sent by s_i.

In order to calculate the response time of a controller, we consider the following assumptions: (i) the arrival process of the number of requests follows a Poisson distribution (*i.e., Packet_In* messages) whereas the inter-arrival times follows an exponential distribution; (ii) the processing times (*i.e.,* service times) of the requests, within an SDN controller, are independent of each other, independent of the arrival process and obey an exponential distribution. Accordingly, the SDN controller can be modeled as an M/M/1 queuing system. By applying Little's law, we calculate the response time of controller c_k as follows:

$$T_{r_k} = \frac{1}{\alpha_k - L_k}. \tag{4.1}$$

To meet the requirements of delay sensitive applications, the response time of controllers should be adequate with the latency required by these applications. Thus, the response time of each controller must not exceed a threshold T_{th}.

From equation 4.1 we notice that the response time T_{r_k} of controller c_k is proportional to its load L_k. Thus, performing load balancing in the control plane maintains fairness between controllers in terms of response time. To measure how fairly the load is distributed among controllers, we define the load balancing factor LB as follows:

$$LB = \sum_{k=1}^{n} \sum_{k'>k}^{n} \left| T_{r_k} - T_{r_{k'}} \right|. \tag{4.2}$$

In equation 4.2, the lower the difference between the response times, the greater is the load balancing in the control plane.

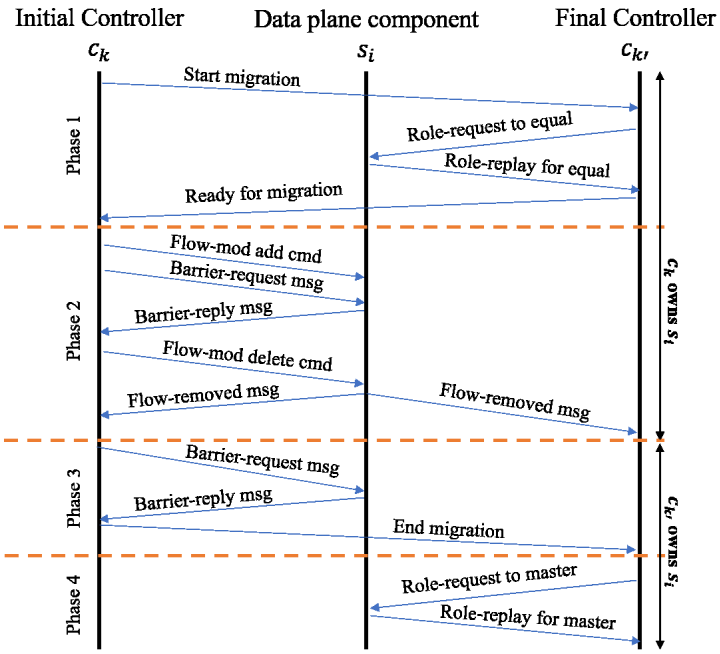


Figure 4.10 Messages exchanged between the data plane component, the initial controller and the final controller during a migration operation.

4.7.2 Migration protocol

In a migration operation, when a data plane component s_i migrates from controller c_k to controller $c_{k'}$, the former is called the initial controller and the later is called the final controller. In order to ensure a normal operation of s_i, c_k and $c_{k'}$ during a migration operation, a migration protocol [42] should be implemented, which includes four phases, as shown in figure 4.10: (i) change role of the final controller to Equal, (ii) insert and remove a dummy flow, (iii) flush pending requests with a barrier, and (iv) change role of the final controller to Master. The first phase consists in changing the role of the final controller from Slave to Equal. For this reason, the final controller should respond to two received messages, the first one is sent by the initial controller (*i.e.,* Start migration) and the second one is an asynchronous message sent by the migrated data plane component (*i.e.,* Role-reply message for Equal). In the end of this phase, the final controller sends to the initial controller a message indicating that it is ready for the migration operation. Consequently, in the second phase, the initial controller sends two successive messages (*i.e.,*

a Flow-mod add command and a Barrier request message) to the migrated data plane component as a response to the last message from the final controller. Next, in response to the Barrier-replay message, the initial controller removes the flow table entry added by the Flow-mod command. In the third phase, the initial controller sends another Barrier message to the migrated data plane component as a consecutive command upon receiving the Flow-Removed message in the second phase. This message is used by the controller to ensure that all previous requests are processed by the data plane component before detaching from its management as a Master controller. Then, as a response to the Barrier-replay message, the initial controller sends a message to the final controller indicating the end of the migration. Finally, in the fourth phase, the final controller changes its role from Equal to Master by sending a Role-request message to the migrated data plane component.

By analyzing the behavior of the initial and the final controllers in the four-phase migration protocol, we can notice that the initial controller should react four times after receiving four messages and the final controller should react three times after receiving three messages. The four responses of the initial controller are: (i) Flow-mod add command and Barrier request message, (ii) Flow-mod delete command, (iii) Barrier request message, and (iv) End-migration.The three responses of the final controller are: (i) Ready for migration, (ii) Role-request message to Equal, and (iii) Role-request message to Master.

4.7.3 Migration cost

We denote the initial association matrix by $X_{n\times m}^{initial}$ and the final association matrix, after performing migration operations, by $X_{n\times m}^{final}$. The logical XOR operation between $X_{n\times m}^{initial}$ and $X_{n\times m}^{final}$ defines the binary migration matrix $M_{n\times m}$, *i.e.*, $M_{n\times m} = X_{n\times m}^{inital} \oplus X_{n\times m}^{final}$, where $m_{ki} = 1$ if and only if c_k is either an initial or a final controller of s_i.

To present the controllers involved in migration operations as initial controllers, we define the binary matrix $M^{initial}_{n\times m} = M_{n\times m} \bullet X^{inital}_{n\times m}$, where $\bullet$ is the AND operator and $m^{initial}_{ki} = 1$ if and only if c_k is the initial controller of s_i. Similarly, we define $M^{final}_{n\times m} = M_{n\times m} \bullet X^{final}_{n\times m}$ to present the controllers involved in migration as final controllers, where $m^{final}_{ki} = 1$ if and only if c_k is the final controller of s_i.

Using the OpenFlow-based migration mechanism, the migration operation of a data plane component is performed by exchanging a set of messages, as shown in figure 4.10, between this component, the initial controller and the final controller. Therefore, we define the migration cost of a data plane component based on the time required to exchange all of these messages. This needed time for the accomplishment of a migration operation depends on the transmission delay of these messages, the response time of the initial controller, the response time of the final controller and the response time of the data plane component. Since the response time of a controller is usually much greater than the transmission delay and the response time of the data plane component [102][103], we assume that the cost of a migration operation depends only on the response time of the initial and final controller. Hence, the cost of migrating s_i can be defined as follows:

$$T_{m_i} = \sum_{k=1}^{n} 4 * T_{r_k} * m^{initial}_{ki} + \sum_{k'=1}^{n} 3 * T_{r_{k'}} * m^{final}_{k'i}. \tag{4.3}$$

In equation 4.3, the first sum calculates the response time of the initial controller required to migrate s_i and the second sum calculates the needed response time of the final controller. Since the initial controller should respond four times after receiving four messages and the final controller should respond three times after receiving three messages (see Section VI-B), we multiply the response time of the initial controller by four and that of the final controller

by three. In addition, we define the total migration cost T_m as follows:

$$T_m = \sum_{i=1}^{m} T_{m_i} \tag{4.4}$$

Performing many migration operations increases the migration cost, which affects the performance of controllers. Since we are studying the SDN load balancing problem for delay-sensitive applications on the edge [104], a significant migration cost heavily impacts the services offered by these applications. Accordingly, the objective of LBSDA is to minimize the LB factor through migration operations, but not at the expense of the migration cost. To optimally solve the LBDSA problem, we formulate it as an optimization problem in the next section.

4.8 Problem formulation and NP-hardness

4.8.1 Problem formulation

In order to optimize the load balancing and the migration cost, we transform the two objectives, LB and T_m, into a weighted sum by introducing a weight factor $\omega \in [0, 1]$. Consequently, LBDSA can be formulated as follows:

$$\underset{X,\,M}{\text{minimize}} \quad \omega \times LB + (1 - \omega) \times T_m \tag{4.5a}$$

$$\text{subject to} \quad \sum_{k=1}^{n} x_{ki} = 1, \forall i \in \{1, 2, 3, ..., m\}, \tag{4.5b}$$

$$L_k < \alpha_k, \forall k \in \{1, 2, 3, ..., n\}, \tag{4.5c}$$

$$T_{r_k} \leq T_{th}, \forall k \in \{1, 2, 3, ..., n\}, \tag{4.5d}$$

$$x_{ki}, m_{ki}, m_{ki}^{initial}, m_{ki}^{final} \in \{0, 1\}. \tag{4.5e}$$

Equation (4.5a) formulates the weighted summation of the two objectives. Constraints (4.5b) ensure that each data plane component must be assigned to exactly one controller. Constraints (4.5c) guarantee that the load of each

controller should not exceed its processing capacity. Constraints (4.5d) state that the response time of each controller cannot exceed a defined threshold T_{th} related to latency required by delay sensitive applications. Constraint (4.5e) lists the optimization variables.

Due to the non-linearity of equation 4.1, LBDSA is a non-linear binary programing problem.

4.8.2 NP-hardness

We denote the LBDSA problem (equation 4.5) here by P. To prove that P is NP-complete we (i) consider a decision version of the problem, (ii) show that P belongs to the non-deterministic polynomial time class (NP), and (iii) reduce, on a polynomial-time, PARTITION problem [105] to P. In [79], the reduction from PARTITION problem has been used to demonstrate the NP-completeness of a similar load balancing problem in the SDN control plane. Since our problem is different from the one in [79], we cannot argue that our problem is NP-hard. A reduction from the problem of [79] to our problem is probably possible but it is not evident as both problems have different objectives and constraints. To present a simple proof, we use partition to prove the NP-hardness of our problem.

Definition 1 (PARTITION problem)**.** *The PARTITION problem is a decision problem and is defined as follows: given a set S of integers, the task is to find if S can be divided into two subsets S_1 and S_2 such that the sum of all elements in S_1 equals the sum of all elements in S_2, i.e., $S_1 \cup S_2 = S$ and $\sum_{s_1 \in S_1} s_1 = \sum_{s_2 \in S_2} s_2$.*

Theorem 1. *P is NP-complete.*

Proof. We prove the theorem by restriction, *i.e.,* we prove that a restricted version of P is NP-complete. First, we prove that P is in NP. This is true since the knowledge of an association matrix $X_{n \times m}$ between data plane com-

ponents and controllers makes it is possible to verify, in polynomial time, that the processing capacity (α) of the controllers has not been violated and the response time (T_r) of the controllers does not exceed the defined threshold (T_{th}).

PARTITION problem can be stated alternatively as follows: given a finite set S and each element $s \in S$ present a load value $load(s) \in \mathbb{N}$, is there a subset $S' \subseteq S$ such that $\sum_{s \in S'} load(s) = \sum_{s \in S \setminus S'} load(s)$?

The restricted version of P is constructed as follows. Let $C = \{c_1, c_2\}$, $\omega = 1$ and $\alpha_1 = \alpha_2$. Each s_i represents an element $s \in S$ and has a number of requests (*i.e.,* load) $l_i = load(s_i)$. Then, given a YES solution S_1 and S_2 for PARTITION problem, we can create an association matrix $X_{n \times m}$ as a solution to P as follows, $x_{1i} = 1$ for all $i \in S_1$ and $x_{2i} = 1$ for all $i \in S_2$. We obtain $\sum_{i=1}^{m} x_{1i} \times l_i = \sum_{i=1}^{m} x_{2i} \times l_i$, so $L_1 = L_2$. Since, we choose $\alpha_1 = \alpha_2$, then $T_{r_1} = T_{r_2}$. Thus, we have a solution $x_{ki} \in \{0, 1\}, \forall i \in \{1, 2, ..., m\}$ and $\forall\ k \in \{1, 2\}$ with $LB = 0$. The reverse part can be done similarly. Since PARTITION problem is NP-complete, the reduction is done in polynomial-time and P is in NP, this proves that P is NP-complete which proves the theorem. □

We showed that LBDSA is NP-complete, and thus it is very challenging to solve optimally, specifically in large-scale networks. To overcome this issue, we invoke machine learning tools for performing the association task between data plane components and SDN controllers in a computationally efficient manner. In particular, we use Reinforcement Learning (RL) to efficiently obtain the association solution to LBDSA. In the next section, we describe our proposed RL algorithm.

4.9 Reinforcement learning algorithm

In RL, an agent interacts with its environment and its aim is to make suboptimal decisions based on its previous experience. The major challenge the agent is facing is to find a tradeoff between exploitation, the desire to make the best decision given current information, and exploration, the desire to try a new decision which may lead to better results. While interacting with its environment, the agent observes the state of the environment and performs an action using a certain policy. According to the performed action, the state of the environment changes and the agent receives a reward or penalty. The agent would like to maximize its accumulated reward (or minimize its accumulated penalty) in the long run. Every time the agent observes the state of the environment, performs an action it learns from this experience and refines its policy. This process is repeated until a suboptimal decision is found. The details are given in the sequel.

4.9.1 Preliminary definitions

First, we assume that the root controller includes a RL agent that knows the necessary network information about this architecture, namely the control domain of each controller, the load of controllers and the response time of controllers. In other words, we assume that the root controller represents the RL agent which will perform the learning process.

For LBDSA, we define the state set, action set and rewards as follows.

States. The observed state by the RL agent is given by the data plane components managed by each controller, the load of controllers, the response time of controllers (equation 4.1) and the achieved load balancing level LB.

Actions. An action represents the decision to choose a controller $c_k \in C$ as the final controller for a data plane component $s_i \in S$. To define the global action of the RL agent, first we define the set of actions available to migrate s_i as C, *i.e.,* the set of controllers. We denote the action for data plane

component s_i by $a_i \in \{c_1, c_2, ..., c_n\}$. If a_i is the initial controller for data plane component s_i, it means that this data plane component will not be migrated.

The RL agent simultaneously chooses an action for each data plane component s_i. Therefore, we define the global action, denoted by a, of the RL agent and it's represented as a vector of size $|S| = m$, where $a \triangleq [a_1, a_2, ..., a_m]$. Note that a global action vector a is equivalent to an association matrix $X_{(n \times m)}$ because each element a_i of vector a corresponds to the master controller of the data plane component s_i. A global action a is ***feasible*** if it meets the constraints (4.5b), (4.5c) and (4.5d) of the LBDSA problem.

Rewards. After performing the actions for all data plane components and obtaining the global action, the RL agent receives a set of rewards denoted by $\{\mathcal{R}(a_i, a_{-i})\}_{i \in \{1,2,...m\}}$. Each reward $\mathcal{R}(a_i, a_{-i})$ depends both on the chosen action a_i for data plane component s_i and on the global action a. We define $\mathcal{R}(a_i, a_{-i})$ to belong to $\{\delta_1, ..., \delta_4\}$ where δ_1, δ_2, δ_3 and δ_4 are any real numbers such that $\delta_1 > \delta_2 > \delta_3 > \delta_4$. For $j \in \{1, 2, 3, 4\}$, the reward $\mathcal{R}(a_i, a_{-i})$ is set to δ_j if condition ζ_j is true. The conditions ζ_j are given by:

1) ζ_1: the global action a is feasible, the data plane component s_i has not been migrated and the LB factor has decreased.

2) ζ_2: the global action a is feasible, the data plane component s_i has been migrated and the LB factor has decreased.

3) ζ_3: the global action a is feasible, the data plane component s_i has been migrated and the LB factor has increased.

4) ζ_4: the global action a is not feasible.

Next, we define how the agent responds to each action to find a suboptimal policy. Our approach is based on the well-known learning rule Win-Stay-Lose-Shift Algorithm [95].

4.9.2 Two-win-stay-lose-shift algorithm (2WSLS)

In order to solve LBDSA based on the RL framework, we developed 2WSLS for two-win-stay-lose shift algorithm which is derived from the win-stay-lose-shift learning strategy. In the original WSLS algorithm, an action is either a winning action or a losing one while in our algorithm, we consider two types of winning actions, hence the name 2WSLS. Given an initial association between controllers and data plane component, 2WSLS aims to solve LBDSA by finding a suboptimal tradeoff between the load balancing and the migration cost while respecting the constraints (4.5b), (4.5c) and (4.5d). In the following, we describe the first iteration of the algorithm, the learning process and the termination process.

Algorithm 3 2WSLS for data plane component s_i

Input: $\mathcal{A}$, τ_1, τ_2, ϵ, π_i
Output: action a_i

if $\mathcal{R}(a_i, a) = \delta_1$ **then**
  $\pi_i[a_i] \leftarrow \pi_i[a_i] + \tau_1(1 - \pi_i[a_i])$
  for $a_j \in A$ **do**
    if $a_j \neq a_i$ **then**
      $\pi_i[a_j] \leftarrow \pi_i[a_j] - \tau_1 \pi_i[a_j]$
    end if
  end for
else if $\mathcal{R}(a_i, a) = \delta_2$ **then**
  $\pi_i[a_i] \leftarrow \pi_i[a_i] + \tau_2(1 - \pi_i[a_i])$
  for $a_j \in A$ **do**
    if $a_j \neq a_i$ **then**
      $\pi_i[a_j] \leftarrow \pi_i[a_j] - \tau_2 \pi_i[a_j]$
    end if
  end for
else if $\mathcal{R}(a_i, a) = \delta_3$ **then**
  $\pi_i[a_i] \leftarrow \pi_i[a_i] - \epsilon$
  $\pi_i[c_{initial}] \leftarrow \pi_i[c_{initial}] + \epsilon$
end if
return a_i

1) First iteration

The RL agent simulates 2WSLS independently for each data plane component. Every data plane component chooses a random action a_i from

its action space C. Once every data plane component has chosen an action, the RL agent formulates the global action $a = [a_1, a_2, ..., a_m]$ and calculates the load and the response time of each controller. If the global action is feasible, the agent calculates the rewards according to the constraints ζ_1, ζ_2, ζ_3 and ζ_4.

According to the computed rewards and the formulated global action, the RL agent can have information about the reliability of certain actions chosen for the data plane components. Indeed, when a global action is feasible the agent can find out if: (i) the non-migration of s_i can increase the LB factor, *i.e.,* $\mathcal{R}(a_i, a_{-i}) = \delta_1$ and (ii) the migration of s_i can decrease or increase the LB factor, *i.e.,* $\mathcal{R}(a_i, a_{-i}) = \delta_2$ or δ_3, respectively. Whereas, when the global action is not feasible, the agent cannot know whether the migration or non-migration of a specific data plane component is beneficial. The acquired information should be efficiently used by the agent to choose a better action in the future. Therefore, we define for each $s_i \in S$ a probability vector $\pi_i = [\pi_i[a_1], \pi_i[a_2], ..., \pi_i[a_n]]$ of size $n = |C|$. Each element $\pi_i[a_k]$ for all $k \in \{1, 2, ..., n\}$ corresponds to the probability of playing action $a_i = c_k$ by RL agent for the data plane component s_i. Part of the 2WSLS pseudo-code executed by the RL agent for data plane component s_i is presented in Algorithm 3.

2) Learning Process

Once the RL agent computes the rewards and associates them to the data plane components, it updates the probability vectors of the data plane components as shown in Algorithm 3. These updates should efficiently reflect the results that the RL agent received to improve the learning process. In WSLS, there are two types of actions: (i) a winning action if the received reward is higher and (ii) a losing action if the received reward is not good. In the case of a winning action, the RL agent continues to play this action in future iterations while in the case of a losing action,

the RL agent should try another action. The WSLS original learning algorithm has been shown to be commonly used in binary rewards choice problems. However, the modeling must be modified and improved for problems where the RL agent receives a non-binary reward after each iteration which is the case in LBDSA.

Since in 2WSLS there is a tradeoff design problem, a single winning action does not model the multi-objective correctly. Thus, we define two winning actions and one losing action. The first winning action results in the δ_1 reward and the second winning action results in the δ_2 reward. We consider that the first winning action is better than the second one ($\delta_1 > \delta_2$) because we aim to minimize the LB factor as well as the migration cost. Thus, the reward of δ_1 that corresponds to a lower LB factor and no migration should have the highest reward. In the second winning action, the LB factor decreases but a cost incurs which corresponds to the migration cost of a data plane component. Therefore, to increase the chance to converge to the two winning actions at the end of the learning process, the probabilities of playing these actions must be increased, but with a preference for the first winning action. Accordingly, for $s_i \in S$, the probabilities corresponding to the two winning actions are updated as follows:

$$\pi_i[a_i] = \begin{cases} \pi_i[a_i] + \tau_1(1 - \pi_i[a_i]), & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_1, \\ \pi_i[a_i] + \tau_2(1 - \pi_i[a_i]), & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_2, \end{cases} \tag{4.6}$$

where τ_1 and τ_2 represent the winning increment factors such that $\tau_1 > \tau_2$ because the first winning action is better than the second one. In order to keep the sum of the probabilities in π_i equal to one, all the probabilities $\pi_i[a_j]$ such that $a_j \neq a_i$ must be decreased by the same factor as follows:

$$\pi_i[a_j] = \begin{cases} \pi_i[a_j] - \tau_1 \pi_i[a_j], & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_1, \\ \pi_i[a_j] - \tau_2 \pi_i[a_j], & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_2. \end{cases} \tag{4.7}$$

When the reward is equal to δ_3, we consider the action chosen as a losing one because it leads not only to increase the LB factor but also to migrate a data plane component which increases the migration cost. Therefore, the probability of choosing this action is reduced and the probability of not migrating the associated data plane component is increased, *i.e.,* the probability of choosing the initial controller. In this way, we teach the RL agent that it is better not to migrate at all. Hence, the probability vector is updated as follows:

$$\pi_i[a_i] = \pi_i[a_i] - \epsilon, \tag{4.8}$$

$$\pi_i[c_{initial}] = \pi_i[c_{initial}] + \epsilon, \tag{4.9}$$

where ϵ represents the losing decrement factor and $c_{initial}$ is the initial controller of the data plane component s_i.

Finally, when the reward is equal to δ_4, the global action is not feasible. In this case, we cannot find out whether the migration or non-migration of a specific data plane component is beneficial to minimize the LB factor and the migration cost. Therefore, the RL agent does not perform any probability update. In other words, the RL agent must favor the exploration in order to find out better solutions.

3) 2WSLS termination

The RL agent terminates the execution of the 2WSLS algorithm when the number of iterations reaches a predefined threshold or the probability vectors converge. Then, the agent chooses the last feasible global action a which represents the final association matrix $X_{n\times m}^{final}$.

4.10 Simulation results

We conduct the simulations using three topologies with a different number of data plane components (switches), namely 20, 25 and 30 components. For all these topologies, the number of controllers is set to 5 and for the sake of simplicity, all controllers have the same capacity α. We assume that the number of *Packet_In* messages sent by the data plane components is random. To provoke a load imbalance state in the control plane, we apply great stress on at least one controller by generating a large number of *Packet_In* messages. The optimization problem (equation 4.5) is modeled in AMPL [106] and solved using the Knitro solver to obtain a global optimal solution OPT. For all simulations, we performed 100 independent random realizations which are then averaged out. Table 4.4 summarizes the key parameters of the simulations. These parameters were chosen based on common settings in the literature [46, 50].

Table 4.4 Simulation parameters

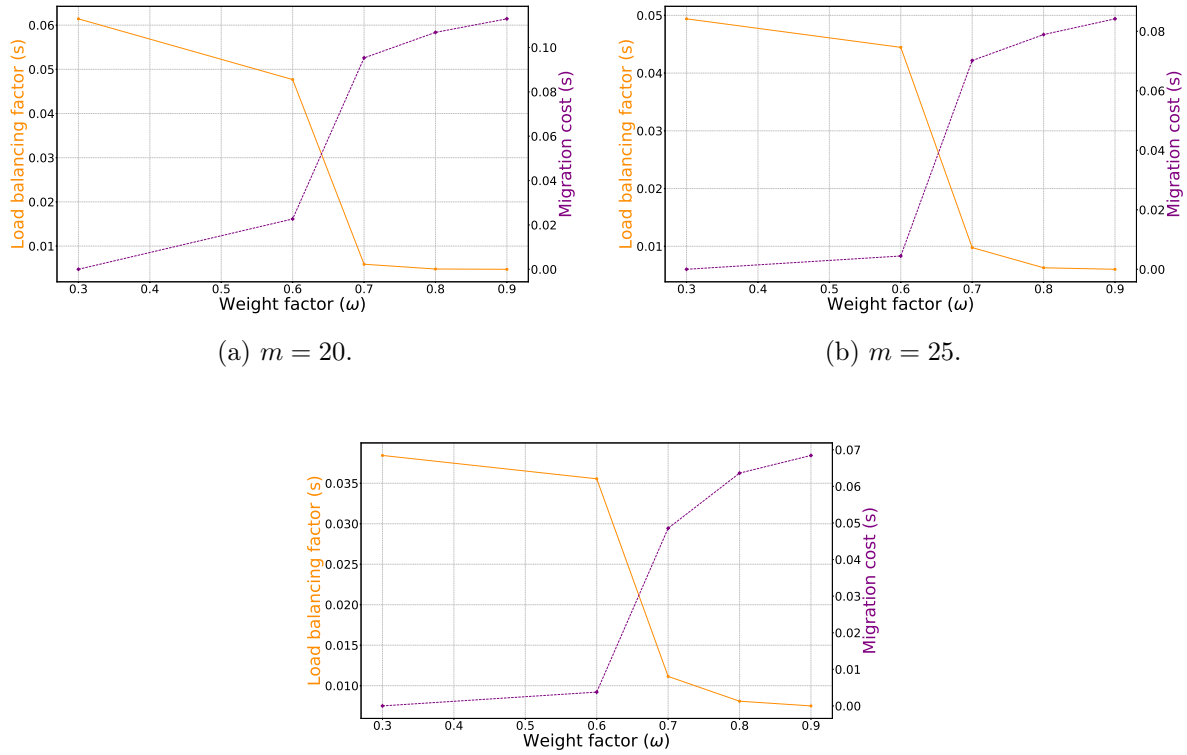
Name	Description	Value
l_i	Number of *Packet_in messages* sent by s_i	[5-500]
α	Capacity of controllers	2000
T_{th}	Controller response time threshold	0.005 s
R	Number of iterations of 2WSLS	500
R_1	Number of realizations of 2WSLS	100

To benchmark our 2WSLS algorithm, we implemented two algorithms from the literature called MCBLB for migration competency-based load balancing [46] and SMCLBRT for SDN multiple controller load-balancing strategy based on response time [50]. In order to balance the load in the SDN control plane, MCBLB uses shift and swap moves to migrate switches between controllers. The shift move is a simple migration operation of a switch from an overloaded controller to an underloaded controller. However, in some cases, shift moves are not always possible. For instance, when the only solution to offload a controller is to migrate a heavy switch, but this switch may overload

the final controller. In this case, swapping two switches can be beneficial for the load balancing. In SMCLBRT, the response time of controllers is used to figure out if a controller is overloaded or not. Indeed, based on an appropriate threshold, the controllers are divided into overloaded controllers and under-loaded controllers. Then, the heavy switches are migrated simultaneously from overloaded controllers to less-loaded ones. We adapted this algorithm by using our definition of the controller response time (equation 4.1) and the controller response time threshold defined in table 4.4. To fairly benchmark the performance of 2WSLS against the two algorithms in [50] and [46], the parameters of our simulation are chosen similar to those used in [50] and [46].

