Universidade Estadual de Campinas
Instituto de Computação

William Fernando Villota Jácome

Admission Control and Resource Allocation in 5G Network Slicing

Controle de Admissão e Alocação de Recursos em Fatiamento de Rede 5G

CAMPINAS
2020

William Fernando Villota Jácome

Admission Control and Resource Allocation in 5G Network Slicing

Controle de Admissão e Alocação de Recursos em Fatiamento de Rede 5G

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Nelson Luis Saldanha da Fonseca
Co-supervisor/Coorientador: Prof. Dr. Oscar Mauricio Caicedo Rendon

Este exemplar corresponde à versão final da Dissertação defendida por William Fernando Villota Jácome e orientada pelo Prof. Dr. Nelson Luis Saldanha da Fonseca.

CAMPINAS
2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

V719a

Villota Jácome, William Fernando, 1992-
Admission control and resource allocation in 5G network slicing / William Fernando Villota Jácome. – Campinas, SP : [s.n.], 2020.

Orientador: Nelson Luis Saldanha da Fonseca.
Coorientador: Oscar Mauricio Caicedo Rendon.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Fatiamento da rede. 2. Aprendizado de máquina. 3. Sistemas de comunicação móvel 5G. I. Fonseca, Nelson Luis Saldanha da, 1961-. II. Caicedo Rendon, Oscar Mauricio. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Controle de admissão e alocação de recursos em fatiamento da rede 5G
Palavras-chave em inglês:
Network slicing
Machine learning
5G mobile communication systems
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Nelson Luis Saldanha da Fonseca [Orientador]
Edmundo Roberto Mauro Madeira
Christian Rodolfo Esteve Rothenberg
Data de defesa: 07-10-2020
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-5869-6181
- Currículo Lattes do autor: http://lattes.cnpq.br/1909464409030607

Universidade Estadual de Campinas
Instituto de Computação

William Fernando Villota Jácome

Admission Control and Resource Allocation in 5G Network Slicing

Controle de Admissão e Alocação de Recursos em Fatiamento de Rede 5G

Banca Examinadora:

- Prof. Dr. Nelson Luis Saldanha da Fonseca
  IC - UNICAMP
- Prof. Dr. Edmundo Roberto Mauro Madeira
  IC - UNICAMP
- Prof. Dr. Christian Rodolfo Esteve Rothenberg
  FEEC - UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 07 de outubro de 2020

Acknowledgements

I would like to thank life for my being in this epoch and this place. To my supervisor Prof. Nelson Luis Saldanha da Fonseca and my co-supervisor Prof. Oscar Mauricio Caicedo Rendon, for expertly guiding me through this path. To my parents, Amanda and Edison, and my brothers, Gabriel and Cristian, for their sincere and unconditional support. To my friends and laboratory colleagues at the LRC. To all professors and workers at the IC for their essential labor. This work was carried out with the support from CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) - Brazil.

Resumo

A infraestrutura dos provedores de serviços de redes 5G receberá requisições para implementação de fatias de rede (do Inglês, *network slices*) solicitadas por usuários com diferentes requisitos de qualidade de serviço (*QoS*). Considerando que os recursos no substrato da rede são finitos e os casos de uso de 5G têm requisitos de QoS heterogêneos, bem como custos de implantação específicos, os provedores de serviço precisam gerenciar a admissão dessas requisições, bem como alocar recursos para que a utilização destes seja eficiente.

Diferentes abordagens lidam com o controle de admissão e alocação de recursos em redes 5G utilizando diferentes referenciais teóricos tais como Teoria de Filas, Teoria de Redes Complexas, e Otimização. No entanto, essas abordagens propõem a tomada de decisões de admissão considerando requisições individuais, o que pode levar a decisões sub-ótimas, uma vez que as requisições mais lucrativas que chegam em futuro breve após a admissão de uma requisição podem ser rejeitadas devido à indisponibilidade de recursos recentemente alocados. Além disso, a maioria dessas abordagens não considera os requisitos específicos de QoS de cada caso de uso da especificação de redes 5G, nem a alocação de recursos em nós do núcleo e da borda da rede. Diversas outras soluções consideram apenas a alocação de recursos sem considerar o controle de admissão, ignorando, assim, interesses múltiplos dos provedores.