We evaluate the performance of 2WSLS in two stages. First, we analyze the parameters of the algorithm, namely the winning increment factors τ_1 and τ_2, the losing decrement factor ϵ and the number of iterations, denoted by R. Then, we compare 2WSLS to the optimal solution, denoted by OPT, MCBLB and SMCLBRT.

Before discussing the performance of 2WSLS, we start by analyzing the results obtained by solving the optimization problem (equation 4.5) through AMPL modeling and using the Knitro solver. Figure 4.11a, 4.11b and 4.11c show the LB factor (equation 4.2) and migration cost T_m (equation 4.4) values for the three topologies of 20, 25 and 30 data plane components, respectively. For each topology, we plot the LB and T_m values for different tuning tradeoff weights $w \in [0, 1]$. As expected, the results show that the more the policy is LB-oriented ($w > 0.5$), the more we sacrifice on migration cost T_m (*i.e.,* T_m increases) and vice versa. This value is almost the same for the three topologies $m = 20$, $m = 25$ and $m = 30$. From these figures, we can see that Pareto solutions to the optimization problem (equation 4.5) can be found around $w = 0.65$. Therefore, in the rest of the simulations, $w = 0.65$ is chosen as an efficient and no dominated solution to the LBDSA problem.



(a) $m = 20$.

(b) $m = 25$.

(c) $m = 30$.

Figure 4.11 The impact of the weight factor (ω) on load balancing factor and migration cost for different data plane topologies.

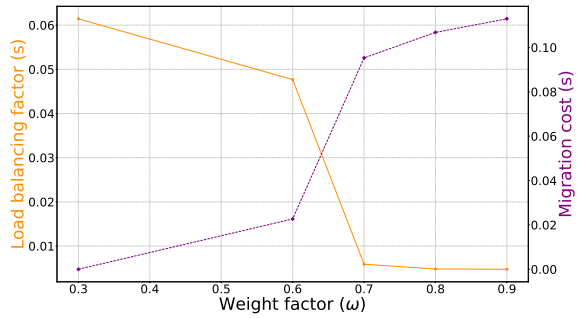
4.10.1 Parameters of 2WSLS

The performance of the 2WSLS algorithm depends on the choice of its parameters, namely the winning increment factors τ_1 and τ_2, the losing decrement factor ϵ and the number of iterations R. Accordingly, they should be chosen carefully to obtain good results. Figures 4.12, 4.13, 4.14, 4.15 and 4.16 illustrate the impact of these parameters on the objective function (equation 4.5).

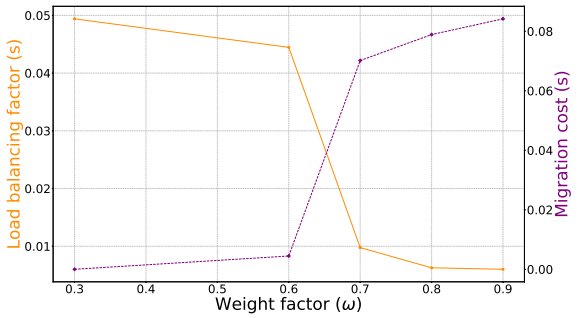
In order to figure out an optimal value of the winning increment factor τ_1, we compare the performance of 2WSLS to that of OPT for different values of τ_1. In other words, given a value of τ_1, how much 2WSLS can minimize the objective function in equation 4.5 compared to the optimal solution. Figure 4.12a, 4.12b and 4.12c show that there exists an optimal value of τ_1 at which

the performance of 2WSLS is close to the OPT value, given by $\tau_1 = 0.5$. This optimal value is almost the same for the three topologies $m = 20$, $m = 25$ and $m = 30$ which is a very good and an important observation that implies that our proposed learning algorithm is scalable. We notice, for all these topologies, that small values of τ_1 prevent the RL agent from converging to the winning actions and, thus, the performance of 2WSLS decreases. On the other hand, when the value of τ_1 is high, the probability of choosing the associated winning action will also be high. Therefore, the RL agent will not explore new actions, which may lead to better performances, but it will quickly choose few actions by simply exploiting the current information. Similarly, figure 4.13a, 4.13b and 4.13c show, for the three topologies, that there is an optimal value of the winning increment factor τ_2, given by $\tau_2 = 0.04$, at which the performance of 2WSLS is close to OPT. Again, figure 4.13 shows the scalability of 2WSLS.

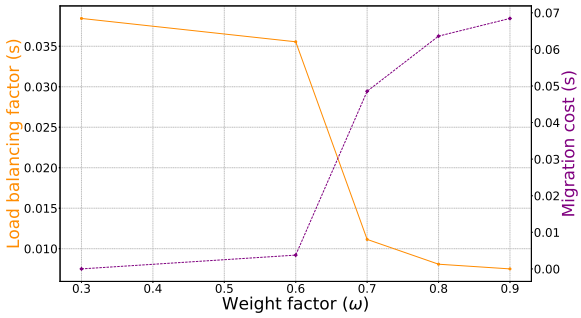
To further verify the scalability of our proposed 2WSLS algorithm, we compare its performance to that of OPT for different control plane topologies, namely $n = 5$, $n = 7$ and $n = 9$. In other words, we show, via simulations, that the optimal values of τ_1 and τ_2 are almost the same for $n = 5$, $n = 7$ and $n = 9$. To verify the 2WSLS algorithm scalability under a dense network topology, the number of data plane components is set to 30 for all these control plane topologies. Figure 4.14a, 4.14b and 4.14c illustrate that the performance of our proposed algorithm is close to OPT performance at $\tau_1 = 0.5$ for $n = 5$, $n = 7$ and $n = 9$, respectively. Figure 4.15a, 4.15b and 4.15c show, for the three control plane topologies, the existence of an optimal value of τ_2, given by $\tau_2 = 0.04$, at which 2WSLS has close-to-optimal performance. These results conform with those presented in figure 4.12 and figure 4.13 and demonstrate the scalability of our 2WSLS algorithm.



(a) $m = 20$.



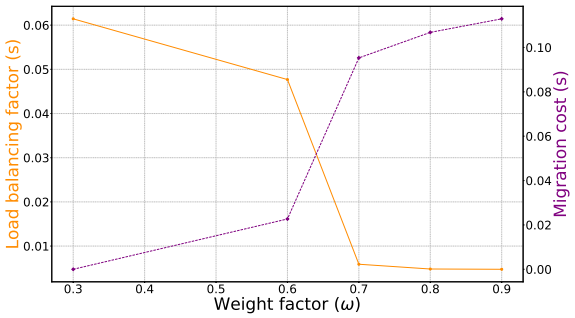
(b) $m = 25$.



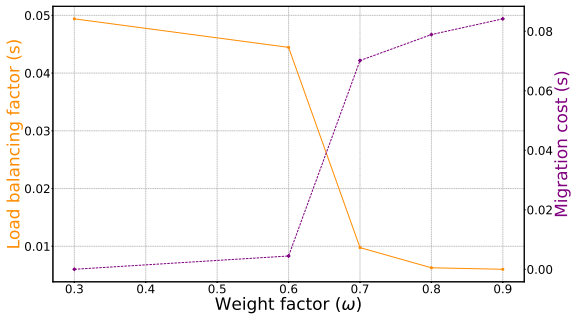
(c) $m = 30$.

Figure 4.12 The impact of the winning increment factor (τ_1) on 2WSLS for different data plane topologies.

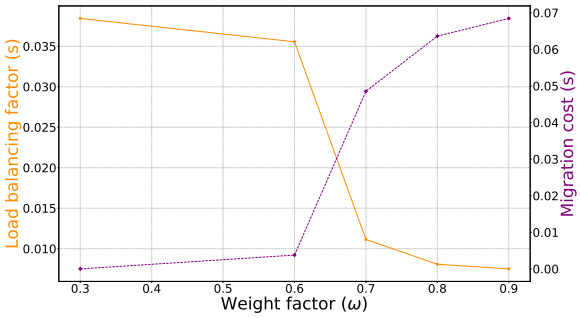
Figure 4.16 illustrates the impact of the number of iterations R on 2WSLS for different values of τ_1. We notice that the performance of the optimal value of τ_1, *i.e.*, $\tau_1 = 0.5$, outperforms the performances of the other values. These results confirm those obtained in figure 4.12 and 4.14. Indeed, higher values of τ_1 improve the performance of 2WSLS faster while they result in low performance for a large number of iterations R. For instance, the performance comparison between $\tau_1 = 0.5$ and $\tau_1 = 0.7$ shows that, with $\tau_1 = 0.7$, 2WSLS is able to reach an objective value equal to 0.046 in just 20 iterations while to reach roughly the same value, 2WSLS needs 30 iterations with $\tau_1 = 0.5$. However, $\tau_1 = 0.5$ gives better performances than $\tau_1 = 0.7$ for a large number of iterations. In the case of small values of τ_1, *i.e.*, $\tau_1 < 0.5$, we observe



(a) $m = 20$.



(b) $m = 25$.

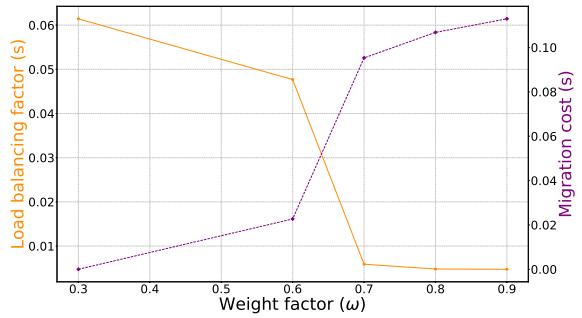


(c) $m = 30$.

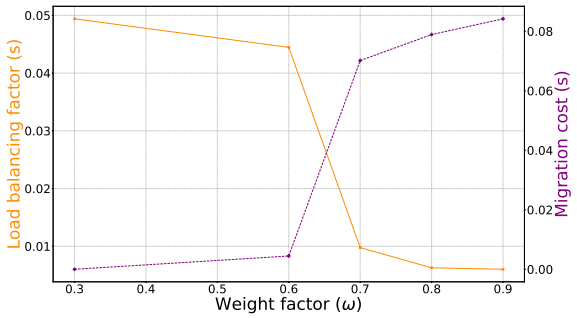
Figure 4.13 The impact of the winning increment factor (τ_2) on 2WSLS for different data plane topologies.

that 2WSLS does not converge towards good performances even with a large number of iterations, *e.g.*, $R = 500$.

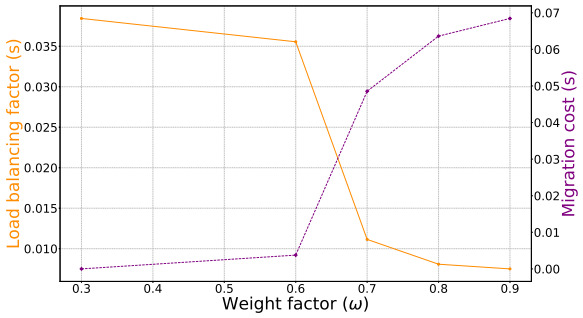
The value of δ_1, δ_2, δ_3, and δ_4 are not very important. In other words, any real number values should work fine for the proposed 2WSLS algorithm subject to the constraints that $\delta_1 > \delta_2 > \delta_3 > \delta_4$. To show that the exact values of δ_1, δ_2, δ_3, and δ_4 are not so important, we compare the performance of our proposed RL algorithm to that of OPT algorithm for different chosen random values of δ_1, δ_2, δ_3, and δ_4. Precisely, we show, via simulations, that 2WSLS converges to the same optimal values of the winning increment factors τ_1 and τ_2. For this purpose, we define the vector $\Delta = (\delta_1, \delta_2, \delta_3, \delta_4)$ and we test three different configurations of Δ. These configurations are given by $\Delta^1 = (2, 1, 0, -4)$, $\Delta^2 = (7, 5, 3, 1)$, and $\Delta^3 = (100, 70, 40, 10)$. Figure 4.17a



(a) $n = 5$.



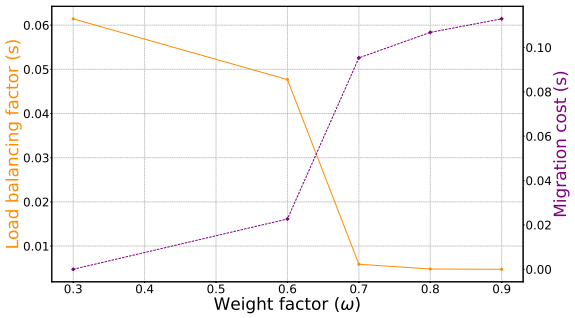
(b) $n = 7$.



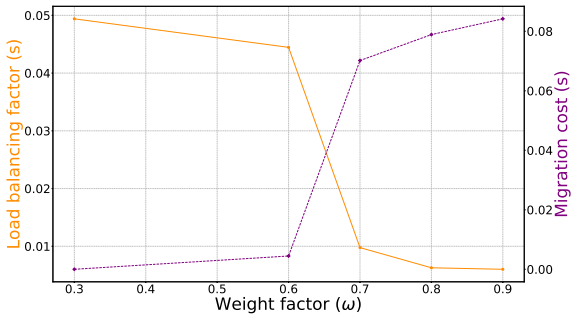
(c) $n = 9$.

Figure 4.14 The impact of the winning increment factor (τ_1) on 2WSLS for different control plane topologies.

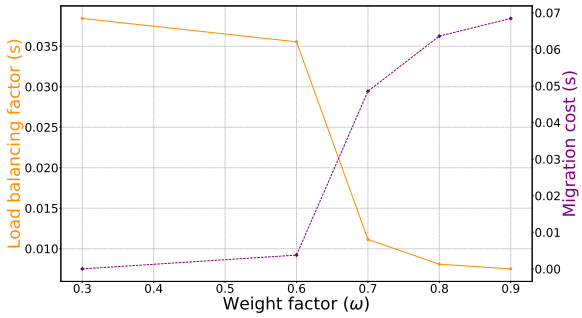
corresponding to Δ^1, figure 4.17b corresponding to Δ^2, and figure 4.17c corresponding to Δ^3, show the performance of the proposed RL algorithm with comparison to OPT algorithm when varying the winning increment factor τ_1. The three figures show the existence of an optimal value of τ_1, given by $\tau_1 = 0.5$, at which the performance of our proposed algorithm is close to optimal. Similarly, figure 4.18a corresponding to Δ^1, figure 4.18b corresponding to Δ^2, and figure 4.18c corresponding to Δ^3, show the performance of the proposed RL algorithm with comparison to OPT algorithm when varying the winning increment factor τ_2. The three figures show the existence of an optimal value of τ_2, given by $\tau_2 = 0.04$, at which the performance of our proposed algorithm is close to the optimal one. These results comply with those presented in figure 4.12, 4.13, 4.14 and 4.15 for the previously chosen



(a) $n = 5$.



(b) $n = 7$.



(c) $n = 9$.

Figure 4.15 The impact of the winning increment factor (τ_2) on 2WSLS for different control plane topologies.

configuration $(2, 1, 0, -4)$ and demonstrate that the values of the deltas are not very important unless $\delta_1 > \delta_2 > \delta_3 > \delta_4$. This, indeed, shows that the 2WSLS is flexible, robust and scalable.

4.10.2 Performance of 2WSLS

To verify the effectiveness of 2WSLS in solving the LBDSA problem, we compare its performance against OPT, MCBLB and SMCLBRT algorithms in terms of load balancing (LB) and migration cost (T_m) separately.

Figure 4.19 presents the comparison of 2WSLS, OPT, MCBLB and SMCLBRT in terms of load balancing (LB) for the three topologies $m = 20$, $m = 25$ and $m = 30$. Based on these results, we make the following observations: (1) It is clear that the performance of 2WSLS is close to that of OPT for

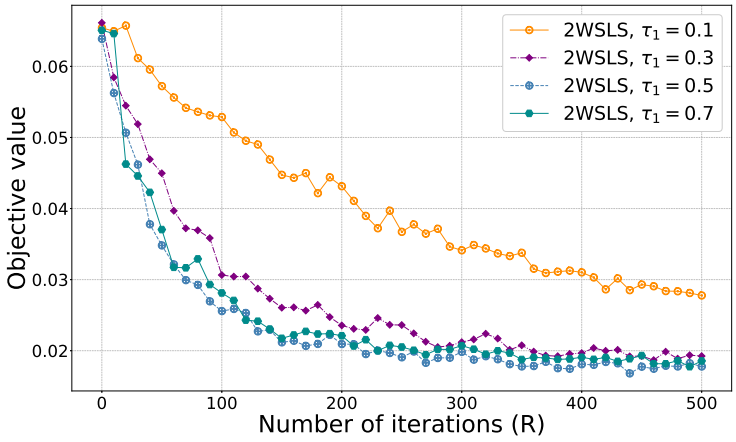
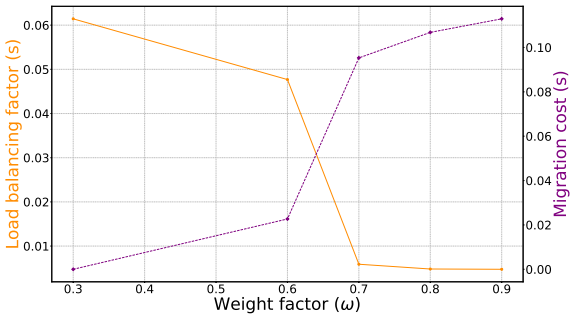


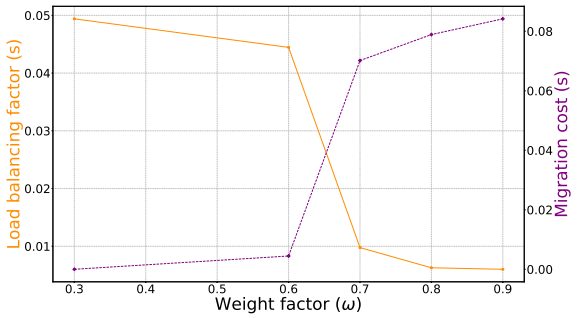
Figure 4.16 The impact of the number of iterations on 2WSLS.

the different values of m; (2) The performance gap between 2WSLS and OPT is almost the same for the different topologies, which illustrates its scalability; (3) 2WSLS outperforms MCBLB and SMCLBRT. Indeed, 2WSLS minimize the load balancing factor better than MCBLB since the latter compares the degree of load balancing achieved by a shift or a swap move against only the non-execution of this move. Also, in the definition of the cost function associated to a shift move or a swap move, MCBLB includes a factor which encourages migrating switches with low latency to final controllers. Therefore, this factor can poorly offload overloaded controllers, which has an impact on the load balancing. In SMCLBRT, the load balancing between the controllers is not really considered since determining whether a controller is overloaded or not depends only on a predefined threshold. Moreover, SMCLBRT simply selects the switch with the maximum load from the overloaded controller's domain then migrates it to an underloaded one.

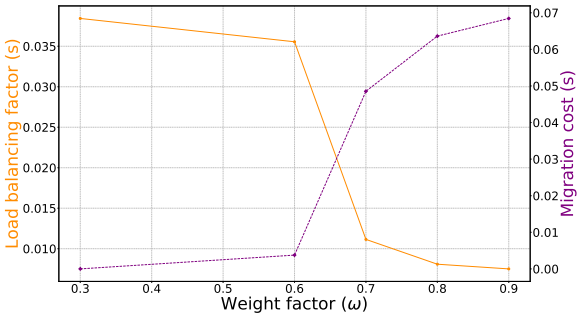
In figure 4.20, we compare the performance of 2WSLS, OPT, MCBLB and SMCLBRT in terms of migration cost (T_m) for the three topologies $m = 20$, $m = 25$ and $m = 30$. Note that the migration cost results, presented in figure 4.20, are obtained for the same realizations as the load balancing results in



(a) Configuration Δ^1.



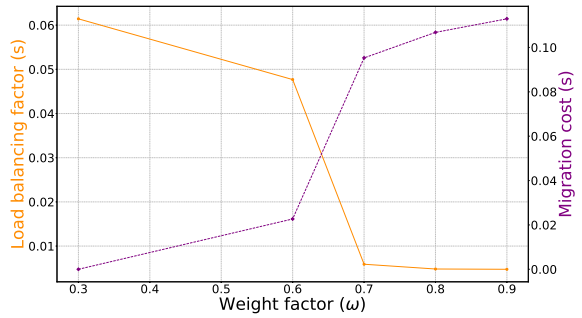
(b) Configuration Δ^2.



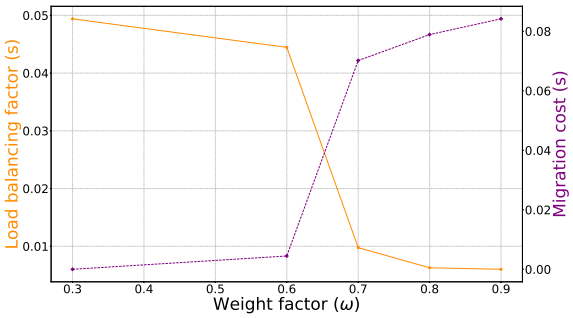
(c) Configuration Δ^3.

Figure 4.17 The impact of the winning increment factor (τ_1) on 2WSLS for different reward configurations.

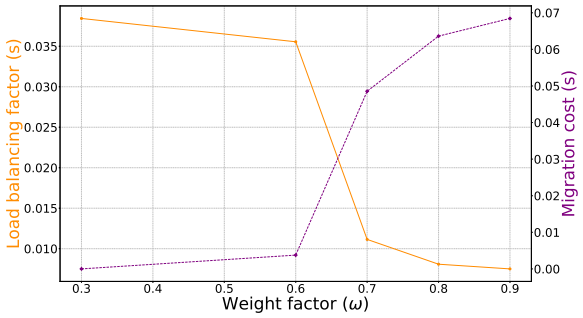
figure 4.19. We can observe that the migration cost of 2WSLS is the closest one to that of OPT for all m values. Furthermore, for the three topologies, we notice that the gap between the 2WSLS and OPT migration costs remains constant which illustrates its scalability. We can once again confirm that 2WSLS gives better performance compared to MCBLB and SMCLBRT. MCBLB is outperformed by 2WSLS because it uses swap moves when shift moves are not possible. In other words, swap moves generate more migration operations than shift moves which increases the migration cost. As for SMCLBRT, we can see that it has the worst performance among all algorithms in terms of migration cost. Indeed, SMCLBRT uses a threshold to determine the overloaded controllers and when this threshold is low which is the case in this paper, *i.e.*, to meet the requirements of delay sensitive applications, the



(a) Configuration Δ^1.



(b) Configuration Δ^2.



(c) Configuration Δ^3.

Figure 4.18 The impact of the winning increment factor (τ_2) on 2WSLS for different reward configurations.

number of overloaded controllers is higher. Hence, the number of migrated switches from overloaded controller increases which increases the migration cost.

4.11 Conclusion

In this paper, we tackled the load balancing problem in distributed SDN architecture using machine learning. Reactive load balancing mechanisms make the overloaded SDN controllers suffer from high response time until their offloading happens through migration operations. Furthermore, the time required to migrate a data plane component is not considered in most proposed algorithms. Thus, we propose to preemptively balance the load in the SDN control plane while minimizing the migration time such that the latency re-

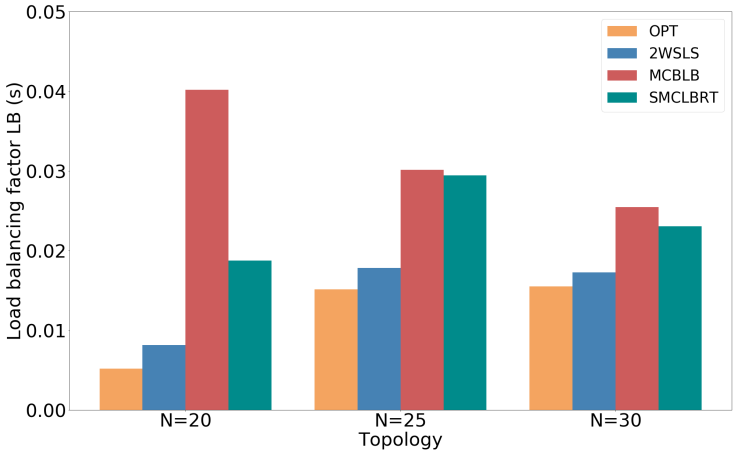


Figure 4.19 The performance of 2WSLS in terms of load balancing (LB) compared to OPT, MCBLB and SMCLBRT.

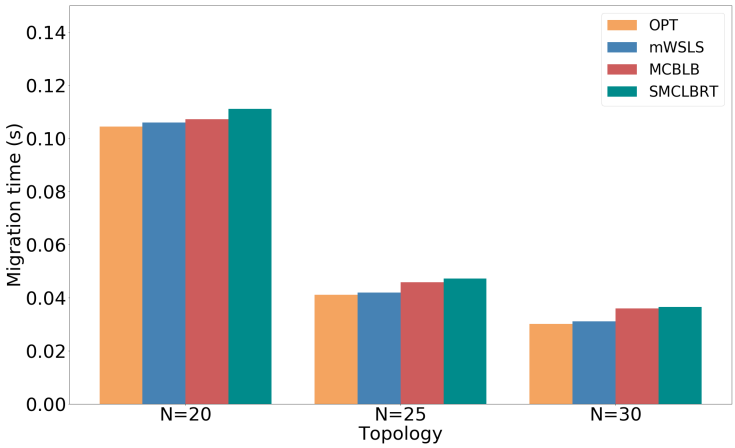


Figure 4.20 The performance of 2WSLS in terms of migration cost (T_m) compared to OPT, MCBLB and SMCLBRT.

quirements of delay sensitive applications are satisfied. For this purpose, first, we built Auto Regressive Integrated Moving Average (ARIMA) and Long Short-Term Memory (LSTM) models to predict the load of SDN controllers and compared their performance in terms of accuracy in short-term and long-term predictions. Comparison results showed that ARIMA lightly outperforms LSTM in short-term predictions while LSTM widely surpasses ARIMA in long-term predictions. Long-term predictions allow to preemptively figure out if the load in the control plane will be unbalanced and, consequently, schedule migration operations in advance. Second, to select which data plane component should be migrated and where the migration

should happen, we formulate the load balancing for delay sensitive applications (LBDSA) problem as a non-linear binary program and prove its NP-completeness. To solve this problem, we proposed a reinforcement learning algorithm, called 2WSLS, which is inspired by the well-known learning rule of win-stay-lose-shift. Finally, we benchmark 2WSLS against two algorithms from the literature. Simulation results demonstrated that 2WSLS has close to optimal performance and outperforms the benchmark algorithms in terms of load balancing and migration cost.

Chapitre 5: Avant-propos

Auteurs et affiliation:

Abderrahime Filali: étudiant au doctorat, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Zoubeir Mlika: Chercheur post-doctoral, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Soumaya Cherkaoui: Professeure, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Abdellatif Kobbane: Professeur, Université Mohammed-V, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Rabat-Maroc.

Date de soumission: avril 2021.

État de l'acceptation: en cours sous révision.

Revue: IEEE Transactions on Network Science and Engineering (TNSE).

Titre français: Découpage dynamique du réseau d'accès radio basé sur SDN avec apprentissage par renforcement profond pour les services URLLC et eMBB.

Résumé français:

Le découpage du réseau d'accès radio (en anglais radio access network, RAN) est une technologie clé qui permet au réseau 5G de répondre aux exigences hétérogènes des services génériques, à savoir la communication ultra-fiable à faible latence (en anglais ultra-reliable low-latency communication, URLLC) et le haut débit mobile amélioré (en anglais enhanced mobile broadband,

eMBB). Le réseau défini par logiciel (en anglais software-defined networking, SDN) peut contribuer à assurer la programmabilité et la flexibilité du découpage du RAN sur une large échelle de temps et à contrôler l'allocation des ressources radio des différentes tranches du RAN sur une échelle de temps plus courte. Dans cet article, nous proposons un mécanisme de découpage du RAN à deux échelles de temps pour optimiser les performances des services URLLC et eMBB. Dans la première grande échelle de temps, les ressources radio sont allouées aux gNodeBs en fonction des exigences des services eMBB et URLLC. Dans la deuxième échelle de temps courte, chaque gNodeB alloue ses ressources disponibles à ses utilisateurs finaux et demande, si nécessaire, des ressources supplémentaires aux gNodeB adjacentes. Nous formulons ce problème d'allocation de ressources à deux échelles de temps sous la forme d'un programme binaire non linéaire et prouvons qu'il est NP-difficile. Ensuite, pour chaque échelle de temps, nous modélisons le problème comme un processus de décision de Markov (en anglais Markov decision process, MDP), où le problème à grande échelle de temps est modélisé comme un MDP à un seul agent tandis que le problème à courte échelle de temps est modélisé comme un MDP multi-agents. Pour résoudre chaque MDP, une approche basée sur l'apprentissage par renforcement est proposée. Plus précisément, nous utilisons l'algorithme d'exploration et d'exploitation à poids exponentiel (en anglais exponential-weight algorithm for exploration and exploitation, EXP3) pour résoudre le MDP à agent unique à grande échelle de temps et l'algorithme d'apprentissage Q-profond (en anglais deep Q-learning, DQL) multi-agent pour résoudre le MDP multi-agent de l'allocation des ressources à courte échelle de temps. Des simulations extensives montrent que notre approche est efficace sous différentes configurations de paramètres de réseau et qu'elle surpasse les solutions de référence récentes.

Chapitre 5: Foreword

Authors and affiliation:

Abderrahime Filali: Ph.D. Student, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Zoubeir Mlika: Postdoctoral Research Fellow, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Soumaya Cherkaoui: Professor, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Abdellatif Kobbane: Professor, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS) Mohammed V University in Rabat, Morocco

Date of submission: april 2021.

Acceptance status: under revision.

Journal: IEEE Transactions on Network Science and Engineering (TNSE).

Title: Dynamic SDN-based Radio Access Network Slicing with Deep Reinforcement Learning for URLLC and eMBB Services

CHAPTER 5

Dynamic SDN-based Radio Access Network Slicing with DRLearning for URLLC and eMBB Services

5.1 Abstract

Radio access network (RAN) slicing is a key technology that enables 5G network to support heterogeneous requirements of generic services, namely ultra-reliable low-latency communication (URLLC) and enhanced mobile broadband (eMBB). Software defined networking (SDN) can help ensuring the programmability and the flexibility of RAN slicing in a large time-scale and monitoring the allocation of radio resources of different RAN slices in a shorter time-scale. In this paper, we propose a two time-scales RAN slicing mechanism to optimize the performance of URLLC and eMBB services. In the first large time-scale, radio resources are allocated to gNodeBs according to the requirements of the eMBB and URLLC services. In the second short time-scale, each gNodeB allocates its available resources to its end-users and requests, if needed, additional resources from adjacent gNodeBs. We formulate this two time-scales resource allocation problem as a non-linear binary program and prove its NP-hardness. Next, for each time-scale, we model the problem as a Markov decision process (MDP), where the large-time scale is modeled as a single agent MDP whereas the shorter time-scale is modeled as a multi-agent MDP. To solve each MDP, a reinforcement learning-based approach is proposed. Specifically, we leverage the exponential-weight algorithm for exploration and exploitation (EXP3) to solve the single agent MDP of the large time-scale MDP and the multi-agent deep Q-learning (DQL) algorithm

to solve the multi-agent MDP of the short time-scale resource allocation. Extensive simulations show that our approach is efficient under different network parameters configuration and it outperforms recent benchmark solutions.

5.2 Introduction

The heterogeneous services supported by the fifth-generation (5G) new radio (NR) can be classified mainly into enhanced mobile broadband (eMBB), ultra-reliable low-latency communication (URLLC) and massive machine-type communication (mMTC) services [107]. The eMBB services target the applications that require a high data rate such as high definition (HD) video or large-scale video streaming. The URLLC services accommodate low-latency and high reliability applications such as autonomous driving or robotic surgery. Finally, the mMTC services provide connectivity to a large number of devices, e.g., massive access in Internet of things (IoT) networks [108, 109], that are characterized by small data and sporadic traffic. To support these three 5G services while respecting their heterogeneous and different requirements over a common wireless network infrastructure, radio access network (RAN) slicing [58] is introduced as a key enabling technology in the new generation of cellular networks. Network slicing (NS) provides the ability to build several independent logical networks, called network slices, each adapted to the requirements of a specific service [85]. Therefore, each RAN slice can be tailored and dedicated to support a specific service with distinctive characteristics and requirements. The network operator can leverage the network programmability provided by software defined networking (SDN) to dynamically manage the provisioning of radio resources for the RAN slices [16, 59, 110].

Unlike the cloud RAN (C-RAN) architecture, which presents major challenges in its deployment with a multi-access edge computing (MEC) environment in terms of maintaining service availability and leveraging MEC resource [111],

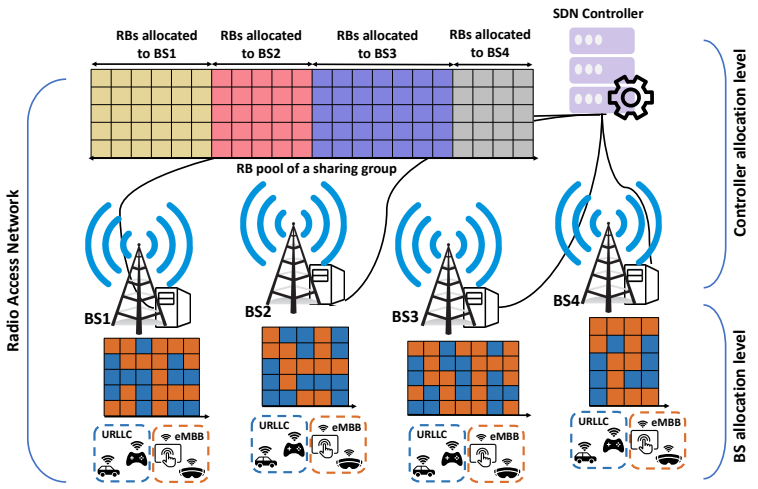


Figure 5.1 Resource block allocation procedure

SDN can be recognized as an important 5G RAN enabler in a fog RAN (F-RAN) architecture. Specifically, co-deploying SDN with F-RAN increases the ability of exploiting radio resources using its global view of the network [112]. The radio resource considered in this work is the resource block (RB), which is the minimum resource element that can be allocated to an end-user and it is identified by the frequency band and time slot pair [113]. Each gNodeB should have enough RBs from the shared radio RBs pool to meet the requirements of its end-users [114]. In a RAN slicing scenario, SDN can be used to allocate the appropriate RBs for each RAN slice in each gNodeB, depending on the radio resource availability and the services required by the end-users of the corresponding gNodeB. A notable challenge here is how to achieve an optimal RBs allocation for each gNodeB to satisfy the quality of service (QoS) requirements of its end-users in different slices.

To solve this problem, a large variety of algorithms and schemes have been proposed in recent years [61, 62, 63, 64, 65, 66, 67, 115, 116] that mainly present either centralized resource allocation solutions or multi-level resource allocation solutions. However, we identify important gaps in these related-works. In the centralized solution approaches, the radio resource allocation

decision relies only on a central entity (e.g., an SDN controller), which increases the signaling overhead in the network caused by frequent communications between the gNodeBs and the central entity, especially when the latter has to perform the resource allocation of the different gNodeBs. Therefore, the operation of allocating radio resources to end-users is expected to be performed in a large time-scale. On the other hand, in the multi-level resource allocation solution approaches, the SDN controller allocates, in the upper level, radio resources to RAN slices or gNodeBs. The latter are responsible, in the lower level and based on the pre-allocated resources by the SDN controller, for allocating radio resources to end-users. Although multi-level resource allocation solutions can efficiently allocate RBs to end-users, when the pre-allocated resources by the SDN controller are not sufficient to handle the required services, the low-level allocation operation can fail since it must wait for the next resource reservation update or immediately solicit the SDN controller for more resources, which can significantly reduce the QoS that the network operator is expected to provide.

In this paper, we fill these two gaps by proposing a two-level RB allocation mechanism. In the first level and in a large time-scale, the SDN controller allocates to each gNodeB a number of RBs from the common radio RBs pool, according to the gNodeB requirements. In the second level, each gNodeB schedules the pre-allocated RBs to its associated end-users in a short time-scale to satisfy their QoS requirements in terms of data rates and delay. This mechanism avoids frequent communications between gNodeBs and the SDN controller caused by abrupt and potentially unexpected fluctuations in wireless network traffic. To further reduce communications between gNodeBs and the SDN controller, a gNodeB can request additional RBs from other adjacent gNodeBs when its pre-allocated RBs are not sufficient, which allows to immediately respond to the requirements of the end-users.

In this work, two types of slices are considered, each one is dedicated to a single 5G service, namely the eMBB service and the URLLC service. Then, we formulate the two-level RB allocation problem as an optimization problem where the objective is to maximize the total achievable data rate of eMBB and URLLC end-users. This objective has to be achieved subject to the ultra-low latency requirements of the URLLC services as well as to the minimum data rate requirements of the eMBB services. The proposed mechanism to solve this optimization problem will: (1) reduce the signaling overhead between the gNodeBs and the SDN controller, and (2) quickly provide the required RBs for different slices to meet the services requested by the corresponding end-users.

The novelty of this work lies in two main parts. In the first part, we define the two-level radio RB allocation problem using integer programming and we analyze its NP-hardness. In the second part, we propose a single-agent multi-agent reinforcement learning (SAMA-RL) framework to solve the two-level radio RB allocation problem. Therefore, the proposed RL-based RAN slicing framework is dynamic since RL algorithms can adapt their RB allocation policies permanently according to different factors that mainly include the density of the end-users in the network, the requirements of the eMBB and the URLLC services, and the transmission conditions of the wireless channel. In addition, the proposed RAN slicing approach is performed in two-time scales to allocate resources for two levels including the SDN controller level and the gNodeBs level. Precisely, the proposed two-time scales RAN slicing approach is supervised by an SDN controller to manage the resource block allocation policies dynamically, which means that the programmability provided by the SDN controller enables an automatic resource management. The main contributions of this paper are summarized as follows:

- We use mathematical programming techniques to model the global RBs allocation for eMBB and URLLC end-users as a non-linear binary program and study its NP-hardness.
- Due to the NP-completeness result, obtaining an optimal solution to the RB allocation problem is computationally expensive. Thus, we model each level of the RB allocation problem as a Markov decision process (MDP).
- To fairly partition RBs between gNodeBs according to their requirements, we design a single agent RL-based algorithm to partition the RBs between gNodeBs in the first allocation level.
- In the gNodeBs level, based on the pre-allocated RBs by the SDN controller to gNodeBs, we propose a multi-agent deep Q-learning (DQL) approach to allocate RBs to eMBB and URLLC end-users and perform RBs sharing between gNodeBs.
- We evaluate the performance of our proposed mechanism against a benchmark algorithm and perform extensive simulations to show the superiority of the proposed framework.

The rest of the paper is organized as follows. Section II discusses analyzes the related works. Section III presents the system model. Section IV formulates the RB allocation problem as a mathematical program and studies its NP-hardness. Section V presents the proposed learning solutions. Section VI highlights the performances of the proposed mechanism and discusses the obtained results. Finally, section VII concludes the paper.

5.3 Related Work

In [61], the authors propose a framework for RAN slicing by using two machine-learning approaches: (i) LSTM is used to predict the resources that should be allocated to a slice in a large time-scale (ii) a multi-agent RL-Algorithm, known as A3C, is exploited for on-line resource scheduling of

RAN slices. However, the proposed framework considers only eMBB end-users and the slicing for individual end-users is not considered. In [115], the controller reserves resources for URLLC and eMBB slices at each base station considering the minimum resource requirement of each slice. Then, to meet the required QoS and increase the resource utilization utility of slices, a deep RL algorithm is executed for each slice on each base station to dynamically update the allocated RBs based on the reserved resources. Similarly, the authors of [62] improve the user QoS satisfaction and resource utilization utility by adjusting the resource provided to individual RAN slices. They leverage DQL approach with the dueling DQN algorithm to solve the slice resource provisioning problem. However, if the resources reserved for a given slice are not sufficient to handle the required services, the slice must wait for the next resources reservation update since the slice cannot occupy more resources than those assigned by the controller. The authors of [63] present a dynamic radio resource slicing scheme for a two-tier heterogeneous wireless network to determine the optimal bandwidth slicing ratios for slices. An alternative concave search algorithm is designed to solve the maximum network utility optimization problem. Although the proposed scheme satisfies the QoS requirement of machine-type and data slices, it cannot support URLLC slices which need much lower latency. Also, resources for each slice are expected to be updated only in a large time-scale. In [64], a two-level SDN-based radio resource allocation framework is designed to improve RAN slicing over different time scales. In a large time-scale, the SDN controller allocates RBs to gNodeBs, while in a small time-scale the pre-allocated RBs are scheduled by each gNodeB to eMBB and URLLC users. Moreover, each gNodeB can borrow RBs from other gNodeBs when the pre-allocated RBs are insufficient. However, the interactions between the base stations to share the RBs are not described. In meeting the eMBB, URLLC and mMTC slice user requirements, the authors of [65] formulate the user-association and resource alloca-

tion problem as a maximum utility optimization problem. The optimization problem is decomposed in two sub-problems using the hierarchical decomposition method. The base station-slice user association sub-problem is solved by many-to-one matching game and a genetic algorithm is adopted to solve the dynamic resource allocation sub-problem. The authors of [66] present a dynamic framework to allocate radio resources for two types of vehicular network services, i.e., delay-sensitive service and delay-tolerant service. The radio resource allocation problem is jointly formulated with that of computing resource allocation as an optimization problem. To solve this problem, they use a two-layer constrained RL algorithm. Based on the proposed RL algorithm, the SDN controller is responsible for allocating resources to slices at all the base stations. In order to provide their mobile users the access to the virtual computation and communication slices, multiple service providers compete in [67] to orchestrate channel access opportunities. The authors model such resource allocation problem as a non-cooperative stochastic game and leverage a deep learning approach based on double deep Q-network to approximate the optimal allocation strategy. In [116], a two-step RAN slicing framework is proposed to improve bandwidth utilization. In the first step, the framework selects a set of users whose QoS can be satisfied simultaneously. In the second step, each admissible user is associated with a slice via a specific base station and a fraction of the base station bandwidth is allocated to it.

5.4 System Model

We consider an SDN-enabled 5G RAN architecture composed of a finite set of gNodeBs $\mathcal{B} = \{1, 2, \ldots, B\}$. Two types of slices are considered, namely the URLLC slice and the eMBB slice, which are denoted by s_u and s_e, respectively. Spectrum resources are represented as a shared RBs pool denoted by $\mathcal{K} = \{1, 2, \ldots, K\}$, where each RB represents the minimum scheduling unit. The end-users $\mathcal{U} = \{1, 2, \ldots, U\}$ are randomly located across the net-

work area and where they are URLLC end-users and eMBB end-users. Each end-user $u \in \mathcal{U}$ is served by one gNodeB and belong to one slice, *i.e.,* s_u or s_e. Each gNodeB $b \in \mathcal{B}$ has a set of associated end-users denoted by $\mathcal{U}_b$. We consider the orthogonal frequency division multiple access (OFDMA) downlink (DL) scenario, where the RBs are organized as a resource grid [117]. With OFDMA, the transmissions to end-users are scheduled in an orthogonal manner to reduce interference.

The RB allocation procedure is performed in two levels. In the first level, the SDN controller, since it has a global view of the network, allocates the RBs to the gNodeBs using the available RBs pool, in a large time-scale. We call this RB allocation level the SDN allocation level. The set of RBs assigned by the SDN controller to a gNodeB $b \in \mathcal{B}$ is donated by $\mathcal{K}_b \subseteq \mathcal{K}$. In the second level, the RB allocation procedure is performed by the gNodeBs to allocate the necessary RBs to each end-user. We call this RB allocation level the gNodeB allocation level. Each gNodeB $b \in \mathcal{B}$ allocates to each of its associated end-users, during a short time-scale, a number of RBs, from the pre-allocated RBs $\mathcal{K}_b \subseteq \mathcal{K}$, to satisfy the various QoS requirements in terms of data rate and latency of each end-user. To ensure the orthogonality of DL transmissions among end-users which are served by the same gNodeB $b \in \mathcal{B}$, each RB $k \in \mathcal{K}_b$ is exclusively assigned to one end-user $u_b \in \mathcal{U}_b$. Also, if $b \in \mathcal{B}$ borrows $k' \in \mathcal{K} \setminus \mathcal{K}_b$ to allocate it to one of its end-users, then this RB k' should be unallocated. Thus, we define the assignment of an RB $k \in \mathcal{K}$ to an associated end-user $u_b \in \mathcal{U}_b$ with $b \in \mathcal{B}$ as a binary variable $x^k_{u_b}$, where:

$$x^k_{u_b} = \begin{cases} 1, & \text{if } k \in \mathcal{K} \text{ is assigned to } u_b \in \mathcal{U}_b, b \in \mathcal{B}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

$$\sum_{u_b \in \mathcal{U}_b} x^k_{u_b} \leq 1, \forall k \in \mathcal{K}, \forall b \in \mathcal{B}. \tag{5.2}$$

Equation (5.2) states that an RB must be allocated to only one end-user at a time, which meets the OFDMA constraints.

In 5G NR, the scalable OFDM technology is a key innovation. The 3GPP 5G NR Release 15 specifications [118] state that the waveform is scalable in the sense that the subcarrier spacing of OFDM can be adapted to channel width. The choice of the spacing parameter depends on several factors, including the requirements of 5G services, *e.g.,* low latency for URLLC services and high data rate for eMBB services. Indeed, eMBB and URLLC services can be supported simultaneously on the same carrier by multiplexing two different numerologies, larger subcarrier spacing for URLLC services and lower subcarrier spacing for eMBB services. In our approach, the RB allocation operation is performed considering a given time-frequency resource grid model. In other words, for each numerology strategy, *i.e.,* a specific subcarrier spacing, such as 15 kHz, 30 kHz and 60 kHz, the appropriate trained model should be used to perform the RB allocation operation. Since the gNodeB is responsible for deciding which numerology strategy should be applied based on the channel state information, it can select the appropriate trained model for the selected numerology to allocate the RBs. Therefore, our approach can be applied with multiple 5G NR OFDM numerologies to allocate the RBs, provided that the appropriate trained model for each possible numerology strategy is available.

In this article, we assume that the gNodeBs have a (near) perfect knowledge of the channel state information [119, 120]. The availability of (near) perfect channel state information at the gNodeB in a real 5G network can be justified when fading varies slowly over time and the mobility of the end-users is low since the wireless channel does not change rapidly, which is similar to our case. The channel state information can be accurately estimated in 5G networks using, for example, deep learning algorithms [121].

The achievable data rate of the u_b-th end-user associated with the b-th gNodeB over the k-th RB and belonging to a slice $s \in \{s_u, s_e\}$ is defined as follows:

$$r^s_{u_b,k} = W \log_2 \left(1 + \frac{P^s_{u_b} G_{u_b,k}}{\sigma^2}\right), \tag{5.3}$$

where W denotes the bandwidth of an RB, $P^s_{u_b}$ is the transmission power of $b \in \mathcal{B}$ to end-user $u_b \in \mathcal{U}_b$ in slice $s \in \{s_u, s_e\}$, $G_{u_b,k}$ is the DL channel gain between $b \in \mathcal{B}$ and its associated end-user $u_b \in \mathcal{U}_b$, and σ^2 is the power of the additive white Gaussian noise (AWGN). We assume that the bandwidth and the downlink transmission power are the same for all RBs.

During the second level of resource allocation, if the SDN controller does not allocate sufficient RBs to a gNodeB, the latter can request additional RBs from other gNodeBs. In other words, we assume that a gNodeB can request additional RBs from other gNodeBs only if all its pre-allocated RBs are already assigned to its end-users. Mathematically, gNodeB b can request additional RBs from other gNodeBs and allocate them to its associated end-users *if and only if* $\sum_{u_b \in \mathcal{U}_b} \sum_{k \in \mathcal{K}_b} x^k_{u_b} \geq |\mathcal{K}_b|$. This constraint can be defined as follows:

$$x^{k'}_{u_b} |\mathcal{K}_b| \leq \sum_{u_b \in \mathcal{U}_b} \sum_{k \in \mathcal{K}_b} x^k_{u_b}, \tag{5.4}$$

where $k' \in \mathcal{K}_{b'}$ is a borrowed RB by the gNodeB $b \in \mathcal{B}$ from gNodeB $b' \neq b \in \mathcal{B}$ and obviously $x^{k'}_{u_b} = 1$.

The total achievable data rate of end-user $u_b \in \mathcal{U}_b$ belonging to slice $s \in \{s_u, s_e\}$ and associated to gNodeB $b \in \mathcal{B}$ is defined as follows:

$$r^s_{u_b} = \sum_{k \in \mathcal{K}_b} x^k_{u_b} r^s_{u_b,k} + \sum_{\substack{b' \in \mathcal{B} \\ b' \neq b}} \sum_{k' \in \mathcal{K}_{b'}} x^{k'}_{u_b} r^s_{u_b,k'} \tag{5.5}$$

To calculate the delay for URLLC and eMBB traffic, we consider the following assumptions: (i) the arrival process of each gNodeB's packets follows a Poisson distribution, and (ii) the inter-arrival times of the packets are independent and follow an exponential distribution [122]. Accordingly, the queuing traffic model can be considered as an M/M/1 queuing system. In addition, the packet lengths for different slices are different but they are similar in a slice $s \in \{s_u, s_e\}$. By applying Little's law, we calculate the average delay $d^s_{u_b}$ experienced by an end-user packet u_b belonging to slice $s \in \{s_u, s_e\}$ and associated with $b \in \mathcal{B}$ as follows:

$$d^s_{u_b} = \frac{1}{r^s_{u_b} - \lambda^s_{u_b}}, \tag{5.6}$$

where $\lambda^s_{u_b}$ is the packets arriving rate of $u_b \in \mathcal{U}_b$ and belonging to slice $s \in \{s_u, s_e\}$.