Nesta tese, são propostas duas soluções para realização conjunta do controle de admissão e da alocação de recursos para o fatiamento de redes 5G. A primeira solução é baseada em Aprendizado por Reforço, que permite o aprendizagem da admissão de requisições de fatia de rede visando o lucro dos provedores. A segunda solução, baseada em Aprendizado por Reforço Profundo, almeja aperfeiçoar ainda mais o alcance dos objetivos dos provedores. A alocação de recursos nessas soluções é realizada por um mapeamento de nós virtuais nos nós físicos da rede seguido de um mapeamento dos enlaces virtuais em enlaces físicos. Consideram-se, nesses mapeamentos, os requisitos de QoS das classes de serviço eMBB, URLLC e MIoT da tecnologia 5G. As soluções propostas foram avaliadas para diferentes condições de tráfego e topologias de redes. Os resultados da avaliação corroboram que as propostas produzem melhores resultados do que as heurísticas *Always Admit Requests* e *Node Ranking* tendo como parâmetro de comparação o lucro e a utilização de recursos. Resultados evidenciam a efetividade do uso de técnicas de Aprendizado por Reforço e Aprendizado por Reforço Profundo para o gerenciamento de requisições de fatia de rede em redes 5G.

Abstract

5G Network Service Providers will receive myriads of network slice requests generated by multiple tenants. Considering that the resources in the network substrate are finite and 5G use cases have particular QoS requirements as well as different deployment costs, providers need to control the admission and allocate resources efficiently for such slice requests.

Several approaches have addressed admission control and resource allocations in 5G by different techniques such as Queuing Theory, Complex Network Theory, Big Data, Heuristics, Integer Linear Programming, Reinforcement Learning, and recently, Deep Reinforcement Learning. Nevertheless, the approaches mentioned above propose making admission decisions considering individual requests, which can lead to sub-optimal decisions since more profitable requests arriving in the short term can be rejected due to the unavailability of resources recently allocated. Moreover, most of these proposals neither consider the particularities of the QoS requirements of different service types (use cases) nor the allocation of resources in 5G core and edge nodes. Several other solutions based on heuristics, Queuing Theory, and Complex Network Theory have considered only resource allocation and neglected admission control, which prevents the achievement of the provider's goals.

In this thesis, we propose two solutions to jointly perform admission control and resource allocation in 5G Network Slicing. The first solution is based on Reinforcement Learning, which allows learning to admit 5G network slice requests in such a way that optimizes the profit to service providers. The second solution is based on Deep Reinforcement Learning aimed at further optimizing the proposed objective. Resource allocation in both solutions is carried out by node mapping and link mapping steps that assign substrate network resources to requests while accomplishing their QoS requirements according to the 5G use case (*i.e.,* eMBB, URLLC, and MIoT) to which they belong. Our solutions are assessed for different requests arrival rates and on topologies of distinct sizes. The evaluation results corroborate that we outperform the heuristics Always Admit Requests and Node Ranking regarding mean profit and overall resource utilization, and demonstrate the convenience of using Reinforcement Learning and Deep Reinforcement Learning to manage the admission of network slice requests in 5G.