We choose the M/M/1 queuing system since it is widely used to characterize wireless communication systems, particularly in RAN slicing approaches [123, 124, 66]. However, in some practical scenarios, the traffic becomes bursty and, therefore, the M/M/1 assumption (i.e., the packet arrival process of each gNodeB follows a Poisson distribution) may become too optimistic. In such scenarios, the gNodeB cannot be considered as an M/M/1 system and the queue cannot be modeled as a continuous-time Markov process. In this case, when the traffic is bursty, the queuing system can be modeled as a discrete-time Markov process [125, 126]. As shown in [127], many continuous-time Markov processes can be transformed into discrete-time Markov processes by observing only the state transitions. Therefore, since the M/M/1 queuing assumption can be seen as a continuous-time Markov process, the analyses and results obtained in this work are valid for most scenarios where the Poisson distribution cannot be applied. Note that, under different queuing assump-

tions, the mathematical analysis may be different and more complicated. Thus, for simplicity, we only assume the M/M/1 case.

5.5 Problem Formulation and NP-Hardness

5.5.1 Problem Formulation

The main question of RB allocation problem in RAN is how to derive a two-level optimal allocation of RBs to URLLC and eMBB end-users that meet their QoS requirements in terms of data rate and delay. For this purpose, the global RB allocation optimization problem is formulated as follows:

$$\underset{x}{\text{maximize}} \quad \sum_{b\in\mathcal{B}} \sum_{s\in\{s_u,s_e\}} \sum_{u_b\in\mathcal{U}_b} r^{s}_{u_b} \tag{5.7a}$$

subject to

$$\sum_{k\in\mathcal{K}} x^{k}_{u_b} \leq K_{\max}, \forall u_b \in \mathcal{U}_b, \forall b \in \mathcal{B}, \tag{5.7b}$$

$$\sum_{u_b\in\mathcal{U}_b} x^{k}_{u_b} \leq 1, \forall k \in \mathcal{K}, \forall b \in \mathcal{B}, \tag{5.7c}$$

$$x^{k'}_{u_b} \times |\mathcal{K}_b| \leq \sum_{u_b\in\mathcal{U}_b} \sum_{k\in\mathcal{K}_b} x^{k}_{u_b}, \forall k' \in \mathcal{K} \setminus \mathcal{K}_b, \forall u_b \in \mathcal{U}_b, \forall b \in \mathcal{B}, \tag{5.7d}$$

$$r^{s_e}_{u_b} \geq \mathcal{R}_{min}, \forall u_b \in \mathcal{U}_b, \forall b \in \mathcal{B}, \tag{5.7e}$$

$$d^{s_u}_{u_b} \leq \mathcal{D}_{max}, \forall u_b \in \mathcal{U}_b, \forall b \in \mathcal{B}, \tag{5.7f}$$

$$x^{k}_{u_b} \in \{0,1\}, \forall k \in \mathcal{K}, \forall u_b \in \mathcal{U}_b, \forall b \in \mathcal{B}. \tag{5.7g}$$

The objective function in (5.7a) maximizes the total sum data rates of the URLLC and eMBB end-users. Constraints (5.7b) guarantee a fair RBs allocation by forcing the number of RBs allocated to each end-user u_b to not exceed a maximum number $K_{\max}$. Constraints (5.7c) respect the OFDMA constraints by ensuring that each RB is allocated to only one end-user at a time. Constraints (5.7d) guarantee that a gNodeB b can request additonal RBs from other gNodeBs if and only if all of its pre-allocated RBs are used

by its end-users. Constraints (5.7e) state that the data rate of the eMBB end-users must be greater than a minimum required threshold $\mathcal{R}_{min}$. Constraints (5.7f) ensure that the delay of URLLC end-users cannot exceed a maximum required threshold $\mathcal{D}_{max}$. Finally, constraints (5.7g)) list the optimization variables.

Note that the dynamic resource allocation problem considers limited resources. In fact, the formulated RB allocation problem consists of maximizing the total achievable data rate of eMBB and URLLC end-users subject to quality-of-service constraints and limited resources constraints represented by the set of constraints given in (5.7b). These constraints guarantee that every end-user cannot be allocated more than a maximum number of resources. They limit the number of resources that each end-user can have and thus can allocate the remaining resources more efficiently and in a fair manner between end-users. The limitation of resources in the considered problem is also stated by the borrowing concept. Once a gNodeB is out of resources because all of its SDN-allocated resources are used, it can borrow other gNodeBs-resources.

Due mainly to the binary nature of the optimization variables and the non-linearity of the delay experienced by an end-user defined in equation (5.6), the RB allocation problem is a non-linear binary programming problem. It is thus very challenging to solve (5.7) in general. In the sequel, we study its NP-hardness.

5.5.2 NP-Hardness

Here, we denote the problem (5.7) by P. To prove that P is NP-hard, we reduce the 0-1 knapsack problem [128], which is NP-hard, to an instance of P.

Definition 2 (0-1 knapsack problem)**.** *A 0-1 knapsack problem is defined as follows: given a set $\mathcal{N}$ of n items, each one with its profit p_i and weight w_i, and a knapsack of capacity C. Each item can be put into the knapsack or not*

(1 or 0). The objective of this problem is to find a subset $\mathcal{N}' \subseteq \mathcal{N}$ such that the total value of items $\sum_{n_i \in \mathcal{N}'} p_i$ is maximized and the total weight of the selected items is less than or equal to the knapsack capacity, i.e., $\sum_{n_i \in \mathcal{N}'} w_i \leq C$.

Theorem 2. *P is NP-hard.*

Proof. We prove the theorem by considering a restricted version of P. This shows that the problem is NP-hard in the general case as well. We consider the following restricted version of P:

- there is only one gNodeB denoted by b with its associated end-users set $\mathcal{U}_b$.
- there is only the eMBB slice.
- the RBs pre-allocated by the SDN controller $\mathcal{K}_b$ for b are known, i.e., the first level of the RB allocation procedure is already performed by the SDN controller.

In this case, P becomes equivalent to the following:

$$\underset{x}{\text{maximize}} \quad \sum_{u_b \in \mathcal{U}_b} r_{u_b}^{s_e} \tag{5.8a}$$

subject to

$$\sum_{k \in \mathcal{K}_b} x_{u_b}^k \leq |\mathcal{K}_b|, \forall u_b \in \mathcal{U}_b, \tag{5.8b}$$

$$\sum_{u_b \in \mathcal{U}_b} x_{u_b}^k \leq 1, \forall k \in \mathcal{K}_b, \tag{5.8c}$$

$$r_{u_b}^{s_e} \geq \mathcal{R}_{min}, \forall u_b \in \mathcal{U}_b, , \tag{5.8d}$$

$$x_{u_b}^k \in \{0, 1\}, \forall k \in \mathcal{K}_b, \forall u_b \in \mathcal{U}_b, . \tag{5.8e}$$

The mathematical statement of the problem in equation (5.8) is equivalent to finding an allocation of the RBs set $\mathcal{K}_b$ to the eMBB end-user set $\mathcal{U}_b$ such

that: (i) the sum data rates of the end-users is maximized, (ii) the maximum number of pre-allocated RBs to gNodeB b does not exceed $K_{max} = |\mathcal{K}_b|$ (5.8b), (iii) each RB is allocated to only one end-user at a time (5.8c), and (iv) the data rate of the eMBB end-users must be greater than a minimum required threshold $\mathcal{R}_{min}$ (5.8d).

To reduce the 0-1 knapsack problem to (5.8), we let (i) the number of items N be the number of eMBB end-users, (ii) the profit p_i for item i be the achievable data rate $r_{u_b}^{s_e}$ for eMBB end-user u_b, (iii) the weight w_i for item i be the allocated RBs for eMBB end-user u_b, and (iv) the knapsack capacity C be the number of the pre-allocated RBs to gNodeB b. Problem P is now clearly reduced to a knapsack problem. The knapsack capacity is respected and, thus, the number of allocated RBs to all eMBB end-users does not exceed $|\mathcal{K}_b|$.

Since the reduction is clearly done in a polynomial-time and the 0-1 knapsack problem is NP-hard, we conclude that the restricted problem formulated in equation (5.8) is also NP-hard, which proves the theorem. □

5.6 Single-Agent Multi-Agent Reinforcement Learning Based RAN Resource Slicing

The resource allocation problem is very challenging to solve optimally in a two-level procedure, *i.e.,* in the controller level and in the gNodeBs level. To overcome this challenge, we leverage machine learning techniques, particularly RL for performing RB allocation tasks due to its excellent capability to solve wireless network resource allocation problems in a computationally efficient manner [129]. We propose a single-agent multi-agent reinforcement learning (SAMA-RL), figure. 5.2, framework to solve the two-level radio RB allocation problem.

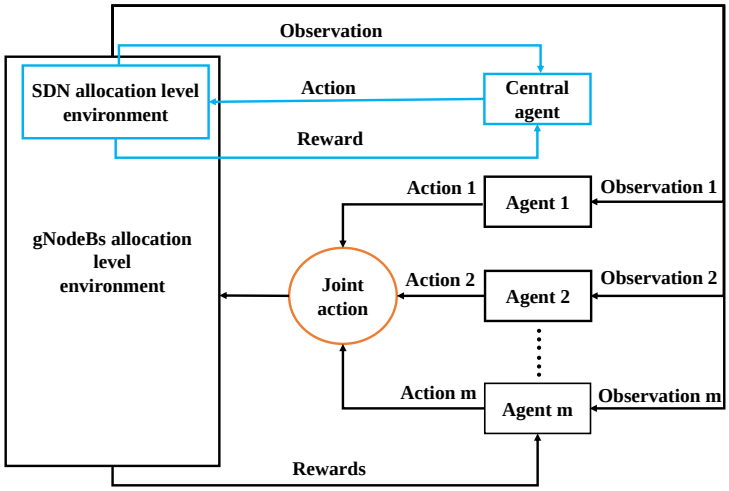


Figure 5.2 Single-agent multi-agent interaction with the MDP environment.

More precisely, in the SDN allocation level, we adapt the well-known exponential-weight algorithm for exploration and exploitation (EXP3) [130]. The SDN controller plays the role of an RL agent that performs the operation of allocating the RBs to the gNodeBs. In the gNodeB allocation level, a distributed multi-agent deep RL approach is proposed. Each gNodeB, acting as a DRL agent, schedules the pre-allocated RBs by the SDN controller to its associated end-users and, if necessary, cooperate with the other gNodeBs to dynamically share the unexploited RBs between them. To do so, we apply a DQL [131] model where each gNodeB acts as an independent agent. To avoid any confusion between the SDN controller agent and a gNodeB agent, the former will be called the central agent.

We opt for a classical RL algorithm, *i.e.,* EXP3, in the SDN allocation level because this resource allocation level does not suffer from the curse of dimensionality problem [132]. In fact, the state and action spaces are not large, which allows the central agent to learn in a reasonable time. EXP3 is a promising algorithm because it does not depend on any assumption related to the dynamicity of the SDN allocation system, which makes it relevant when the channel gains offered by RBs vary randomly. Moreover, EXP3

provides a good tradeoff between exploitation, the desire to make the best decision given current information, and exploration, the desire to try a new decision which may lead to better results. On the other hand, the gNodeB allocation level suffers from the curse of dimensionality due to the multi-agent scenario and presence of a large number of end-users, which leads the size of state space to grow significantly. Therefore, using a simple RL framework becomes computationally intractable. To overcome this challenge, we resort to a distributed multi-agent DRL approach in the gNodeB allocation level. Indeed, the agents are capable of discovering meaningful information through an appropriate deep neural network architecture. Therefore, the agents can learn close-optimal policies. In addition, DRL has been proven to be efficient in solving many resource allocation problems in wireless networks, such as energy scheduling [133] and orchestration of edge computing and caching resources [134].

Before explaining in details the proposed SAMA-RL algorithms, we first model, in what follows, each RB allocation problem of each level as a Markov decision process (MDP) and we define the main components of each MDP in details.

5.6.1 MDP formulation of the SDN allocation level

The MDP of the first resource allocation level is given by the triplet $(\mathcal{S}_c, \mathcal{A}_c, \mathcal{R}_c)$ where $\mathcal{S}_c$ represents the state space, $\mathcal{A}_c$ designates the action space and $\mathcal{R}_c$ is the reward function.

1) The state space

As discussed previously, the central agent is incorporated into the SDN controller which has a global view of the network. In fact, the central agent's state contains information about its previously allocated RBs to the gNodeBs and other global parameters of the network. More precisely, the state space of the central agent is composed of the triple $\mathcal{S}_c$ given as

follows:

$$\mathcal{S}_c = (\mathcal{B}, \mathcal{K}, \boldsymbol{\mathcal{K}_h}), \tag{5.9}$$

where $\boldsymbol{\mathcal{K}_h} = \{\mathcal{K}_b, \forall b \in \mathcal{B}\}$ represents the allocated RBs previously to each gNodeB. Also, the central agent' state includes the sets $\mathcal{B}$ and $\mathcal{K}$ that represent the sets of gNodeBs and RBs, respectively.

2) The action space

Since the central agent performs the RB allocation in a large time-scale, it has to decide which subset $\mathcal{K}_b \subseteq \mathcal{K}$ of RBs it should allocate to each gNodeB $b \in \mathcal{B}$. Note that the central agent has to make sure that an RB is assigned to only one gNodeB each time it performs the RB allocation. Therefore, the action space of the central agent, denoted by $\mathcal{A}_c$, is defined as follows:

$$\mathcal{A}_c = \{0, 1\}^{|\mathcal{K}| \times |\mathcal{B}|}, \tag{5.10}$$

where an action $\boldsymbol{a_c} \in A_c$ is given by the row vector $[a_1^1, \ldots, a_1^{|\mathcal{K}|}, a_2^1, \ldots, a_2^{|\mathcal{K}|}, \ldots, a_{|\mathcal{B}|}^1, \ldots, a_{|\mathcal{B}|}^{|\mathcal{K}|}]$. If an element a_b^k of vector $\boldsymbol{a_c}$ is equal to 1, it means that the central agent has decided to allocate RB k to gNodeB b. Also, to avoid assigning the same RB k to several gNodeBs, the central agent has to make sure that the constraints $a_b^k \neq a_{b'}^k, \forall k \in \mathcal{K}$ and $b \neq b' \in \mathcal{B}$ are satisfied.

3) The reward function

Once the central agent observes the environment through its current state, it chooses an action $\boldsymbol{a_c}$ from the set $\mathcal{A}_c$. After choosing an action from the action space $\mathcal{A}_c$, the central agent receives a reward $\mathcal{R}_c$. Since the objective is to maximize the total sum-rate, the objective of the central agent has to be related to the sum-rate of the network given in equation (5.7a). We define the reward as how the assigned RBs to gNodeBs affect the achieved data rate of the entire system. In other

words, the more the total data rate of the system is higher, the better is the action chosen by the central agent. Thus, the reward of the central agent is given by equation (5.7a), where:

$$\mathcal{R}_c = \sum_{b \in \mathcal{B}} \sum_{s \in \{s_u, s_e\}} \sum_{u_b \in \mathcal{U}_b} r^s_{u_b} \tag{5.11}$$

According to equation (5.11), the value of $\mathcal{R}_c$ depends on the RB allocation decisions in the gNodeB level. Indeed, the central agent should wait for the results of the second resource allocation level in order to efficiently explore its environment. Therefore, the RB allocation is performed in a joint manner between the two allocation levels.

5.6.2 MDP formulation of the gNodeB allocation level

The MDP of the second resource allocation level is given by the triplet $(\mathcal{S}_b, \mathcal{A}_b, \mathcal{R}_b)$ where $\mathcal{S}_b$ represents the state space of agent b, $\mathcal{A}_b$ designates the action space of agent b and $\mathcal{R}_b$ is the reward function of agent b.

1) The state space

As illustrated in figure 5.2, we propose a multi-agent DQL algorithm, with each gNodeB acts as an agent, to allocate the required RBs to eMBB and URLLC end-users and performs RBs sharing between gNodeBs, if necessary. Indeed, based on the chosen action by the central agent in the SDN allocation level, each agent observes its local state $\mathcal{S}_b$. In our model, we consider that the controller communicates to each agent the information about the RBs allocated to all other agents. Accordingly, each agent b's state $\mathcal{S}_b, \forall b \in \mathcal{B}$, is given by the following tuple:

$$\mathcal{S}_b = (\mathcal{G}_b, \mathcal{U}_b, \mathcal{K}_b, \mathcal{R}_{min}, \mathcal{D}_{max}), \tag{5.12}$$

where $\mathcal{U}_b$ and $\mathcal{K}_b$ represents the set of associated end-users with agent b and the set of pre-allocated RBs to agent b respectively. The term $\mathcal{G}_b =$

$(G_{u_b,k} : u \in U_b, k \in \mathcal{K})$ represents the DL channel gain between agent b and its associated end-users in each RB $k \in \mathcal{K}$. The channel gains $\mathcal{G}_b$ can be easily collected by agent b as follows: (i) agent b broadcasts pilot signals to all of its associated end-users. Then, each end-user estimates the channel state information and sends it back to the corresponding agent through a feedback channel. The estimation of $\mathcal{G}_b$ between each associated end-user over each RB in the system helps agent b to take good decisions, especially to borrow the needed RBs from the other agents. The state $\mathcal{S}_b$ also includes the minimum data rate threshold $\mathcal{R}_{min}$ and the maximum delay threshold $\mathcal{D}_{max}$ that corresponds to the requirements of eMBB and URLLC slices, respectively.

2) The action space

In the gNodeB allocation level, each agent takes actions according to its own allocation policy. In fact, the agent has to (i) assign the pre-allocated RBs to its associated end-users and (ii) request additional RBs from other agents when its pre-allocated RBs are not sufficient. Also, it is assumed that each agent is aware of the number of RBs present in the entire system. This information is acquired as follows: in each round of the SDN allocation level, the controller communicates with each agent the information on the RBs allocated to the other agents. Therefore, we define the space action of agent b, denoted by $\mathcal{A}_b$, as follows:

$$\mathcal{A}_b = \{0, 1\}^{|\mathcal{K}| \times |\mathcal{U}_b|}, \tag{5.13}$$

where an action $\boldsymbol{a_b} \in \mathcal{A}_b$ is given by the row vector $[a_1^1, \ldots, a_1^{|\mathcal{K}|}, a_2^1, \ldots, a_2^{|\mathcal{K}|}, \ldots, a_{|\mathcal{U}_b|}^1, \ldots, a_{|\mathcal{U}_b|}^{|\mathcal{K}|}]$. Note that a vector $\boldsymbol{a_b}$ is equivalent to an association matrix $[x_{u_b}^k]$ because each element $a_{u_b}^k$ of vector $\boldsymbol{a_b}$ corresponds to the assignment of an RB $k \in \mathcal{K}$ to an associated end-user $u_b \in \mathcal{U}_b$. In other words, if an element $a_{u_b}^k$ of vector $\boldsymbol{a_b}$ is equal to 1, it means that agent b has decided to allocate RB k to end-user u_b. Also, if all elements

$a_{u_b}^{k'}, \forall k' \in \mathcal{K} \setminus \mathcal{K}_b$, of vector $\boldsymbol{a_b}$ are equal to 0, it means that agent b does not request additional resources from the other gNodeBs. When constructing the action space $\mathcal{A}_b$ of agent b, the constraints (5.7b), (5.7c) and (5.7d) should be respected. By applying these constraints, we significantly reduce the action space $\mathcal{A}_b$. As a result, the exploration phase is significantly improved to discover better strategies, which considerably accelerates the learning process of agent b. Further, considering that agents should cooperate to dynamically share the unexploited RBs among themselves, each agent communicates its chosen action to the others [135, 136]. Therefore, each agent b forms a joint action $\boldsymbol{a} = (\boldsymbol{a}_b, \boldsymbol{a}_{-b})$ where $\boldsymbol{a}_{-b}$ denotes the actions chosen by the other agents.

3) The reward function

Multi-agent reinforcement learning methods seek to learn a policy that achieves the maximum expected total reward for all agents. Indeed, the learning process of all agents is driven by the reward function. In our model, the main objective is to maximize the total achievable data rate of the entire system to meet the QoS requirement of eMBB and URLLC end-users. Therefore, the reward function of agent b relates to its total sum-rate subject to the ultra-low latency requirements of the URLLC services as well as to the minimum data rate requirements of the eMBB services.

The reward of agent b depends on whether or not it has successfully allocated the needed RBs to its associated end-users. An RB allocation operation is considered to be feasible if the chosen action $\boldsymbol{a_b}$ satisfies the constraints (5.7b), (5.7c), (5.7d), (5.7e) and (5.7f), otherwise it is considered as an infeasible operation. The constraints (5.7b), (5.7c) and (5.7d) are already verified in the action space construction phase. However, as an allocation operation can require borrowing some RBs from other agents, it is necessary to verify if a borrowed RB is: (i)

exploited by its owner and (ii) also chosen by at least one agent other than its owner. Since each agent forms the joint action $\boldsymbol{a}$, (i) and (ii) can be easily verified by agent b. If at least one of them is correct, the constraint (5.7d) is not satisfied and thus the action chosen by agent b is not feasible. As a result, the individual reward of agent b, denoted by $\mathcal{R}_b$ is expressed as follows:

$$\mathcal{R}_b = \begin{cases} \sum\limits_{s\in\{s_u,s_e\}} \sum\limits_{u_b\in\mathcal{U}_b} r^s_{u_b}, & \text{if } \boldsymbol{a_b} \text{ is feasible,} \\ -1, & \text{if } \boldsymbol{a_b} \text{ is not feasible.} \end{cases} \tag{5.14}$$

When an action $\boldsymbol{a_b} \in \mathcal{A}_b$ is not feasible, it is penalized with a negative reward, $\mathcal{R}_b = -1$, to prevent the agent from choosing infeasible actions in the future.

5.6.3 Single-agent EXP3 algorithm

In order to solve the SDN allocation level problem, we adopt an online learning algorithm that is based on the multi-armed bandit (MAB) approach [130]. In MAB, a player needs to choose, at each round of the game, one arm from a finite set of arms, each characterized by an unknown reward, with the objective of maximizing his expected cumulative reward. The single-agent MDP is modeled as a MAB as follows. The central agent, i.e., the SDN controller, represents the player and the set of arms is given by its action space $\mathcal{A}_c$. We propose the EXP3 algorithm as a popular bandit strategy to solve the SDN allocation level problem [108]. The SDN controller runs the EXP3 algorithm where each action is assigned a weight to evaluate how good the action is for the SDN controller, *i.e.,* the higher the weight of an action, the better the action is. At the beginning of the algorithm, the weights of all actions are uniformly distributed. Then, the algorithm iterates several rounds. For each round, the SDN controller:

1. calculates the probability of choosing each action, which is proportional to its weights;

2. chooses an action according to the probability distribution calculated previously and receives a reward; and

3. uses the received reward to update the weights of each action by applying an exponential weighting scheme. The advantage of such a scheme is that it rapidly increases the probability of good actions, while rapidly reducing the probability of bad actions.

The pseudo-code of the EXP3 algorithm is presented in Algorithm 4.

In detail, the EXP3 algorithm takes as input an exploration parameter $\alpha \in [0, 1]$ that controls the desire to choose an action uniformly at random. It starts by assigning a weight ψ_i to each action $\boldsymbol{a_{c,i}}$, which is initialized to 1. Then, it iterates the rounds. For each round, the central agent calculates the probability π_i, given in equation (5.15), of choosing action $\boldsymbol{a_{c,i}}$. Based on π, it selects an action $\boldsymbol{a_{c,i}}$ and receives a reward $\mathcal{R}_{c,i}$. The obtained reward is scaled to the range $[0, 1]$ and it is denoted by $\bar{\mathcal{R}}_{c,i}$. Since $\mathcal{R}_{c,i}$ is the total achievable data rate of the system, it can be scaled using equation (5.16). After scaling the obtained reward, the central agent calculates an estimated reward $\hat{\mathcal{R}}_{c,i}$ using equation (5.16). The idea behind estimating the reward in such manner is to compensate for a potentially low probability of obtaining the observed reward. Finally, the weights are updated as $\psi_i = \psi_i \exp(\alpha \hat{\mathcal{R}}_{c,j}/|\mathcal{A}_c|)$.

$$\pi_i = (1 - \alpha)\frac{\psi_i}{\sum_{j=1}^{|\mathcal{A}_c|} \psi_j} + \frac{\alpha}{|\mathcal{A}_c|} \tag{5.15}$$

$$\bar{\mathcal{R}}_{c,i} = 1 - \frac{1}{(1 + \mathcal{R}_{c,i})} \tag{5.16}$$

$$\hat{\mathcal{R}}_{c,i} = \frac{\bar{\mathcal{R}}_{c,i}}{\pi_i} \tag{5.17}$$

Algorithm 4 EXP3-based SDN allocation level

Parameters: $\alpha \in [0, 1]$
Initialize $\psi_i = 1$ for all $i \in \{1, 2, \ldots, |\mathcal{A}_c|\}$

for *each round* **do**
  for $i = 1, 2, \ldots, |\mathcal{A}_c|$ **do**
    Calculate π_i using Eq. (5.15)
  end for
  Select an action $\boldsymbol{a_{c,i}}$ according to π_i
  Receive reward $\mathcal{R}_{c,i}$
  Calculate $\bar{\mathcal{R}}_{c,i}$ using Eq. (5.16)
  for $j = 1, 2, \ldots, |\mathcal{A}_c|$ **do**
    $\hat{\mathcal{R}}_{c,j} \leftarrow \bar{\mathcal{R}}_{c,j}/\pi_i \cdot \mathbb{1}_{j=i}$
    $\psi_i \leftarrow \psi_i \exp\left(\alpha \frac{\hat{\mathcal{R}}_{c,j}}{|\mathcal{A}_c|}\right)$
  end for
end for

The EXP3 algorithm is simple to implement and does not require enormous computational complexity since it simply updates the weights of choosing actions by increasing or decreasing their probabilities according to their performance. Note that the EXP3 algorithm is an online learning algorithm that enables the SDN controller to increase or decrease the weight of the actions according to the feedback received from the gNodeBs. Precisely, the SDN controller selects an action $\boldsymbol{a_{c,i}}$ to allocate the resource blocks to the gNodeBs. Then, it waits for the results of the second resource block allocation level, which is performed by the gNodeBs. Once the gNodeBs have assigned the resource blocks, chosen by the SDN controller, to their end-users, each gNodeB: 1) calculates its achieved data rate, and 2) communicates this information to the SDN controller. The total sum-data rate of all gNodeBs will be the reward received by the SDN controller after choosing action the $\boldsymbol{a_{c,i}}$, that will be used by the EXP3 algorithm to update the actions' weights.

5.6.4 Multi-agent deep Q-Learning algorithm

The DQL algorithm [131] extends the classical Q-learning RL [137] algorithm by approximating the Q-function using a deep neural network known as deep Q-network (DQN). In order to be used as an efficient non-linear approximator,

Algorithm 5 DQL Algorithm Training Phase

Input: Agents and environment
Output: Trained DDQNs
Start simulator: generate end-users and network parameters;
Initialize for each agent b: the main DQN, the target DQN and the replay memory $\mathcal{M}_b$;

for *each episode* **do**
  Reset and build the agents' environment;
  for *each step* **do**
    for *each agent b* **do**
      Get observation $\mathcal{S}_b$;
      Choose an action $\boldsymbol{a}_b$ using ϵ-greedy;
    end for
    for *each agent b* **do**
      Obtain the joint action $\boldsymbol{a}$ and receive reward $\mathcal{R}_b$;
      Obtain the next observation $\mathcal{S}'_b$;
      Store the experience exp_b in replay buffer $\mathcal{M}_b$;
      if *batch size* **then**
        Randomly sample a mini-batch from $\mathcal{M}_b$;
        Calculate target Q-value;
        Calculate loss between the main network and the target network;
        Update the parameters of the main network using gradient descent to minimze loss;
      end if
      if *target step* **then**
        Update the target network parameters;
      end if
    end for
  end for
end for

such a network must be trained to observe the state of the agent and learn weights in order to play actions that yield the highest rewards. Once the DQN is properly trained, it is exploited by the agent to take actions based on the observed state [138]. To solve the multi-agent MDP model, we propose a multi-agent DQL algorithm. This algorithm consists of two main phases: the training phase and the implementation phase. In the training phase, each agent trains a deep neural network (DNN) in an offline manner using a large amount of experiences (collected dataset). In the implementation phase, each agent chooses actions in an online manner using its trained model. We

describe, in the following, the training and implementation phases of the proposed multi-agent DQL approach.

1) The training phase of DQL

DQN approximates the Q-value function $\mathcal{Q}(s, a)$ through a neural network that performs a mapping between states and actions. In other words, this network returns, for any given state-action pair, the estimated Q-value $\mathcal{Q}(s, a; w)$, where w represents the parameters of the network (i.e., the weights). In order to improve the learning performance, DQN introduces the experience replay memory strategy that overcomes the learning stability issues. Indeed, this strategy stores the agent's experiences that include state transitions, actions and rewards, and then randomly samples from these experiences to perform Q-learning. As a result, the experience replay memory strategy reduces the correlation between the training samples, which prevents the optimal policy from being conducted to a local minimum. Although DQN can be effective, it still suffers from the problem of overestimating action Q-values. Double DQN (DDQN) [139] is proposed to mitigate this limitation and improve learning performance. The idea behind DDQN is to decouple action selection from evaluation. To achieve this, two neural networks are used, a main Q-network that selects an action and a target Q-network that calculates the Q-value of the selected action. In our DQL-based MARL algorithm, each agent b has a DDQN that takes the current state as input and outputs the Q-value function of all actions. The training phase of the proposed multi-agent DDQN is given in Algorithm 5.

In detail, the training phase requires as input the environment of each agent which includes gNodeBs, end-users, pre-allocated RBs, service requirements and channel state information. As output, it returns the trained DQN of each agent. The training starts by : (1) generating the network parameters, the end-users including their positions on the grid,

the service required (i.e., eMBB or URLLC) and their packet sizes, and (2) initializing the DQN of each agent. Next, DQL iterates the episodes. At the beginning of each episode, each agent's environment is built by updating the end-user locations and the channel coefficients (i.e., large scale fading). In each step, each agent b observes the current state $\mathcal{S}_b$ of its environment and takes an action $\boldsymbol{a_b}$ from its action space $\mathcal{A}_b$ by using the ϵ-greedy policy. With the ϵ-greedy policy, an agent selects an action randomly or using the Q-network. Precisely, this policy chooses the action with the highest Q-value with probability $1 - \epsilon$, where the exploration rate ϵ represents the probability that an agent will explore its environment rather than exploit it. As we go forward (i.e., after each step), each agent learns more about its environment and ϵ decays by some rate, so that exploring the environment becomes less probable. Once all agents choose their action following the ϵ-greedy policy, they communicate them to each other. Accordingly, each agent b forms its joint action, calculates its reward $\mathcal{R}_b$ using equation (5.14) and moves to a new state $\mathcal{S}'_b$. Next, the obtained tuple $(\mathcal{S}_b, \boldsymbol{a_b}, \mathcal{R}_b, \mathcal{S}'_b)$, called agent's b experience and denoted by exp_b, is stored in its replay memory $\mathcal{M}_b$. In practice, since the size of the replay memory is limited to a defined threshold $\mathcal{M}$, only the last $\mathcal{M}$ experiences can be stored. After storing enough experiences, each agent samples a random mini-batch from its replay memory. Note that the size of the replay memory should be large enough to reduce the correlation between the data that will be sampled from it. The obtained dataset is used by the agent to perform the training. With the objective of minimizing the loss function, given by equation (5.18), the main Q-network is used to approximate the Q-

value function while the target Q-network is used to outputs the target Q-value.

$$L_b(w_b) = \mathbb{E}[(y_b - \mathcal{Q}(\mathcal{S}_b, \boldsymbol{a_b}; w_b))^2], \tag{5.18}$$

where $\mathcal{Q}_b(\mathcal{S}_b, \boldsymbol{a_b}; w_b)$ is the approximated Q-value function given by the main Q-network of agent b with weight parameter w_b and y_b denotes the target Q-value and it is given as follows:

$$y_b = \mathcal{R}_b + \gamma \mathcal{Q}(\mathcal{S}_b, \underset{\boldsymbol{a_b}}{max} \{\mathcal{Q}(\mathcal{S}_b, \boldsymbol{a_b}; w_b)\}; w_b^-), \tag{5.19}$$

where $0 \leq \gamma \leq 1$ is called the discount factor, w_b^- is the weight parameter of the target Q-network. Note that the value of y_b is not necessarily the largest Q-value in the target Q-network, which allows to avoid chossing an overestimated action.

After calculating the loss function, each agent performs a gradient descent to update the parameters of the main Q-network. Finally, the parameters of the target Q-network are updated, at each fixed target step, by copying the parameters of the main Q-network.

Since the learning of the DDQNs is computationally heavy, the training phase of the DQL algorithm is performed in an offline manner. Accordingly, the training can be conducted using a large amount of dataset resulting from different network topologies and channel conditions.

2) The implementation phase of DQL

After the training phase, the parameters of the main Q-networks are used to find an RB allocation solution for the end-users in the online implementation phase of the DQL algorithm. The implementation phase is presented in Algorithm 6. This phase uses the trained DDQNs of the agents. At the beginning of each episode, it builds the environment of each agent. Then, for each step, when a new state is observed, each

Algorithm 6 DQL Algorithm Implementation Phase

Input: The trained DDQNs
Output: RB allocation for end-users

Load the DDQN of each agent;
for *each episode* **do**
  Reset and build the agents' environment;
  for *each step* **do**
    for *each agent* b **do**
      Obtain observation $\mathcal{S}_b$;
      Choose $\boldsymbol{a}_b$ that maximize the Q-function;
    end for
    Obtain the joint action $\boldsymbol{a}$;
    Find a solution to RB allocation for end-users;
  end for
end for

agent b selects the action that maximizes the Q-value of current state. Once all agents have chosen their actions, each agent can form a joint action. Accordingly, an RBs allocation solution is obtained.

5.7 SIMULATION RESULTS

This section investigates the performance of the proposed two-level RB allocation mechanism through several simulated scenarios.

5.7.1 Experiment scenarios and setup

We consider an SDN-enabled RAN architecture where the gNodeBs are deployed in a square of area $1\,\mathrm{km}^2$. The end-users are uniformly distributed across the entire coverage area where each one of them is associated to only one gNodeB. Each end-user is assumed to be either an eMBB end-user or a URLLC end-user. For the sake of simplicity, the transmission power $P^s_{u_b}$ is the same for all $u_b \in \mathcal{U}_b$ and $b \in \mathcal{B}$. The key parameters of the simulations are summarized in table 5.1. These parameters were chosen based on common settings in the literature [62, 115]. In order to select an effective Q-network model, the training phase of the DQL algorithm is performed on a laptop with

an Intel Core i7-8750H processor, 16 GB of RAM and NVIDIA GeForce GTX 1070 graphic card. We create and train the DDQNs of the agents using the PyTorch framework. To select the best hyperparameters values for training the DDQN models, extensive simulations are performed. Indeed, we tested random combinations of hyperparameters in a fine-grained set of values chosen based on common settings in the literature [67, 140]. In particular, each DDQN consists of two fully connected hidden layers, each with 256 neurons. Rectified linear unit (ReLU) is used as the activation function to avoid the vanishing gradient problem in backpropagation, which accelerates the learning process. The Adam optimizer is used with a learning rate of 0.001 since it is computationally efficient. During the training process, the weights of the main Q-network are copied to the weights of the target Q-network every 1000 steps to avoid overestimating the Q-values. In addition, we consider end-users with low mobility, so the channel gains between a gNodeB and an end-user remain unchanged for a certain period of time. To do so, we fix the location of the end-user for a few training episodes, which helps the learning algorithm to better acquire the dynamics of the end-users and, at the same time, stabilize the training. The other hyperparameters of DDQN are given in table 5.2.

Table 5.1 Simulation parameters.

Parameter	**Value**
Number of gNodeBs	2
Total number of end-users	8
Bandwidth of an RB	180 KHz
Transmit power of gNodeB, $P^s_{u_b}$	30 dBm
Power of AWGN, σ^2	-114 dBm
Packet arriving rate per end-user, $\lambda^s_{u_b}$	100 packets/s
Packet length for an eMBB & URLLC end-user	400 & 120 bits
Minimum data rate for eMBB end-user, $\mathcal{R}_{min}$	100 kbps
Maxim delay for URLLC end-user, $\mathcal{D}_{max}$	10 ms

Table 5.2 Retained hyper-parameters for DDQN.

Hyper-parameter	**Value**
Learning rate	0.001
Epsilon/ϵ-greedy	1
Discount factor	0.996
ϵ-min	0.01
Size of replay memory	100000
Size of mini-batch	64
Target network update interval	1000 steps
Loss function	Mean squared error
Optimizer	Adam
Activation function	ReLu

5.7.2 DDQN training results

To assess the training performance of the proposed multi-agent DRL approach, we observe the cumulative rewards per training episode and the behavior of the loss function during the training process.

Figure 5.3 shows the cumulative average rewards of agents per episode. From this figure, the cumulative rewards improve as the number of training episodes increases, which demonstrates the effectiveness of the proposed training algorithm. When the training episode approximatively reaches 1800, the agents gain interesting experiences and start to exploit better actions. Accordingly, the cumulative reward approaches to a maximum value, indicating that the training process converges after an acceptable number of training episodes. Note that the convergence of the DQL algorithm does not present large fluctuations which are principally due to the low-mobility of the end-users in the environment.

Figure 5.4 illustrates the convergence of the loss function, equation (5.18), during the training process of the DDQNs. It plots the average of the agents' loss results versus the episodes. The loss decreases with the increase in training episodes. In the first few episodes, the loss declines gradually because various new actions were explored randomly and as learning progressed, good

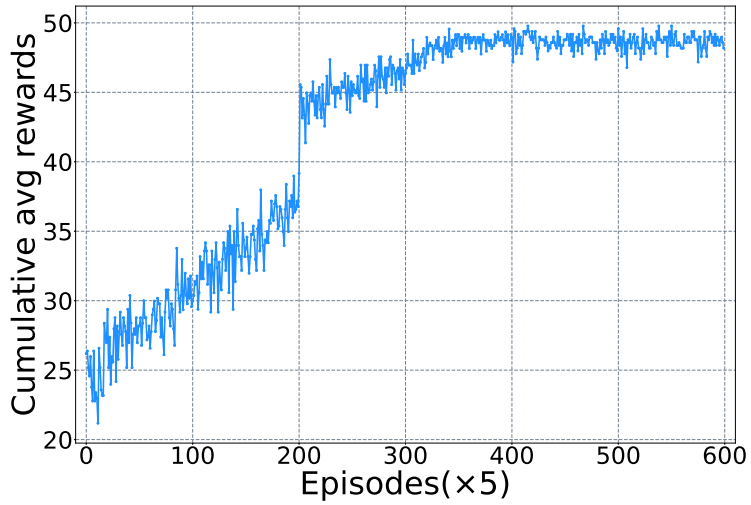


Figure 5.3 Training rewards.

actions were selectively performed based on the Q-value function that had become more reliable. Consequently, after episode 2500, the loss converges to a minimum value, which demonstrates the accurate Q-value approximation.

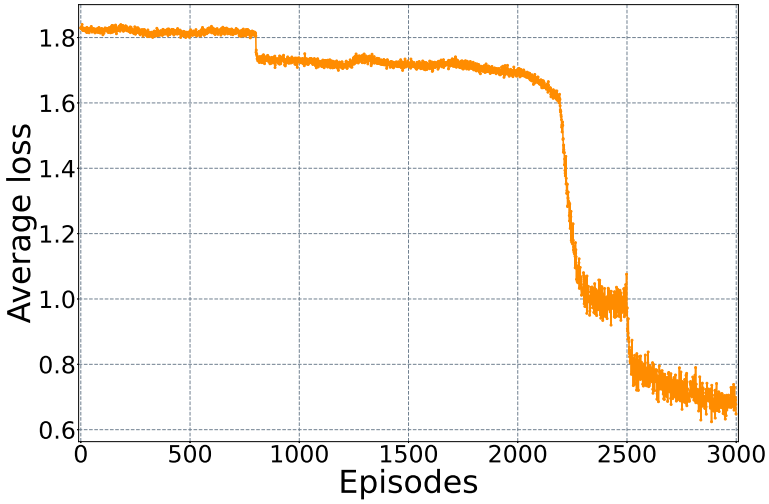


Figure 5.4 Training loss.

It is worth noting that the convergence results of the reward and loss function are obtained by assuming that the gNodeBs have perfect knowledge of the channel state information, which is used as input of the DDQN algorithm.

In the case of imperfect channel state information, the convergence time of the DDQN algorithm may increase [141]. Therefore, the DDQN algorithm takes more time to accurately learn the appropriate policies. But, once the offline training of the DDQN algorithm is performed, the learned policy can be applied rapidly to obtain the resource allocation solution.

5.7.3 SAMA-RL performance Evaluation

To benchmark our SAMA-RL approach, we implemented a DRL-based radio resource allocation approach from the literature [115], which we called 3-SRA for three-stage resource allocation. 3-SRA allocates radio resources to end-users by considering two types of slices, namely the rate constrained slice (*i.e.,* eMBB slice) and the delay constrained slice (i.e., URLLC slice). In the first stage, a central controller uses a heuristic algorithm to reserve radio resources (i.e., a fraction of the system bandwidth) to the eMBB and URLLC slices in each gNodeB. In the second stage, the reserved radio resources for each slice in each gNodeB are adjusted using a DRL algorithm. In the final stage, a heuristic algorithm is deployed to map the fraction of bandwidth assigned to a slice with the physical resource blocks. Accordingly, we have chosen 3-SRA since it: 1) is a competitive approach that performs the slicing operation of radio resources in a hierarchical manner, 2) is integrated into a network architecture similar to our proposed architecture, for instance, the presence of a central controller that has a global view of the RAN, 3) considers two types of slices, namely the eMBB slice and the URLLC slice, and 4) is based on a DRL algorithm to solve the radio resource allocation problem in the RAN.

To adapt 3-SRA to our model, two gNodeBs cannot serve a single end-user under the 3-SRA algorithm. Also, to fairly benchmark the performance of SAM-RL against 3-SRA, most of the simulation parameters are chosen similarly to those used in 3-SRA. Note that the comparison results are averaged over 1000 trials.

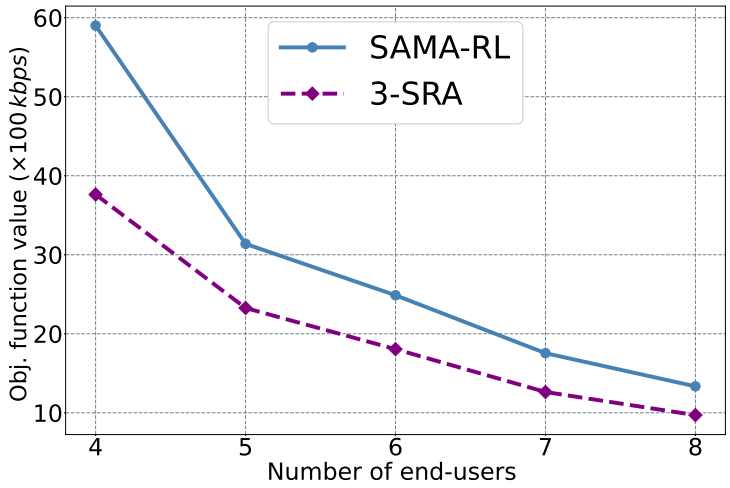


Figure 5.5 Impact of the number of end-users on the objective function.

Figure 5.5 shows the performance of SAMA-RL compared to the benchmark approach when varying the number of end-users. SAMA-RL always achieves the best performance in terms of the objective function value when the number of end-users increases. We notice that, for both approaches, the value of the objective function, that represents the total sum data rates of the URLLC and eMBB end-users, decreases as the number of end-users increases, which is due to the increase of the competitiveness among end-users to obtain a sufficient number of RBs that guarantees their requirements. Indeed, in the case of a low number of end-users, the agent can select an RB allocation action where several RBs are assigned to one end-user to guarantee its requirements in terms of data rate and delay. On the other hand, when the number of end-users is large, the agent seeks to allocate a minimum number of RBs to each end-user in order to satisfy as many end-users as possible in terms of data rate and delay requirements.

Figure 5.6 illustrates the impact of the minimum data rate threshold ($\mathcal{R}_{min}$) on the objective function equation (5.7a). Based on these results, we make the following observations: (1) SAMA-RL outperforms 3-SRA for all minimum

data rate thresholds; (2) the higher is the minimum data rate threshold, the larger is the performance gap between SAMA-RL and 3-SRA. This is due to the multi-agent approach proposed in the gNodeB allocation level where a gNodeB can borrow RBs from other adjacent gNodeBs when the RBs pre-allocated by the SDN controller are insufficient. This indeed illustrates the effectiveness of the proposed multi-agent method and demonstrates its robustness.

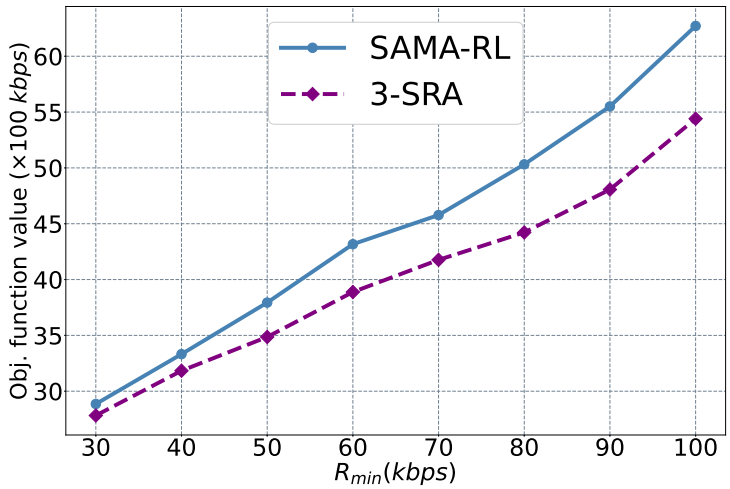


Figure 5.6 Impact of the minimum data rate threshold ($\mathcal{R}_{min}$) on the objective function.

Figure 5.7 presents the comparison of SAMA-RL and 3-SRA in terms of the average number of end-users whose achieved data rate is greater than or equal the minimum data rate threshold $\mathcal{R}_{min}$. We can see that SAMA-RL always outperforms 3-SRA for all minimum data rate thresholds. These results can be justified as follows: in 3-SRA, if the reserved resources for a given slice are not sufficient to provide the required service, *i.e.,* minimum data rate threshold, the slice must wait for the next resources reservation update, while in SAMA-RL, a gNodeB can request, if necessary, some RBs from other gNodeB.

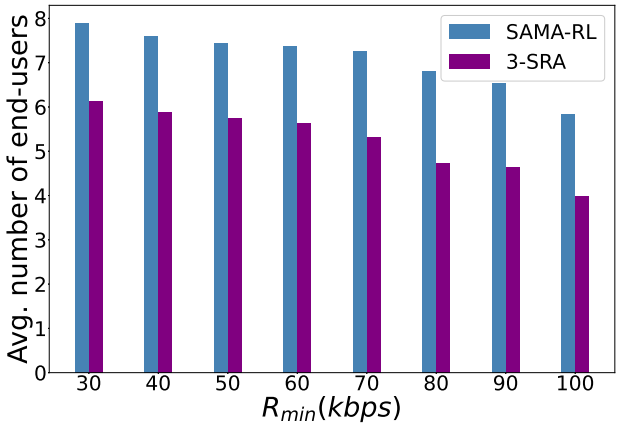


Figure 5.7 Impact of the minimum data rate threshold ($\mathcal{R}_{min}$) on the number of end-users.

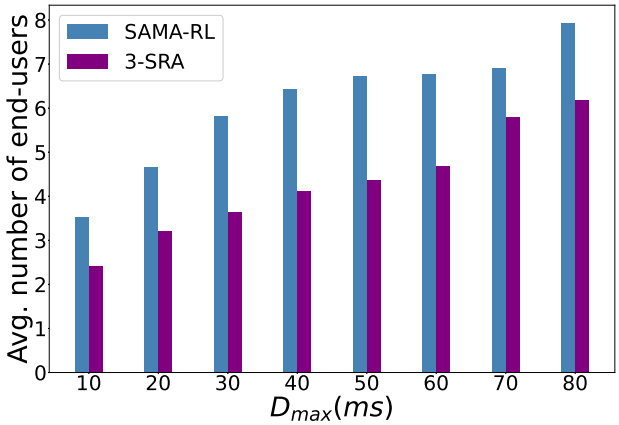


Figure 5.8 Impact of the the maximum delay threshold ($\mathcal{D}_{max}$) on the number of end-users.

In figure 5.8, we compare the performance of SAMA-RL and 3-SRA in terms of the average number of end-users whose delay experienced by a packet is less than or equal to the maximum delay threshold $\mathcal{D}_{max}$. Similar to the obtained results in figure 5.7, we can once again confirm that SAMA-RL gives better performance compared to 3-SRA for all maximum delay thresholds.

5.8 Conclusion

Toward an efficient radio resource slicing of an SDN-enabled RAN, we proposed a two-level RAN slicing mechanism where the allocation of radio RBs

is performed in the SDN level and in the gNodeBs level. The SDN level allocates RBs to gNodeBs in a large time-scale while the gNodeBs allocate their RBs to its associated end-users in a short time-scale to efficiently meet their dynamic requirements. Moreover, each gNodeB can borrow some RBs from other gNodeBs when the pre-allocated RBs are insufficient, which decreases the signaling overhead between the two allocation levels and rapidly provide the needed resources to its end-users. We formulated a data rate maximization problem subject to the ultra-low latency requirements of URLLC services as well as to the minimum data rate requirements of eMBB services. Subsequently, we proved its NP-hardness. Then, we modeled the SDN allocation level problem and the gNodeB allocation problem as a single MDP and a multi-agent MDP, respectively. We adapt the EXP3 algorithm and the DQL algorithm to solve the SDN allocation problem and the gNodeB allocation problem, respectively. In addition, the DQL algorithm applies robust state-of-the-art approaches such as double DQN and replay memory to improve its performance. Simulation results have shown that the proposed two-level mechanism yields significant improvements in terms of RBs allocation. Furthermore, our mechanism outperformed a benchmark algorithm by ensuring the requirements of the URLLC and eMBB services in terms of data rate and delay.

Chapitre 6: Avant-propos

Auteurs et affiliation:

Abderrahime Filali: étudiant au doctorat, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Boubakr Nour: Chercheur post-doctoral, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Soumaya Cherkaoui: Professeure, Université de Sherbrooke, Faculté de génie, Département de génie électrique et de génie informatique, Laboratoire de recherche INTERLAB.

Abdellatif Kobbane: Professeur, Université Mohammed-V, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Rabat-Maroc.

Date de soumission: août 2021.

État de l'acceptation: en cours sous révision.

Revue: IEEE Communications Standards Magazine.

Titre français: Découpage des ressources O-RAN de communication et de calcul pour les services URLLC à l'aide de l'apprentissage par renforcement profond.

Résumé français:

L'évolution des futurs réseaux 6G et au-delà de la 5G vers un réseau adapté aux services repose sur la technologie de découpage du réseau. Avec le découpage du réseau, les fournisseurs de services de communication visent à répondre à toutes les exigences imposées par les secteurs verticaux, notamment les services de communication ultra-fiable à faible latence (en anglais URLLC). En outre, l'architecture du réseau d'accès radio ouvert (en anglais open radio access network, O-RAN) ouvre la voie à un partage flexible des ressources du réseau en introduisant plus de programmabilité dans le RAN. Le découpage du RAN est une partie essentielle du découpage du réseau de bout en bout, car il assure un partage efficace des ressources de communication et de calcul. Cependant, en raison des exigences strictes des services URLLC et de la dynamique de l'environnement RAN, le découpage du RAN représente un véritable défi. Dans cet article, nous proposons une approche de découpage du RAN à deux niveaux basée sur l'architecture O-RAN pour allouer les ressources de communication et de calcul du RAN entre les dispositifs finaux URLLC. Pour chaque niveau de découpage du RAN, nous modélisons le problème de découpage des ressources sous la forme d'un processus de décision de Markov à un seul agent et concevons un algorithme d'apprentissage par renforcement profond pour le résoudre. Les résultats des simulations démontrent l'efficacité de l'approche proposée pour répondre aux exigences de qualité de service souhaitées.

Chapitre 6: Foreword

Authors and affiliation:

Abderrahime Filali: Ph.D. Student, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Boubakr Nour: Postdoctoral Research Fellow, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Soumaya Cherkaoui: Professor, INTERLAB Research Laboratory, Faculty of Engineering, Department of Electrical and Computer Science Engineering, Université de Sherbrooke.

Abdellatif Kobbane: Professor, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS) Mohammed V University in Rabat, Morocco

Date of submission: august 2021.

Acceptance status: under revision.

Journal: IEEE Communications Standards Magazine.

Title: Communication and Computation O-RAN Resource Slicing for URLLC Services Using Deep Reinforcement Learning

CHAPTER 6

Communication and Computation O-RAN Resource Slicing for URLLC Services Using Deep Reinforcement Learning

6.1 Abstract

The evolution of the future beyond-5G/6G networks towards a service-aware network is based on network slicing technology. With network slicing, communication service providers seek to meet all the requirements imposed by the verticals, including ultra-reliable low-latency communication (URLLC) services. In addition, the open radio network (O-RAN) architecture paves the way for flexible sharing of network resources by introducing more programmability into the RAN. RAN slicing is an essential part of end-to-end network slicing since it ensures efficient sharing of communication and computation resources. However, due to the stringent requirements of URLLC services and the dynamics of the RAN environment, RAN slicing is challenging. In this article, we propose a two-level RAN slicing approach based on the O-RAN architecture to allocate the communication and computation RAN resources among URLLC end-devices. For each RAN slicing level, we model the resource slicing problem as a single-agent Markov decision process and design a deep reinforcement learning algorithm to solve it. Simulation results demonstrate the efficiency of the proposed approach in meeting the desired quality of service requirements.

6.2 Introduction

Although fifth-generation (5G) standards are not yet fully finalized, the roadmap for sixth-generation (6G) networks is already taking shape due to several in-

dustrial and academic research efforts [142]. 6G networks are expected to support more diversified services than 5G networks aiming to create compelling business opportunities in many vertical industries. Achieving this requires (i) improving the technologies behind the evolution of 5G, such as network slicing [58], and (ii) leveraging machine learning (ML)/artificial intelligence (AI) techniques, such as deep reinforcement learning (DRL) for an efficient management of network resources. In addition, to meet the requirements of various industries, 6G should not only rely on new enabling technologies but also provide an innovative network architecture beyond current network designs. Open radio access network (O-RAN) is a key component of this architectural transition to more open and intelligent networks [17]. The O-RAN approach sustains the disaggregation between hardware and software to create a multi-supplier RAN solution through open and interoperable protocols and interfaces. The O-RAN specification, which is still compliant with 3GPP standards, introduces the hierarchical RAN Intelligent Controller (RIC), including non-real-time RIC (non-RT) and near-real-time RIC (near-RT) where ML/AI algorithms are integrated to enable RAN programmability.

RAN slicing is a critical component of end-to-end network slicing, as it determines the degree of flexibility network operators have to meet the needs of new verticals [58]. In particular, ultra-reliable low-latency communications (URLLC) is the foundation for emerging mission-critical applications in 6G networks, such as autonomous driving, industrial IoT, e-health (*e.g.,* remote surgery) and mobile or m-health (*e.g.,* patient monitoring and virtual reality-assisted care in ambulances). Due to the stringent requirements of these applications, they are expected to rely on multi-access edge computing (MEC) to deliver added value services to the end users. Therefore, effective management of RAN slicing will rely on the ability to optimally manage communication and computing resources placed at the MEC [85].

Considerable efforts have been devoted to improving the performance of RAN slicing to efficiently offload tasks at the MEC [143], where the RAN resource slicing problem is usually formulated using optimization techniques [144]. However, due the dynamics of the RAN environment, solving the problem of RAN slicing is complex and challenging to solve in polynomial time.

To overcome these issues, 6G RAN slicing operations will need to be performed with more intelligent resource allocation capabilities that achieve delay-efficient performances. Under the O-RAN architecture, RICs can dynamically create multiple RAN slices tailored to URLLC services using ML/AI capabilities such as DRL algorithms. Indeed, non-RT and near-RT RICs can leverage DRL's excellent learning ability and effectiveness in solving complex and dynamic environment problems, such as the RAN environment, to make optimal RAN slicing decisions for URLLC services [145].

In this work, we are motivated to apply the DRL algorithms within the O-RAN architecture to jointly slice the communication and computation resources at the RAN level for URLLC task offloading operations. Indeed, we propose a two-level RAN slicing approach based on DRL. The first RAN slicing level, called the communication slicing level, concerns the allocation of radio resources to end-devices. The second RAN slicing level, called the computation slicing level, deals with the allocation of computation resources to end-devices. The contribution of our work is as follows. We first introduce the RAN slicing paradigm and its associated challenges and highlight the role of non-RT and near-RT RICs in performing RAN slicing operations in an O-RAN architecture. Then, we model each RAN slicing resource level as a single agent Markov decision process. Next, we propose, for each RAN slicing level, a DRL algorithm to solve it. Finally, we illustrate through extensive simulations that the proposed approach exhibits fast convergence and achieves delay-efficient performance.

6.3 Unveiling the Curtain: Network Slicing

Network slicing is the transformation of a physical network into a set of logical networks on top of a shared infrastructure. This logical separation aims to meet the emerging requirements of a wide range of verticals. Each logical network, *i.e.,* network slice, is defined in a way to meet efficiently the business requirements of a vertical application by providing the needed network resources from the RAN to the core network [146]. In particular, RAN slicing [58] is a critical part of end-to-end network slicing to enable differentiated traffic processing and isolation. This can be achieved through application-based prioritization of data, resource allocation, and scheduling.

To date, various efforts have been presented to improve URLLC services through RAN slicing. The Third Generation Partnership Project (3GPP) has made significant standardization efforts to define RAN slicing specifications and promote its implementation. For example, 3GPP introduced the RAN slicing management framework to manage the life cycle of RAN slices [146]. In addition, it provided efficient solutions that allow end-devices to rapidly access a cell and select the desired RAN slices [147]. To satisfy as many RAN slices as possible, [148] proposes a portioning algorithm that allocates radio resources to RAN slices according to their prioritization. A joint RAN slicing framework for communication and computation resources has been developed in [135]. Communication and computation resources are allocated to RAN slices in order to minimize the delay needed to offload and process time-sensitive users' tasks. To support a maximum number of RAN slices while meeting their performance requirements, [143] proposes to share the radio resource between RAN slices by allocating, to each of them, a fraction of bandwidth that maximizes the access probability to the base station and the energy efficiency of end-devices.

Reinforcement learning-based RAN slicing approaches have also emerged as practical solutions with low computational complexity, simplifying implementation for real-world applications. For instance, [66] introduces an RL-based framework to dynamically allocate radio spectrum and computation resources to RAN slices. The allocation process considers the delay as the primary QoS metric that should be less than a maximum threshold. [149] employs DRL to design a decentralized RAN resource orchestration system. The latter includes an agent to slice each RAN resource and a central coordinator that manages the resource orchestration between the agents. Each orchestration agent uses DRL to allocate its resources to the RAN slices, while the central coordinator ensures SLA requirements.

Despite the aforementioned solutions, various issues remain open. These issues could be summarized as follows:

- *Resource Sharing:* Efficient resource sharing is a primary objective of RAN slicing. However, when a slice is instantiated, dedicated resources may become unavailable to others. Resources reallocation among slices may further enhance optimizing resources utilization as well as improving the network performance. However, dynamic changes in network load, end-devices mobility, and task distribution make resource reallocation challenging. Furthermore, to fulfill the minimum requirements of vertical applications, efficient scheduling mechanisms to allocate radio resources among slices are required while maintaining efficient sharing of computation resources between MEC servers.
- *Dynamic Slice Creation and Management:* In light of the previous point, optimizing resource allocation is indispensable to maximize verticals' benefits, where dynamic slice creation and management are critical during the slice lifecycle. With the aim to accommodate a maximum amount of service requests with minimum resources, the network operator needs to deploy various dynamic functions and mechanisms to quickly create

and manage slices. Verticals, on the other hand, must have partial permission and the ability to configure the slices while ensuring a high level of security and privacy.

- *Mobility Management:* Today's users may shift from a network to another while requesting services. Seamless handover and interference management add more challenges to RAN slicing. For instance, it is critical to ensure fast mobility handover for real-time services. The system performance relies on the performance of the handover mechanism. Therefore, there is a need for a slice-oriented mobility management protocol to tackle the mobility issues in RAN slicing.
- *Algorithmic Aspects of Resource Allocation:* Resource allocation is a challenging problem that often encompasses many parameters and constraints. Different algorithms have been adopted to solve the problem according to its complexity. Exact algorithms can be applied to find optimal solutions for less complex problems, while meta-heuristic algorithms are more efficient when dealing with more complex problems. Therefore, practical and efficient resource allocation algorithms are necessary with the ability to reconfigure slice resources based on the dynamic network changes.

With O-RAN, the door is now unlocked to enhance RAN slicing and address many of its challenges using the non-RT and near-RT RICs [150]. The former handles the heaviest RAN functions, at a time scale $>$ 1s, including robust RAN analytics, control policy design, providing trained AI/ML models and guidance to support near-RT RIC operations. The latter executes critical RAN functions, at a time scale that could be as low as 10ms, to interpret and enforce the received policies from non-RT RIC such as using AI/ML inference to control RAN behavior. Therefore, the interaction between non-RT and near-RT RICs can be used to design and fine-tune efficient AI/ML control algorithms for RAN slicing.

6.4 Joint Slicing of Communication and Computation RAN Resources

We describe, in the following, the proposed two-level RAN slicing approach, where the communication and computation RAN resources are jointly sliced and allocated to the end-devices according to their URLLC requirements.

6.4.1 System Model

Network Model: We consider an O-RAN-based cellular network architecture, as depicted in figure 6.1, composed of five network components: (i) non-RT RIC that is directly connected to near-RT RIC through A1 interface, MEC servers, and gNodeBs through O1 interface to enables non-real–time control and optimization of RAN elements and resources, (ii) near-RT RIC that performs near-real-time control and optimization of O-RAN elements and resources over the E2 interface, (iii) a set of MEC servers, controller by the near-RT RIC, among which we consider a group of servers relatively close to each other as a MEC server sharing group, (iv) a set of gNodeBs that provide communication resources to URLLC end-devices in their coverage area, and (v) URLLC end-devices that offload their computing tasks under URLLC constraints, *i.e.,* strict latency, to the MEC servers.

Each gNodeB is attached to one MEC server to provide computation resources to URLLC end-devices. A gNodeB sharing group consists of a group of gNodeBs with highly overlapped in their communication coverage areas. The communication and computation resources, considered in this work, are the resource block (RB) of gNodeB and the CPU core of the MEC server, respectively. RB is the smallest unit of radio resources that can be allocated to an end-device. A CPU core is defined as the computation capability in terms of CPU cycles per second. We also consider the orthogonal frequency division multi-access offloading scenario, where the radio resources of a gN-

odeB are divided into multiple RBs. Hence, we avoid intra-cell interference where a specific RB is exclusively assigned to only one end-device.

Assumptions: In our model, we consider the following assumptions: (i) since radio resources are limited, it is challenging to provide enough orthogonal radio resources (in a multi-cell scenario). Thus, some gNodeBs can share the same radio resources, which may cause interference between cells. To counterbalance radio resources sharing and inter-cell interference reducing, the same set of radio resources can be assigned to multiple gNodeBs as long as the distance between them is sufficient to reduce inter-cell interference; (ii) since a gNodeB sharing group is an area with strong overlaps between gNodeBs, the orthogonal resources are assigned to gNodeBs within one sharing group, which means the unavailability of interference within gNodeB sharing group; (iii) each gNodeB covers a set of end-devices that are uniformly distributed in the gNodeB's coverage area; (iv) each end-device is associated with only one gNodeB; and (v) each MEC server is equipped with multiple CPU cores to provide parallel computing.

The near-RT RIC performs the slicing operation of communication and computation RAN resources in two levels: (a) *communication slicing level*, and (b) *computation slicing level*

Communication Slicing Level: Each gNodeB assigns a number of RBs to its associated end-devices. The objective is to meet the QoS requirements of the URLLC services (*e.g.,* delay). The RBs allocated to each end-device should ensure a low communication delay in offloading the task from end-device to the associated gNodeB through wireless transmission. A task's communication delay depends on its size and the total attainable data rate over the allocated RBs. Each end-device can be considered as an M/M/1 queuing system under the following assumptions: (i) the arrival process of each end-device's tasks follows a Poisson distribution, and (ii) the inter-arrival

times of the tasks are independent and follow an exponential distribution. Therefore, the delay experienced by a given task, in an offloading operation, can be calculated by applying Little's law.

Computation Slicing Level: The computation resource slicing consists of allocating the required CPU cycles to successfully execute the offloaded tasks and meet the required QoS requirements. In fact, for each arrival task in each MEC server, the near-RT RIC needs to decide: (i) where the task should be executed, and (ii) how many computation resources should be allocated to this task. For a given task, the near-RT RIC checks the available computation resources of the associated MEC servers, based on which it decides whether the task could be executed locally by its associated MEC server or forwarded to another MEC server in the same sharing group. Then, the near-RT RIC allocates the required CPU cycles to execute this task. The computation delay can be defined as the ratio of the number of CPU cycles required to accomplish this task to the CPU cycles allocated by the near-RT RIC. When a task is forwarded to a different MEC server, the round-trip communication delay is added to the computation delay.

6.4.2 Deep Reinforcement Learning based RAN Resource Slicing

In an O-RAN architecture, the near-RT RIC is responsible for making resource allocation decisions. The efficiency of these decisions impacts the performance of the overall system. In particular, each gNodeB communicates the state of its environment, through the E2 interface, with the near-RT RIC that allocates the required communication resources, *i.e.,* RBs, to the end-devices associated with this gNodeB. Similarly, the near-RT RIC collects information about the computation resource status of the MEC servers and allocates the resources, *i.e.,* CPU cycles, needed to execute the offloaded tasks in the appropriate MEC servers. Intending to ensure URLLC services, communication and computation resource slicing operations become very challenging to solve, especially in large-scale networks where the number of end-devices is

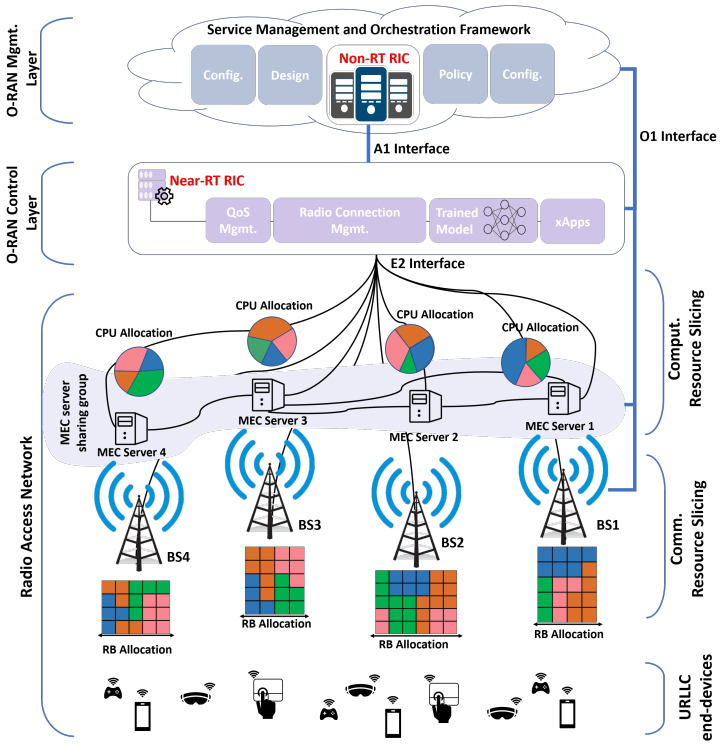


Figure 6.1 Reference network RAN slicing model.

huge. To overcome this challenge, DRL can be applied since it can efficiently deal with the curse of dimensionality problem. Figure 6.2 illustrates the overall working principle of the proposed DRL-based RAN resource slicing in an O-RAN architecture.

In this work, we opt for deep Q-learning (DQL), a traditional DRL algorithm [131], to solve the RAN resource allocation problem in both communication and computation slicing. Contrary to classic Q-learning algorithms, DQL uses a deep neural network (DNN) as a Q-function approximator. This extension of the Q-learning algorithm is known as the deep Q-network (DQN) algorithm. Indeed, for a given input state, DQN generates a Q-value of all possible actions. The agent frequently interacts with its environment to effectively enhance its decision-making. The agent's experience is defined by the tuple (current state, action, reward, next state). Instead of immediately training the DNN by feeding it with successive experience tuples, they are

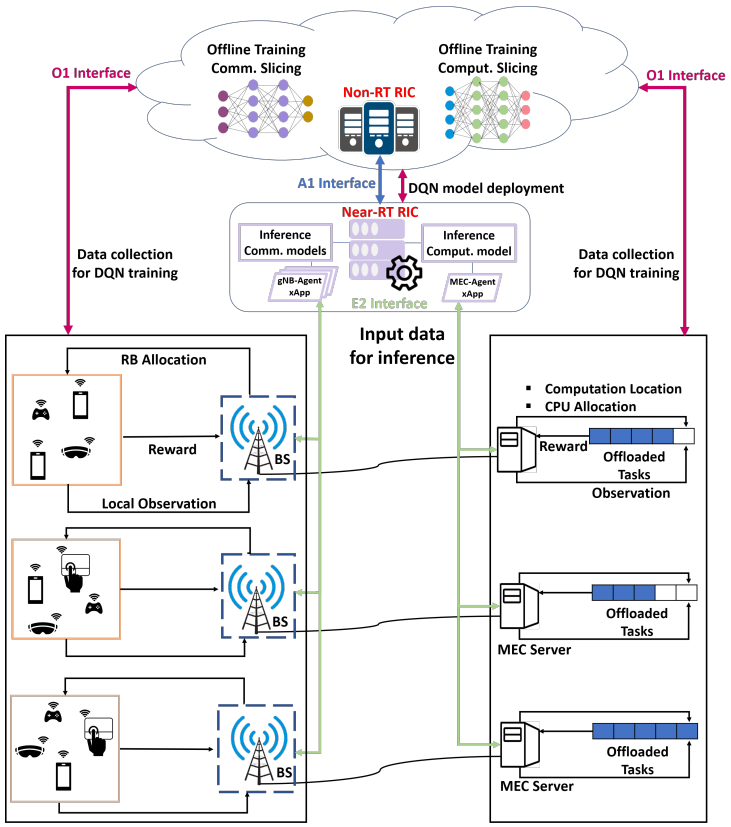


Figure 6.2 Deep Reinforcement Learning based RAN Resource Slicing.

stored in a replay buffer according to the time sequence. During the DQN training process, the stored experiences are randomly sampled to train the DNN. The experience replay memory strategy allows efficient use of previous experiences in the DNN training process since it breaks the correlations in the observation sequences.

To further stabilize the approximation of the Q-value function, we employ the double DQN (DDQN) algorithm. DDQN mitigates the overestimation problem that occurs in DQN algorithms since it applies a maximization operation on both the selection and evaluation actions. Specifically, DDQN uses two neural networks: main Q-network to select action, and target Q-network to calculate the estimated Q-value of each selected action. The main Q-network is trained by minimizing the loss function. The latter calculates the mean

square error between the current Q-values of actions selected by the main Q-network and their estimated Q-values calculated by the target Q-network.

In the O-RAN architecture, figure 6.2, we consider that each gNodeB is controlled by a DRL agent, called gNodeB-agent, which performs the communication resource slicing between the associated end-devices of this gNodeB. For the computation slicing level, we consider that the MEC server sharing group is controlled by a DRL agent, called MEC-agent, which allocates the CPU cycles required to successfully execute the offloaded tasks. Each DRL agent runs in a xApp on the near-RT RIC and manages its resources through the E2 interface.

Before describing the proposed DRL-based approach, we first model each resource slicing problem as a Markov decision process (MDP).

MDP-based Communication Resource Slicing: Each gNodeB-agent observes its environment and allocates RBs to its associated end-devices. For each gNodeB, the communication resource slicing is modeled as a single-agent MDP given by the following state space, action space, and reward function.

- *The State Space:* The state space of a gNodeB-agent includes the: (i) set of associated end-devices, (ii) radio resource that represents the available RBs, (iii) channel gain between the gNodeB and its associated end-devices over a given RB, and (iv) maximum delay threshold required by the URLLC service.

- *The Action Space:* A gNodeB-agent has to decide which RBs should be allocated to each of the associated end-devices. Since an end-device can have more than one RB to meet the desired QoS, an action is defined by a row vector where each element represents the RB - end-device assignment.

- *The Reward Function:* The reward received by the gNodeB-agent depends on whether it successfully allocated the required RBs to the associated end-devices or not. An action is considered to be successful if it meets the constraints of the RB allocation model. Since our objective is to minimize the communication delay, the received reward is the inverse of the sum of all communication delays of all tasks offloaded by the associated end-devices. If an unsuccessful action is chosen, the gNodeB-agent is penalized with a negative reward.

MDP-based Computation Resource Slicing: The MEC-agent collects information about all MEC servers in the sharing group (*e.g.,* computation resources and offloaded tasks). It is able to observe the environment and make decisions. Therefore, we model the computation resource slicing as a single-agent MDP.

- *The State Space:* The state space of the MEC-agent is given by information about each MEC server including the offloaded tasks and the available computation resources. Since the observed state is unknown directly to the MEC-agent, each MEC server regularly updates the MEC-agent about its local state. An update can include task-related information such as the number of tasks currently in its buffer, the size of each task, the number of CPU cycles needed, and a maximum delay threshold required by the URLLC service.
- *The Action Space:* The MEC-agent decides the computation resource allocation for each offloaded task. A decision includes: (i) in which MEC server a task should be executed , and (ii) CPU cycles allocation that consists in determining the number of CPU cores to be assigned for computing a received task.
- *The Reward Function:* The reward obtained by the MEC-agent after taking an action depends on whether the chosen action is feasible or not

and at what level the computation delay was minimized. An action is considered feasible if it meets the computation resource allocation constraints (*e.g.,* maximum computation delay threshold). The received reward is the inverse of the sum of computation delays of all tasks offloaded by the end-devices. Otherwise, the received reward is set to a negative value to prevent the MEC-agent from choosing non-feasible actions in the future.

6.4.3 Deep Q-learning Slicing Algorithm

A DQL-based approach consists of two main phases: the training phase and the implementation phase, *i.e.,* inference. In the training phase, a DDQN is trained in an offline manner. In the implementation phase, the agent takes actions in an online manner based on its trained DDQN. In the O-RAN architecture, the DDQN model is trained offline in the non-RT RIC, while the model inference is deployed in the near-RT RIC. The non-RT RIC uses the O1 interface to collect data for offline model training. Note that the trained model can undergo an evaluation step validating that it is reliable for deployment in the near-RT RIC. The model inference is executed and fed with online data, through the E2 interface, to produce the slicing actions that will be used in the resource allocation operation. The training and implementation phases of both slicing levels are conducted in the same way [1], which can be summarized as follows.

The Training Phase: The training phase takes place in several episodes and requires, in each episode, the state of the environment as input. As output, a trained DDQN is produced. Systematically, to train the DDQN, the agent initializes the DDQN hyperparameters and collects information about its environment.

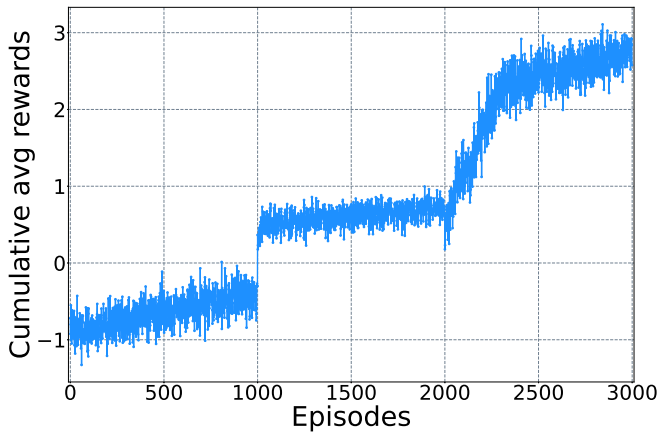
1. The term agent is used to refer to the gNodeB-agent or MEC-agent, based on the slicing level.

The learning process is then begun by iterating the episodes. At the beginning of each step for each episode, the agent observes the state of its environment and chooses an action according to an ϵ-greedy policy. With the help of ϵ-greedy policy, the training process is balanced between exploitation and exploration. At each step, the agent takes a random action with a probability of ϵ (*i.e.,* exploration) and follows its current policy by choosing the action with the highest Q-value in the remaining time (*i.e.,* exploitation). As the training process proceeds, the ϵ value gradually decreases, indicating that the agent becomes more confident to optimally interact with the environment and choosing optimal actions. The obtained experience tuple is stored in a replay buffer. When the buffer contains enough experiences, the agent picks a random sample to create training data. Then, it performs the gradient descent algorithm to minimize the loss function and update the parameters of the main Q-network. On the other hand, the target Q-network parameters do not need to be updated at each training step but replaced by the main Q-network parameters with a certain frequency.

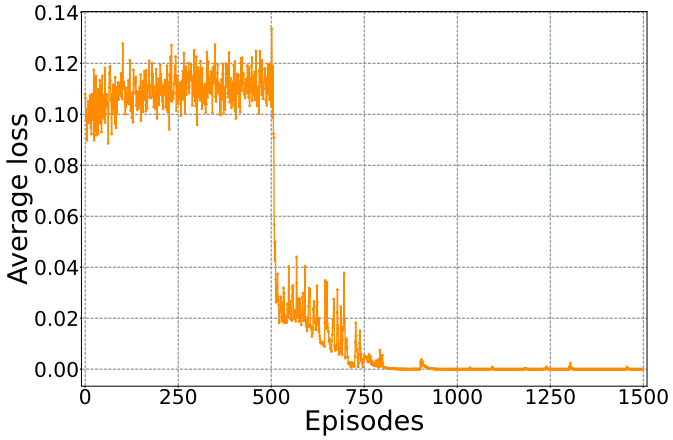
The Implementation Phase: Once the offline training phase is complete, the agents can use their trained DDQNs to efficiently allocate RBs and CPU cycles. During the implementation phase, when a new state of the environment is observed, the agent selects the best action (*i.e.,* the action with the highest Q-value). Afterward, end-devices can offload their tasks to the associated gNodeB using the optimal RBs. Then, tasks will be executed by the MEC servers using an optimal CPU cycle allocation.

6.5 Performance Evaluation

Simulation Setup and Scenario: Following the reference network model shown in figure 6.1, we implemented an O-RAN-based cellular network architecture with four gNodeBs. The gNodeBs are deployed in a geographical zone modeled by a square of a side of 2000 m. Each gNodeB covers a circular



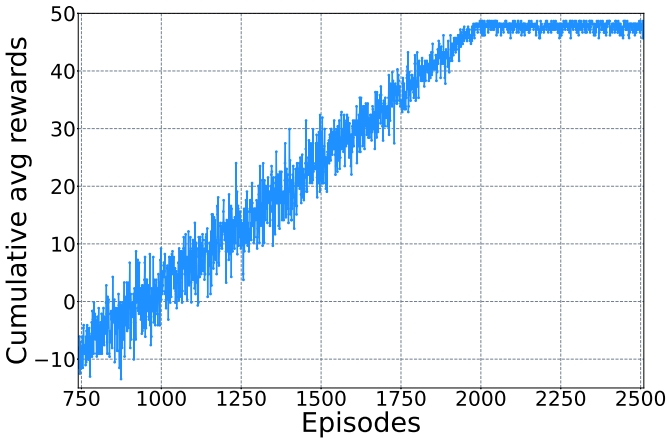
(a) Training reward.



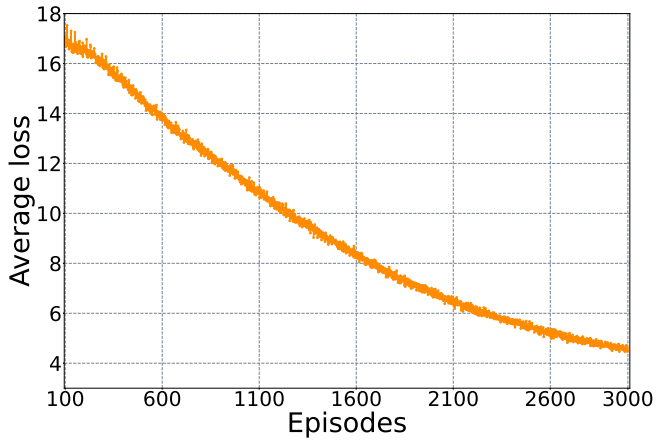
(b) Training loss.

Figure 6.3 Training performance of the communication model.

area with a radius of 500 m and is accompanied by one MEC server. Each MEC server is equipped with four CPU cores with a computation capability equals to 3 gigacycles each. End-devices, with URLLC services, are uniformly distributed within the coverage area. Each end-device is associated with only one gNodeB and can offload only one task at a time. The data size of each task is uniformly distributed from 0.5 MB to 2 MB and the required CPU cycles to compute one bit is 400. The transmission power of end-devices is 23 dBm, while the bandwidth of an RB is 180 kHz and the noise power is -114 dBm. The DDQNs were implemented and trained using the PyTorch frame-



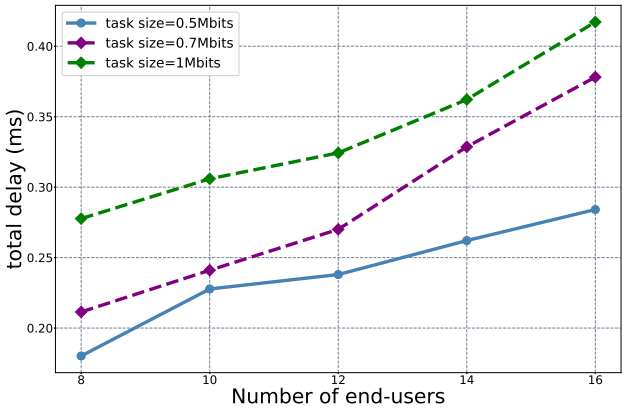
(a) Training reward of the computation model.



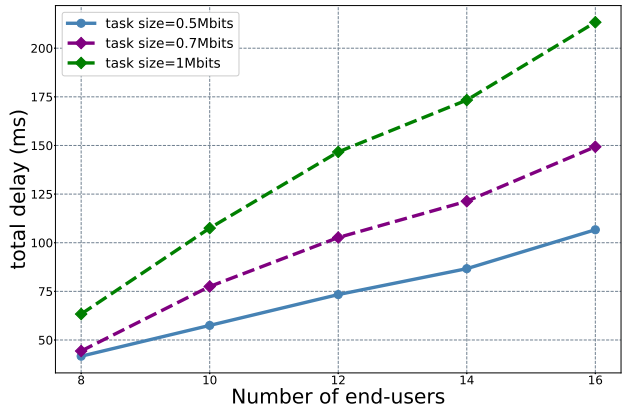
(b) Training loss of the computation model.

Figure 6.4 Training performance of the computation model.

work. For the training, we used two fully connected hidden layers composed of 256 neurons each, ReLu as the activation function, Adam as the optimizer, and the mean square error as the loss function. The non-identical hyperparameters include the learning rate and the mini-batch size. The learning rate of the communication model and the computation model is 0.01 and 0.001, respectively, while the mini-batch size is 64 and 256, respectively. We evaluated the performance for DDQN training as well as the performance of the delay experienced end-users.



(a) Communication model.



(b) Computation model.

Figure 6.5 Delay performance.

DDQN Training Performance: Figures 6.3 and 6.4 show the training performance of the communication and the computation models, respectively. Indeed, figures 6.3a and 6.4a illustrate the convergence of the communication and computation DQL algorithms, respectively, versus training episodes. They show the cumulative average rewards of the agent per episode. It can be seen from both figures that when the number of training episodes increases, the cumulative average reward grows. We also notice that the convergence of the computation DQL algorithm is faster (converge after 2000 episodes) than that of the communication DQL algorithm (from episode number 2500). The

convergence of the communication DQL algorithm is relatively slow due to the mobility of end-devices, so the channel gains between the gNodeBs and the end-devices change frequently. These convergence results demonstrate the effectiveness of the proposed algorithms.

Figures 6.3b and 6.4b show how the behavior of the loss function for both communication and the computation DQL algorithms, respectively, evolves as training proceeds. In the early stages of the training process, the performance of both algorithms is weak due to exploration phenomena, *i.e.,* the gNodeB-agents and the MEC-agent take random actions more than exploiting what they have learned. The loss value decreases to reach a minimum value at the end of the training process, which indicates that the Q-value approximation has become accurate.

Delay Performance: In this experiment, we evaluated the performance of the proposed RAN slicing approach in terms of the delay experienced by tasks. For each resource slicing model, we varied the number of end-device and calculated the delay experienced by tasks for different task sizes, *e.g.,* 0.5 MB, 0.7 MB, and 1 MB. Based on the observed results in figures 6.5a and 6.5b, we make the following observations: (1) it is clear that as the number of end-device increases, the delay experienced by the tasks increases, and (2) the performance gap between the three task sizes remains relatively constant for a different number of end-devices. For (1) when the number of the end-devices becomes higher, the competitiveness among end-devices increases to obtain sufficient RBs and CPU cycles. In fact, when the number of the end-devices is low, the gNodeB-agents and the MEC-agent can assign several RBs and CPU cycles, respectively, to only one end-device. In contrast, when the number of end-devices is high, the gNodeB-agents and the MEC-agent, respectively, assign a minimum of RBs and CPU cycles to satisfy all end-

devices. Observation (2) demonstrates the scalability of the proposed RAN slicing approach under a dense network topology.

6.6 Conclusion and Future Work

The use of dynamic resource allocation algorithms will undoubtedly be paramount to increasing the efficiency of network slicing in future mobile networks. While current management and orchestration frameworks offer fast resource allocation capabilities, these frameworks will also need to rely on fast intelligent algorithms to predict service demands and anticipate resource needs for optimal orchestration. Artificial intelligence, in particular DRL, can offer interesting techniques to provide such functionality. In this article, we designed a two-level RAN slicing approach to allocate communication and computation resources to URLLC end-devices. The approach is integrated in the O-RAN architecture with MEC technology. We modeled each RAN resource slicing problem as a single-agent MDP. Then, we developed a DQL algorithm to solve each resource slicing problem and described the role of non-RT and near-RT RICs in performing slicing operations. The proposed DQL-based solution shows robust and efficient performance in meeting the requirements of URLLC services. The results of this study show that a deep reinforcement learning based RAN resource slicing architecture such as the one presented is promising and deserves further investigation.

CHAPTER 7

Conclusions and Future Works

We presented throughout this thesis several approaches to improve the management of next-generation network resources by leveraging the SDN paradigm as a key enabler of edge computing and network slicing technologies. This chapter concludes the thesis and suggests new research directions for future works.

7.1 Conclusions

In chapter 3, we tackled the load balancing problem in distributed SDN control plane architectures. We proposed a proactive approach to fairly distribute the load among SDN controllers based on the data plane component migration mechanism. For this reason, we introduced a long-term prediction model using ARIMA—a stochastic prediction model—to forecast the load of the SDN controllers. Then, we formulated the load balancing problem as an optimization program with the objective to minimize the difference between the loads handled by the controllers. To solve the formulated optimization problem, we developed a heuristic algorithm that migrates the data plane components from overloaded controllers to underloaded controllers.

In chapter 4, we improved the proactive load balancing approach proposed in chapter 3. Specifically, we presented two prediction models to forecast the load of the SDN control plane. The first model is ARIMA and the second one is LSTM—a machine learning prediction method. Then, we conducted a comparative study between these models to evaluate their prediction accuracy. To improve the migration operations performance in balancing the load in the SDN control plane, we formulated the load balancing problem as a non-linear binary program considering the tradeoff between the load balanc-

ing degree in the control plane and the cost of migration operations. To solve this optimization problem, we designed a reinforcement learning algorithm.

Long-term forecasts enable preventive detection of whether the load of the control plane will be unbalanced and, consequently, trigger migration operations in advance. We demonstrated through extensive simulations that the proposed algorithms for load balancing in the SDN control plane outperform recent benchmark algorithms and have close to optimal performance.

In chapter 5, we addressed the slicing problem of communication resources in the RAN. We proposed an SDN controller-based RAN slicing mechanism to allocate the radio RBs to eMBB and URLLC end-users. This mechanism allocates the RBs in two time-scales. The SDN controller manages a shared pool of RBs and allocates, in a large time-scale, to each base station a number of RBs according to its requirements in terms of delay and data rate. Then, each base station schedules the pre-allocated RBs to its associated end-users in a short time scale. In addition, a base station can request additional RBs from other base stations if the allocated RBs by the SDN controller are not sufficient. To solve this RAN slicing problem, we formulated it as a non-linear binary program where the objective is to maximize the achievable data rate of the end-users subject to the ultra-low latency requirements of URLLC services as well as to the minimum data rate requirements of eMBB services. We leveraged reinforcement learning algorithms to solve the RB allocation to the base stations and end-users. Specifically, we adopted the EXP3 algorithm and the DQL algorithm to allocate RBs to base stations in a large time scale and end-users in a short time scale, respectively.

In chapter 6, we proposed a two-level RAN slicing approach where the communication and the computation resources are jointly sliced and allocated to end-devices. In the first level, each base station allocates a number of its radio RBs to each of its associated end-devices to offload their tasks. In the

second level, a central controller allocates the required computation resources of MEC servers to compute the tasks offloaded by the end-devices. We developed, for each RAN slicing level, a DRL algorithm to appropriately allocate the required resources. The proposed approach is designed in an O-RAN architecture, where we described how the non-RT RIC and the near-RT RIC can perform the RAN slicing operations.

The presented RAN slicing approaches, in chapter 5 and chapter 6, perform the allocation of the radio RBs and computation resources hierarchically. These hierarchical approaches prevent entrusting the allocation of RAN resources solely to a central control entity, *e.g.,* the SDN controller, thereby reducing network overhead due to frequent communication between network entities, such as base stations and the SDN controller.

7.2 Future Works

The works performed in this thesis open the door to several future research directions. We list in the following some perspectives for future works.

- In chapter 4, we formulated the load balancing problem as a non-linear binary program considering the tradeoff between a load balancing factor and the cost of migration operations. Investigating other objective functions is, obviously, an important and open research direction, either to improve the proposed algorithm or to develop new competitive algorithms.
- In chapter 5, we studied the RB allocation problem to base stations by the SDN controller, and provided only a classical reinforcement learning algorithm, i.e., EXP3. Developing more competitive algorithms for this problem is important and remains an open problem.
- In chapter 5, when we defined the system model, we considered the queueing traffic model as an M/M/1 queueing system and assumed that

the base stations have perfect knowledge of the channel state information. Considering more general systems is a potential research direction to propose more general solutions.

- In chapter 6, the proposed RAN slicing approach only considers the URLLC service requirements. Therefore, it can be improved to support other service requirements, such as eMBB and mMTC service requirements.

CHAPTER 8

Conclusions et Travaux Futurs

Nous avons présenté tout au long de cette thèse plusieurs approches pour améliorer la gestion des ressources dans les réseaux de nouvelle génération en tirant parti du paradigme SDN en tant qu'outil clé des technologies de l'informatique en périphérie et de découpage du réseau. Ce chapitre conclut la thèse et propose de nouvelles directions de recherche pour les travaux futurs.

8.1 Conclusions

Dans le chapitre 3, nous avons abordé le problème de l'équilibrage de charge dans les architectures de plan de contrôle SDN distribuées. Nous avons proposé une approche proactive pour répartir équitablement la charge entre les contrôleurs SDN en se basant sur le mécanisme de migration des composants du plan de données. Pour cette raison, nous avons introduit un modèle de prédiction à long terme utilisant ARIMA - un modèle de prédiction stochastique - pour prévoir la charge des contrôleurs SDN. Ensuite, nous avons formulé le problème d'équilibrage de charge comme un programme d'optimisation dont l'objectif est de minimiser la différence entre les charges gérées par les contrôleurs. Pour résoudre le problème d'optimisation formulé, nous avons développé un algorithme heuristique qui migre les composants du plan de données des contrôleurs surchargés vers des contrôleurs sous-chargés.

Dans le chapitre 4, nous avons amélioré l'approche proactive d'équilibrage de charge proposée dans le chapitre 3. Plus précisément, nous avons présenté deux modèles de prédiction pour prévoir la charge du plan de contrôle SDN. Le premier modèle est ARIMA et le second est LSTM - une méthode de prédiction par apprentissage automatique. Ensuite, nous avons mené une étude comparative entre ces modèles pour évaluer leur précision de prédiction. Afin

d'améliorer la performance des opérations de migration dans l'équilibrage de la charge dans le plan de contrôle SDN, nous avons formulé le problème d'équilibrage de la charge comme un programme binaire non linéaire en considérant le compromis entre le degré d'équilibrage de la charge dans le plan de contrôle et le coût des opérations de migration. Pour résoudre ce problème d'optimisation, nous avons conçu un algorithme d'apprentissage par renforcement.

Les prévisions à long terme permettent de détecter de manière préventive si la charge du plan de contrôle sera déséquilibrée et, par conséquent, de déclencher à l'avance des opérations de migration. Nous avons démontré par des simulations extensives que les algorithmes proposés pour l'équilibrage de la charge dans le plan de contrôle SDN surpassent des algorithmes de benchmark récents et ont une performance proche de l'optimum.

Dans le chapitre 5, nous avons abordé le problème du découpage des ressources de communication dans le RAN. Nous avons proposé un mécanisme de découpage du RAN basé sur un contrôleur SDN pour allouer les RBs radio aux utilisateurs finaux eMBB et URLLC. Ce mécanisme alloue les RBs à deux échelles de temps. Le contrôleur SDN gère un pool partagé de RBs et alloue, dans une échelle de temps large, à chaque station de base un nombre de RBs en fonction de ses besoins en termes de délai et de débit. Ensuite, chaque station de base attribue les RBs préalloués à ses utilisateurs finaux associés dans une échelle de temps courte. En outre, une station de base peut demander des RBs supplémentaires à d'autres stations de base si les RBs alloués par le contrôleur SDN ne sont pas suffisants. Pour résoudre ce problème de découpage du RAN, nous l'avons formulé sous la forme d'un programme binaire non linéaire dont l'objectif est de maximiser le débit de données atteignable par les utilisateurs finaux en tenant compte des exigences de latence ultra-faible des services URLLC ainsi que des exigences de débit

de données minimum des services eMBB. Nous avons utilisé des algorithmes d'apprentissage par renforcement pour résoudre l'allocation des RBs aux stations de base et aux utilisateurs finaux. Plus précisément, nous avons adopté l'algorithme EXP3 et l'algorithme DQL pour allouer les RBs aux stations de base dans une échelle de temps large et aux utilisateurs finaux dans une échelle de temps courte, respectivement.

Dans le chapitre 6, nous avons proposé une approche de découpage du RAN à deux niveaux où les ressources de communication et de calcul sont découpées conjointement et allouées aux équipements finaux. Au premier niveau, chaque station de base alloue un certain nombre de ses RBs radio à chacun de ses appareils finaux associés pour décharger leurs tâches. Au deuxième niveau, un contrôleur central alloue les ressources de calcul requises des serveurs MEC pour calculer les tâches déchargées par les dispositifs finaux. Nous avons développé, pour chaque niveau de découpage du RAN, un algorithme DRL pour allouer de manière appropriée les ressources requises. L'approche proposée est conçue dans une architecture O-RAN, où nous avons décrit comment le non-RT RIC et le near-RT RIC peuvent effectuer les opérations de découpage du RAN.

Les approches de découpage du RAN présentées dans les chapitres 5 et 6 effectuent l'allocation des RBs radio et des ressources de calcul de manière hiérarchique. Ces approches hiérarchiques évitent de confier l'allocation des ressources RAN uniquement à une entité de contrôle centrale, telle que le contrôleur SDN, réduisant ainsi la charge du réseau due à la communication fréquente entre les entités du réseau, telles que les stations de base et le contrôleur SDN.

8.2 Travaux Futurs

Les travaux réalisés dans cette thèse ouvrent la porte à plusieurs directions de recherche pour le futur. Nous citons dans ce qui suit quelques perspectives pour des travaux futurs.

- Dans le chapitre 4, nous avons formulé le problème d'équilibrage de charge sous la forme d'un programme binaire non linéaire considérant le compromis entre un facteur d'équilibrage de charge et le coût des opérations de migration. L'étude d'autres fonctions objectives est, évidemment, une direction de recherche importante et ouverte, soit pour améliorer l'algorithme proposé, soit pour développer de nouveaux algorithmes compétitifs.
- Dans le chapitre 5, nous avons étudié le problème de l'allocation de RBs aux stations de base par le contrôleur SDN, et nous n'avons fourni qu'un algorithme classique d'apprentissage par renforcement, qui est l'algorithme EXP3. Le développement d'algorithmes plus compétitifs pour ce problème est important et reste un problème ouvert.
- Dans le chapitre 5, lorsque nous avons défini le modèle du système, nous avons considéré le modèle de trafic de file d'attente comme un système de file d'attente M/M/1 et supposé que les stations de base aient une connaissance parfaite de l'information sur l'état du canal. La considération des systèmes plus généraux est une direction de recherche potentielle pour proposer des solutions plus générales.
- Dans le chapitre 6, l'approche proposée pour le découpage du RAN ne prend en compte que les exigences du service URLLC. Elle peut donc être améliorée pour prendre en charge d'autres exigences de service, telles que les exigences de service eMBB et mMTC.

LIST OF REFERENCES

[1] 5G Americas. Global statistics. https://www.5gamericas.org/resources/charts-statistics/global, 2021. Accessed: 2021-09-30.

[2] Bugel Jim, John Suja, and Schwartz Stacy. Ericsson mobility report. Technical report, Ericsson, June 2021. https://www.ericsson.com/en/mobility-report/reports.

[3] ITU-R. IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond. Technical report, , Sep 2015. M.2083-0.

[4] Quoc-Viet Pham, Fang Fang, Vu Nguyen Ha, Md Jalil Piran, Mai Le, Long Bao Le, Won-Joo Hwang, and Zhiguo Ding. A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access*, 8:116974–117017, 2020.

[5] Latif U Khan, Ibrar Yaqoob, Nguyen H Tran, Zhu Han, and Choong Seon Hong. Network slicing: Recent advances, taxonomy, requirements, and open research challenges. *IEEE Access*, 8:36009–36028, 2020.

[6] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[7] Yuan Zhang, Lin Cui, Wei Wang, and Yuxiang Zhang. A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications*, 103:101–118, 2018.

[8] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1):333–354, 2017.

[9] Sushant *et al.* Jain. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[10] Opendaylight project. https://www.opendaylight.org/. Accessed: 2021-09-30.

[11] Open Networking Foundation. Open Network Operating System (ONOS). `https://opennetworking.org/onos/`. Accessed: 2021-09-30.

[12] Amin Tootoonchian and Yashar Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, page 3, USA, 2010. USENIX Association.

[13] Teemu *et al.* Koponen. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[14] Open Networking Foundation. OpenFlow Switch Specification. Technical specification, Mar. 2015. Version 1.5.1.

[15] Xenofon *et al.* Foukas. FlexRAN: A flexible and programmable platform for software-defined radio access networks. In *International on Conference on emerging Networking EXperiments and Technologies*, pages 427–441, 2016.

[16] Estefanía Coronado, Shah Nawaz Khan, and Roberto Riggio. 5G-EmPOWER: A software-defined networking platform for 5G radio access networks. *IEEE Transactions on Network and Service Management*, 16(2):715–728, 2019.

[17] O-RAN Alliance. O-RAN: Towards an Open and Smart RAN. Technical report, , Oct. 2018. White Paper.

[18] Leonardo *et al.* Bonati. CellOS: Zero-touch softwarized open cellular networks. *Computer Networks*, 180:107380, 2020.

[19] Arsany Basta, Andreas Blenk, Klaus Hoffmann, Hans Jochen Morper, Marco Hoffmann, and Wolfgang Kellerer. Towards a cost optimal design for a 5G mobile core network based on SDN and NFV. *IEEE Transactions on Network and Service Management*, 14(4):1061–1075, 2017.

[20] Adlen Ksentini, Miloud Bagaa, and Tarik Taleb. On using SDN in 5G: The controller placement problem. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[21] Abdulaziz Abdulghaffar, Ashraf Mahmoud, Marwan Abu-Amara, and Tarek Sheltami. Modeling and evaluation of software defined networking based 5G core network architecture. *IEEE Access*, 9:10179–10198, 2021.

[22] ETSI. Industry Specification Group (ISG) on Multi-access Edge Computing (MEC). `https://www.etsi.org/committee/1425-mec`. Accessed: 2021-09-30.

[23] Davide Borsatti, Gianluca Davoli, Walter Cerroni, and Carla Raffaelli. Enabling Industrial IoT as a Service with Multi-Access Edge Computing. *IEEE Communications Magazine*, 59(8):21–27, 2021.

[24] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. A Survey on Mobile Augmented Reality With 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects. *IEEE Communications Surveys & Tutorials*, 23(2):1160–1192, 2021.

[25] Hongjun Dai, Xiangyu Zeng, Zhilou Yu, and Tingting Wang. A scheduling algorithm for autonomous driving tasks on mobile edge computing servers. *Journal of Systems Architecture*, 94:14–23, 2019.

[26] Alaa Awad Abdellatif, Amr Mohamed, Carla Fabiana Chiasserini, Mounira Tlili, and Aiman Erbad. Edge computing for smart health: Context-aware approaches, opportunities, and challenges. *IEEE Network*, 33(3):196–203, 2019.

[27] Service requirements for the 5G system; Stage 1 (Release 18). Technical Specification Group Services and System Aspects 22.261, 3rd Generation Partnership Project (3GPP), Jun. 2021. Version 18.3.0.

[28] R Meulen. What edge computing means for infrastructure and operations leaders. *Web post on Infrastructure & Operations, Garner*, 2018.

[29] Ahmet Cihat Baktir, Atay Ozgovde, and Cem Ersoy. How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. *IEEE Communications Surveys & Tutorials*, 19(4):2359–2391, 2017.

[30] Partha Pratim Ray and Neeraj Kumar. SDN/NFV architectures for edge-cloud oriented IoT: A systematic review. *Computer Communications*, 2021.

[31] Haixia Peng, Qiang Ye, and Xuemin Sherman Shen. SDN-based resource management for autonomous vehicular networks: A multi-access edge computing approach. *IEEE Wireless Communications*, 26(4):156–162, 2019.

[32] Kuljeet Kaur, Sahil Garg, Gagangeet Singh Aujla, Neeraj Kumar, Joel JPC Rodrigues, and Mohsen Guizani. Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay. *IEEE communications magazine*, 56(2):44–51, 2018.

[33] Prateek Shantharama, Akhilesh S Thyagaturu, Nurullah Karakoc, Lorenzo Ferrari, Martin Reisslein, and Anna Scaglione. LayBack: SDN management of multi-access edge computing (MEC) for network access services and radio resource sharing. *IEEE Access*, 6:57545–57561, 2018.

[34] Wenchao Xia, Jun Zhang, Tony QS Quek, Shi Jin, and Hongbo Zhu. Mobile edge cloud-based industrial internet of things: improving edge intelligence with hierarchical SDN controllers. *IEEE Vehicular Technology Magazine*, 15(1):36–45, 2020.

[35] Haibo Zhang, Zixin Wang, and Kaijian Liu. V2X offloading and resource allocation in SDN-assisted MEC-based vehicular networks. *China Communications*, 17(5):266–283, 2020.

[36] Nahida Kiran, Xuanlin Liu, Sihua Wang, and Changchuan Yin. VNF placement and resource allocation in SDN/NFV-enabled MEC networks. In *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–6. IEEE, 2020.

[37] Kyle E Benson, Guoxi Wang, Nalini Venkatasubramanian, and Young-Jin Kim. Ride: A resilient IoT data exchange middleware leveraging SDN and edge cloud resources. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 72–83. IEEE, 2018.

[38] Di Wu, Xin Huang, Xiaofeng Xie, Xiang Nie, Lichun Bao, and Zhijin Qin. LEDGE: Leveraging edge computing for resilient access management of mobile IoT. *IEEE Transactions on Mobile Computing*, 2019.

[39] Xiangwang Hou, Zhiyuan Ren, Jingjing Wang, Wenchi Cheng, Yong Ren, Kwang-Cheng Chen, and Hailin Zhang. Reliable computation offloading for edge-computing-enabled software-defined IoV. *IEEE Internet of Things Journal*, 7(8):7097–7111, 2020.

[40] Federico Cimorelli, Francesco Delli Priscoli, Antonio Pietrabissa, Lorenzo Ricciardi Celsi, Vincenzo Suraci, and Letterio Zuccaro. A dis-

tributed load balancing algorithm for the control plane in software defined networking. In *2016 24th Mediterranean Conference on Control and Automation (MED)*, pages 1033–1040. IEEE, 2016.

[41] Ping Song, Yi Liu, Tianxiao Liu, and Depei Qian. Flow Stealer: lightweight load balancing by stealing flows in distributed SDN controllers. *Science China Information Sciences*, 60(3):032202, 2017.

[42] Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ramana Rao Kompella. ElastiCon; an elastic distributed SDN controller. In *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 17–27. IEEE, 2014.

[43] Chuan'an Wang, Bo Hu, Shanzhi Chen, Desheng Li, and Bin Liu. A switch migration-based decision-making scheme for balancing load in SDN. *IEEE Access*, 5:4537–4544, 2017.

[44] Rajat Chaudhary and Neeraj Kumar. LOADS: Load optimization and anomaly detection scheme for software-defined networks. *IEEE Transactions on Vehicular Technology*, 68(12):12329–12344, 2019.

[45] Tao Hu, Julong Lan, Jianhui Zhang, and Wei Zhao. EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking. *Peer-to-Peer networking and applications*, 12(2):452–464, 2019.

[46] Faroq Al-Tam and Noélia Correia. On load balancing via switch migration in software-defined networking. *IEEE Access*, 7:95998–96010, 2019.

[47] Kshira Sagar Sahoo, Deepak Puthal, Mayank Tiwary, Muhammad Usman, Bibhudatta Sahoo, Zhenyu Wen, Biswa PS Sahoo, and Rajiv Ranjan. ESMLB: Efficient switch migration-based load balancing for multicontroller SDN in IoT. *IEEE Internet of Things Journal*, 7(7):5852–5860, 2019.

[48] Ziyong Li, Yuxiang Hu, Tao Hu, and Peng Wei. Dynamic SDN controller association mechanism based on flow characteristics. *IEEE Access*, 7:92661–92671, 2019.

[49] Penghao Sun, Zehua Guo, Gang Wang, Julong Lan, and Yuxiang Hu. MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning. *Computer Networks*, 177:107230, 2020.

[50] Jie Cui, Qinghe Lu, Hong Zhong, Miaomiao Tian, and Lu Liu. A load-balancing mechanism for distributed SDN control plane using response time. *IEEE transactions on network and service management*, 15(4):1197–1206, 2018.

[51] Tao Wang, Fangming Liu, and Hong Xu. An efficient online algorithm for dynamic SDN controller assignment in data center networks. *IEEE/ACM Transactions on Networking*, 25(5):2788–2801, 2017.

[52] Abderrahime Filali, Abdellatif Kobbane, Mouna Elmachkour, and Soumaya Cherkaoui. SDN controller assignment and load balancing with minimum quota of processing capacity. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[53] 3GPP. Study on Enhanced Access to and Support of Network Slice (Release 18). Technical Specification (TS) 28.835, 3rd Generation Partnership Project (3GPP), June 2021. Version 18.1.0.

[54] 3GPP. Study on enhancement of network slicing Phase 2 (Release 17). Technical Specification (TS) 28.835, 3rd Generation Partnership Project (3GPP), Mar. 2021. Version 17.0.0.

[55] 3GPP. Study on enhancement of Radio Access Network (RAN) slicing (Release 17). Technical Specification (TS) 28.835, 3rd Generation Partnership Project (3GPP), June 2021. Version 17.0.0.

[56] GSMA. E2E Network Slicing Architecture. White Paper NG.127, Global System for Mobile Communications, June 2021. Version 1.0.

[57] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018.

[58] Salah Eddine Elayoubi, Sana Ben Jemaa, Zwi Altman, and Ana Galindo-Serrano. 5G RAN slicing for verticals: Enablers and challenges. *IEEE Communications Magazine*, 57(1):28–34, 2019.

[59] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.

[60] ETSI. 5G; Management and orchestration; Concepts, use cases and requirements (Release 16). Technical Specification (TS) 128.530, European Telecommunications Standards Institute (ETSI), Jan. 2021. Version 16.4.0.

[61] Mu Yan, Gang Feng, Jianhong Zhou, Yao Sun, and Ying-Chang Liang. Intelligent resource scheduling for 5G radio access network slicing. *IEEE Transactions on Vehicular Technology*, 68(8):7691–7703, 2019.

[62] Guolin Sun, Kun Xiong, Gordon Owusu Boateng, Guisong Liu, and Wei Jiang. Resource slicing and customization in RAN with dueling deep Q-Network. *Journal of Network and Computer Applications*, 157:102573, 2020.

[63] Qiang Ye, Weihua Zhuang, Shan Zhang, A-Long Jin, Xuemin Shen, and Xu Li. Dynamic radio resource slicing for a two-tier heterogeneous wireless network. *IEEE Transactions on Vehicular Technology*, 67(10):9896–9910, 2018.

[64] Junling Li, Weisen Shi, Peng Yang, Qiang Ye, Xuemin Sherman Shen, Xu Li, and Jaya Rao. A hierarchical soft RAN slicing framework for differentiated service provisioning. *IEEE Wireless Communications*, 27(6):90–97, 2020.

[65] Sunday Oladayo Oladejo and Olabisi Emmanuel Falowo. Latency-aware dynamic resource allocation scheme for multi-tier 5G network: A network slicing-multitenancy scenario. *IEEE Access*, 8:74834–74852, 2020.

[66] Wen Wu, Nan Chen, Conghao Zhou, Mushu Li, Xuemin Shen, Weihua Zhuang, and Xu Li. Dynamic RAN Slicing for Service-Oriented Vehicular Networks via Constrained Learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[67] Xianfu Chen, Zhifeng Zhao, Celimuge Wu, Mehdi Bennis, Hang Liu, Yusheng Ji, and Honggang Zhang. Multi-tenant cross-slice resource orchestration: A deep reinforcement learning approach. *IEEE Journal on Selected Areas in Communications*, 37(10):2377–2392, 2019.

[68] Qiang Ye, Weisen Shi, Kaige Qu, Hongli He, Weihua Zhuang, and Xuemin Shen. Joint RAN slicing and computation offloading for autonomous vehicular networks: A learning-assisted hierarchical approach. *IEEE Open Journal of Vehicular Technology*, 2:272–288, 2021.

[69] Yin Ren, Aihuang Guo, Chunlin Song, and Yidan Xing. Dynamic Resource Allocation Scheme and Deep Deterministic Policy Gradient-Based Mobile Edge Computing Slices System. *IEEE Access*, pages 86062–86073, 2021.

[70] Pengfei Zhu, Jiawei Zhang, Yuming Xiao, Jiabin Cui, Lin Bai, and Yuefeng Ji. Deep reinforcement learning-based radio function deployment for secure and resource-efficient NG-RAN slicing. *Engineering Applications of Artificial Intelligence*, 106:104490, 2021.

[71] Tao Hu, Zehua Guo, Peng Yi, Thar Baker, and Julong Lan. Multi-controller based software-defined networking: A survey. *IEEE Access*, 6:15980–15996, 2018.

[72] Meysam Azizian, Soumaya Cherkaoui, and Abdelhakim Senhaji Hafid. Vehicle software updates distribution with SDN and cloud computing. *IEEE Communications Magazine*, 55(8):74–79, 2017.

[73] Zainab Zaidi, Vasilis Friderikos, Zarrar Yousaf, Simon Fletcher, Mischa Dohler, and Hamid Aghvami. Will SDN be part of 5G? *IEEE Communications Surveys & Tutorials*, 20(4):3220–3258, 2018.

[74] Guozhen Cheng, Hongchang Chen, Zhiming Wang, and Shuqiao Chen. DHA: Distributed decisions on the switch migration toward a scalable SDN control plane. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2015.

[75] Hongchang Chen, Guozhen Cheng, and Zhiming Wang. A game-theoretic approach to elastic control in software-defined networking. *China Communications*, 13(5):103–109, 2016.

[76] Marco Cello, Yang Xu, Anwar Walid, Gordon Wilfong, H Jonathan Chao, and Mario Marchese. BalCon: A distributed elastic SDN control via efficient switch migration. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 40–50. IEEE, 2017.

[77] Long Yao, Peilin Hong, Wen Zhang, Jianfei Li, and Dan Ni. Controller placement and flow based dynamic management problem towards SDN. In *2015 IEEE International Conference on Communication Workshop (ICCW)*, pages 363–368. IEEE, 2015.

[78] Tao Wang, Fangming Liu, Jian Guo, and Hong Xu. Dynamic SDN controller assignment in data center networks: Stable matching with trans-

fers. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[79] Xiaofeng Gao, Linghe Kong, Weichen Li, Wanchao Liang, Yuxiang Chen, and Guihai Chen. Traffic Load Balancing Schemes for Devolved Controllers in Mega Data Centers. *IEEE Transactions on Parallel & Distributed Systems*, 28(02):572–585, 2017.

[80] Yuanhao Zhou, Mingfa Zhu, Limin Xiao, Li Ruan, Wenbo Duan, Deguo Li, Rui Liu, and Mingming Zhu. A load balancing strategy of sdn controller based on distributed decision. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 851–856. IEEE, 2014.

[81] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[82] Qingjia Huang, Kai Shuang, Peng Xu, Jian Li, Xu Liu, and Sen Su. Prediction-based dynamic resource scheduling for virtualized cloud systems. *Journal of Networks*, 9(2):375, 2014.

[83] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280, 2010.

[84] Murat Karakus and Arjan Durresi. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks*, 112:279–293, 2017.

[85] Abderrahime Filali, Amine Abouaomar, Soumaya Cherkaoui, Abdellatif Kobbane, and Mohsen Guizani. Multi-access edge computing: A survey. *IEEE Access*, 8:197017–197046, 2020.

[86] Catherine Nayer Tadros, Mohamed RM Rizk, and Bassem Mahmoud Mokhtar. Software defined network-based management for enhanced 5G network services. *IEEE Access*, 8:53997–54008, 2020.

[87] Chuan Lin, Guangjie Han, Xingyue Qi, Mohsen Guizani, and Lei Shu. A distributed mobile fog computing scheme for mobile delay-sensitive applications in SDN-enabled vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(5):5481–5493, 2020.

[88] Meysam Azizian, Soumaya Cherkaoui, and Abdelhakim Hafid. An optimized flow allocation in vehicular cloud. *IEEE Access*, 4:6766–6779, 2016.

[89] Ali Akbar Neghabi, Nima Jafari Navimipour, Mehdi Hosseinzadeh, and Ali Rezaee. Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature. *IEEE Access*, 6:14159–14178, 2018.

[90] Guowei Wu, Jinlei Wang, Mohammad S Obaidat, Lin Yao, and Kuei-Fang Hsiao. Dynamic switch migration with noncooperative game towards control plane scalability in SDN. *International Journal of Communication Systems*, 32(7):e3927, 2019.

[91] Ali El Kamel and Habib Youssef. Improving switch-to-controller assignment with load balancing in multi-controller software defined WAN (SD-WAN). *Journal of Network and Systems Management*, pages 1–23, 2020.

[92] Noelia Correia and AL-Tam Faroq. Flow setup aware controller placement in distributed software-defined networking. *IEEE Systems Journal*, 14(4):5096–5099, 2019.

[93] GM Jenkins. Autoregressive–Integrated Moving Average (ARIMA) Models. *Wiley StatsRef: Statistics Reference Online*, 2014.

[94] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[95] Martin Nowak and Karl Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game. *Nature*, 364(6432):56–58, 1993.

[96] Zoubeir Mlika, Elmahdi Driouch, and Wessam Ajib. A fully distributed algorithm for user-base station association in HetNets. *Computer Communications*, 105:66–78, 2017.

[97] Yang Zhou, Kangfeng Zheng, Wei Ni, and Ren Ping Liu. Elastic switch migration for control plane load balancing in SDN. *IEEE Access*, 6:3909–3919, 2018.

[98] Mateus AS Santos, Bruno AA Nunes, Katia Obraczka, Thierry Turletti, Bruno T De Oliveira, and Cintia B Margi. Decentralizing SDN's control plane. In *39th Annual IEEE Conference on Local Computer Networks*, pages 402–405. IEEE, 2014.

[99] Abderrahime Filali, Soumaya Cherkaoui, and Abdellatif Kobbane. Prediction-based switch migration scheduling for SDN load balancing. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[100] Afaf Taïk and Soumaya Cherkaoui. Electrical load forecasting using edge computing and federated learning. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[101] Telephone Corporation Nippon Telegraph and. RYU the Network Operating System(NOS) — Ryu 4.34 documentation. https://ryu.readthedocs.io/en/latest/.

[102] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, and Mohsen Guizani. SDN controllers: Benchmarking & performance evaluation. *arXiv preprint arXiv:1902.04491*, 2019.

[103] Danny Yuxing Huang, Kenneth Yocum, and Alex C Snoeren. High-fidelity switch models for software-defined network emulation. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 43–48, 2013.

[104] Amine Abouaomar, Soumaya Cherkaoui, Abdellatif Kobbane, and Oussama Abderrahmane Dambri. A resources representation for resource allocation in fog computing networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[105] Gerhard J Woeginger and Zhongliang Yu. On the equal-subset-sum problem. *Information Processing Letters*, 42(6):299–302, 1992.

[106] Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

[107] Anutusha Dogra, Rakesh Kumar Jha, and Shubha Jain. A survey on beyond 5G network with the advent of 6G: Architecture and emerging technologies. *IEEE Access*, 9:67512–67547, 2020.

[108] Zoubeir Mlika and Soumaya Cherkaoui. Massive IoT Access with NOMA in 5G Networks and Beyond using Online Competitiveness and Learning. *IEEE Internet of Things Journal*, 2021.

[109] Boubakr Nour and Soumaya Cherkaoui. A Network-based Compute Reuse Architecture for IoT Applications. *arXiv preprint arXiv:2104.03818*, 2021.

[110] Zhaogang Shu and Tarik Taleb. A novel QoS framework for network slicing in 5G and beyond networks based on SDN and NFV. *IEEE Network*, 34(3):256–263, 2020.

[111] Alex Reznik, Luis Miguel Contreras Murillo, Yonggang Fang, Walter Featherstone, Miltiadis Filippou, Francisco Fontes, Fabio Giust, Qiang Huang, Alice Li, Charles Turyagyenda, et al. Cloud RAN and MEC: A perfect pairing. *ETSI White paper*, (23):1–24, February 2018.

[112] Kai Liang, Liqiang Zhao, Xiaoli Chu, and Hsiao-Hwa Chen. An integrated architecture for software defined and virtualized radio access networks with fog computing. *IEEE Network*, 31(1):80–87, 2017.

[113] Yu Abiko, Takato Saito, Daizo Ikeda, Ken Ohta, Tadanori Mizuno, and Hiroshi Mineno. Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning. *IEEE Access*, 8:68183–68198, 2020.

[114] Rui Dong, Changyang She, Wibowo Hardjawana, Yonghui Li, and Branka Vucetic. Deep learning for radio resource allocation with diverse quality-of-service requirements in 5G. *IEEE Transactions on Wireless Communications*, 20(4):2309–2324, 2020.

[115] Guolin Sun, Zemuy Tesfay Gebrekidan, Gordon Owusu Boateng, Daniel Ayepah-Mensah, and Wei Jiang. Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks. *IEEE Access*, 7:45758–45772, 2019.

[116] Yao Sun, Shuang Qin, Gang Feng, Lei Zhang, and Muhammad Imran. Service provisioning framework for RAN slicing: user admissibility, slice association and bandwidth allocation. *IEEE Transactions on Mobile Computing*, 2020.

[117] Praveenkumar Korrai, Eva Lagunas, Shree Krishna Sharma, Symeon Chatzinotas, Ashok Bandi, and Björn Ottersten. A RAN resource slicing mechanism for multiplexing of eMBB and URLLC services in OFDMA based 5G wireless networks. *IEEE Access*, 8:45674–45688, 2020.

[118] 3GPP. 5GM; NR; Physical channels and modulation. Technical Specification (TS) 38.211, 3rd Generation Partnership Project (3GPP), 2019.

[119] Petar Popovski, Kasper Fløe Trillingsgaard, Osvaldo Simeone, and Giuseppe Durisi. 5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view. *IEEE Access*, 6:55765–55779, September 2018.

[120] Eduardo Noboro Tominaga, Hirley Alves, Onel L Alcaraz López, Richard Demo Souza, João Luiz Rebelatto, and Matti Latva-Aho. Network Slicing for eMBB and mMTC with NOMA and Space Diversity Reception. In *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, pages 1–6, April 2021.

[121] Changqing Luo, Jinlong Ji, Qianlong Wang, Xuhui Chen, and Pan Li. Channel state information prediction for 5G wireless communications: A deep learning approach. *IEEE Transactions on Network Science and Engineering*, 7(1):227–236, 2018.

[122] Abderrahime Filali, Zoubeir Mlika, Soumaya Cherkaoui, and Abdellatif Kobbane. Preemptive SDN load balancing with machine learning for delay sensitive applications. *IEEE Transactions on Vehicular Technology*, 69(12):15947–15963, 2020.

[123] Farhad Rezazadeh, Hatim Chergui, and Christos Verikoukis. Zero-touch Continuous Network Slicing Control via Scalable Actor-Critic Learning. *arXiv preprint arXiv:2101.06654*, 2021.

[124] Mojdeh Karbalaee Motalleb, Vahid Shah-Mansouri, and Salar Nouri Naghadeh. Joint power allocation and network slicing in an open RAN system. *arXiv preprint arXiv:1911.01904*, 2019.

[125] Qianyu Xu, Suoping Li, Tien Van, Kejun Jia, and Nana Yang. Performance Analysis of Cognitive Radio Networks with Burst Dynamics. *IEEE Access*, 2021.

[126] Nikolaos Pappas. Performance Analysis of a System with Bursty Traffic and Adjustable Transmission Times. In *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–6. IEEE, 2018.

[127] John F Shortle, James M Thompson, Donald Gross, and Carl M Harris. *Fundamentals of queueing theory*, volume 399. John Wiley & Sons, 2018.

[128] David Pisinger and Paolo Toth. *Knapsack Problems*, pages 299–428. Springer US, Boston, MA, 1998.

[129] Jingjing Wang, Chunxiao Jiang, Haijun Zhang, Yong Ren, Kwang-Cheng Chen, and Lajos Hanzo. Thirty years of machine learning: The road to Pareto-optimal wireless networks. *IEEE Communications Surveys & Tutorials*, 22(3):1472–1514, 2020.

[130] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The Nonstochastic Multiarmed Bandit Problem. *SIAM J. Comput.*, 32(1):48–77, January 2003.

[131] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[132] Richard E. Bellman. *Dynamic Programming*. July 2010.

[133] Qingchen Zhang, Man Lin, Laurence T Yang, Zhikui Chen, and Peng Li. Energy-efficient scheduling for real-time systems based on deep Q-learning model. *IEEE Transactions on Sustainable Computing*, 4(1):132–141, 2019.

[134] Yueyue Dai, Du Xu, Sabita Maharjan, Guanhua Qiao, and Yan Zhang. Artificial intelligence empowered edge computing and caching for internet of vehicles. *IEEE Wireless Communications*, 26(3):12–18, 2019.

[135] Sheyda Zarandi and Hina Tabassum. Delay Minimization in Sliced Multi-Cell Mobile Edge Computing (MEC) Systems. *arXiv preprint arXiv:2101.03405*, 2021.

[136] Xenofon Foukas, Mahesh K Marina, and Kimon Kontovasilis. Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture. In *Proceedings of the 23rd annual international conference on mobile computing and networking*, pages 127–140, October 2017.

[137] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[138] Zoubeir Mlika and Soumaya Cherkaoui. Network slicing with mec and deep reinforcement learning for the internet of vehicles. *IEEE Network*, 35(3):132–138, 2021.

[139] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, March 2016.

[140] Fengsheng Wei, Gang Feng, Yao Sun, Yatong Wang, Shuang Qin, and Ying-Chang Liang. Network Slice Reconfiguration by Exploiting Deep Reinforcement Learning With Large Action Space. *IEEE Transactions on Network and Service Management*, 17(4):2197–2211, August 2020.

[141] Amandeep Kaur and Krishan Kumar. Imperfect CSI based Intelligent Dynamic Spectrum Management using Cooperative Reinforcement Learning Framework in Cognitive Radio Networks. *IEEE Transactions on Mobile Computing*, 2020.

[142] Sonia Shahzadi, Muddesar Iqbal, and Nauman Riaz Chaudhry. 6G Vision: Toward Future Collaborative Cognitive Communication (3C) Systems. *IEEE Communications Standards Magazine*, 5(2):60–67, 2021.

[143] Peng Yang, Xing Xi, Tony QS Quek, Jingxuan Chen, Xianbin Cao, and Dapeng Wu. RAN slicing for massive IoT and bursty URLLC service multiplexing: Analysis and optimization. *IEEE Internet of Things Journal*, 2021.

[144] Ruoyu Su, Dengyin Zhang, Ramachandran Venkatesan, Zijun Gong, Cheng Li, Fei Ding, Fan Jiang, and Ziyang Zhu. Resource allocation for network slicing in 5G telecommunication networks: A survey of principles and models. *IEEE Network*, 33(6):172–179, 2019.

[145] Leonardo Bonati, Salvatore D'Oro, Michele Polese, Stefano Basagni, and Tommaso Melodia. Intelligence and Learning in O-RAN for Data-driven NextG Cellular Networks. *arXiv preprint arXiv:2012.01263*, 2020.

[146] 3GPP. Management and orchestration Concepts, use cases and requirements (Release 17). Technical Specification (TS) 28.530, 3rd Generation Partnership Project (3GPP), Mar. 2021. Version 17.1.0.

[147] 3GPP. Study on enhancement of Radio Access Network (RAN) slicing (Release 17). Technical Specification (TS) Group RAN 38.832, 3rd Generation Partnership Project (3GPP), Mar. 2021. Version 1.0.0.

[148] Chia-Yu Chang, Navid Nikaein, and Thrasyvoulos Spyropoulos. Radio access network resource slicing for flexible service execution. In *IEEE*

Conference on Computer Communications Workshops, pages 668–673. IEEE, 2018.

[149] Qiang Liu, Tao Han, and Ephraim Moges. EdgeSlice: Slicing Wireless Edge Computing Network with Decentralized Deep Reinforcement Learning. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 234–244, 2020.

[150] Solmaz Niknam, Abhishek Roy, Harpreet S Dhillon, Sukhdeep Singh, Rahul Banerji, Jeffery H Reed, Navrati Saxena, and Seungil Yoon. Intelligent O-RAN for beyond 5G and 6G wireless networks. *arXiv preprint arXiv:2005.08374*, 2020.