List of Figures

List of Tables

List of Algorithms

Acronyms

AAR	Always Admit Requests
AC	Admission Control
ACM	Admission Control Module
AMF	Access and Mobility Function
CP	Control Plane
DQN	Deep Q-Learning
DRL	Deep Reinforcement Learning
$DSARA$	DRL-based SARA
$eMBB$	Enhanced Mobile Broad Band
MDP	Markov Decision Process
$MIoT$	Massive IoT
ML	Machine Learning
NF	Network Function
NFV	Network Function Virtualization
NN	Artificial Neural Network
NR	Node Ranking
NSL	Network Slice
$NSLR$	Network Slice Request
NSP	5G Network Service Provider
QoS	Quality of Service
RA	Resource Allocation
RAM	Resource Allocation Module
RAN	Radio Access Network
RL	Reinforcement Learning

$SARA$	network Slice requests Admission and Resource Allocation
SBA	Service Based Architecture
SDN	Software-Defined Networking
SMF	Session Management Function
UP	User Plane
UPF	User Plane Function
$URLLC$	Ultra Reliable Low Latency Communications
VNF	Virtual Network Function

Contents

Chapter 1

Introduction

The Fifth Generation of Cellular Mobile Communications (5G) is envisioned to be the key enabler technology that will support the provisioning of myriads of future specific services reaching a large number of devices. These on-demand services will be provided to several customers simultaneously [2], each having different high Quality of Service (QoS) requirements in terms of bandwidth, latency, and coverage [54]. 5G opens opportunities for different vertical industries (*e.g.*, automotive, healthcare, entertainment, energy, etc.) to offer innovative and specialized services with a broader range of requirements than the current ones. To meet these new requirements and hence, accomplish the provisioning of such specialized services, 5G demands important characteristics like flexibility and modularity. Such characteristics may be exploited when using Network Slicing.

Network Slicing can be related to the concept of network softwarization, which involves Software-defined Networking (SDN) and Network Function Virtualization (NFV). Network softwarization provides programmability, flexibility, and modularity to build and run several Network Slices (NSLs) over a single network infrastructure, each of them customized to accomplish particular needs (*i.e.*, Service Level Agreements (SLAs) requirements) [46]. An NSL is a set of Virtual Network Functions (VNF) created on-demand following different principles like isolation, automation, customization, elasticity, programmability, end-to-end service delivery, and hierarchical abstraction [2].

The slicing process starts when Network Slice Providers (NSPs) receives several NSL requests (NSLRs) from multiple tenants. These NSLRs may belong to one of three types: Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communication (URLLC), or Massive IoT (MIoT). As each NSLR has particular QoS requirements and the substrate resources are finite, NSPs face the challenge of controlling the admission of NSLRs to increase its overall profit and improve network resource utilization. Coping with NSLRs is a twofold challenge that involves both admission control and resource allocation. The former is a verifying and decision process intended to permit the access or restrict the admission to a system (e.g., substrate network) considering one or more criteria like profit [48]. The latter can be faced as a Virtual Network Embedding (VNE) problem that maps virtual networks onto a physical network [35, 57].

Performing admission control and resource allocation jointly and intelligently in 5G core network slicing is critical to optimize resource utilization and maximize the NSP profit. In recent years, Machine Learning (ML) techniques have been remarkably useful

in several fields through Supervised Learning, Unsupervised Learning, and Reinforcement Learning (RL). RL and the recent Deep Reinforcement Learning (DRL) are efficient tools for solving decision-making problems modeled as Markov Decision Processes. Since no data is available a priori about the 5G Network Slicing dynamic, RL and DRL arise as the appropriate tools to learn from the instantaneous information generated in the slicing process. Network Slicing involves repetitive decisions meaning it produces a large quantity of data that can be used to train RL/DRL-algorithms [43, 40].

In this thesis, we propose two solutions that leverage ML techniques, in particular RL and DRL, to perform admission control and resource allocation jointly for 5G NSLRs. First, we introduce SARA (network **S**lice requests **A**dmission and **R**esource **A**llocation), a mechanism that manages the admission of 5G NSLRs and allocates substrate resources to them, aiming at optimizing the NSP profit. SARA processes NSLRs on batches collected on time windows, which favors profit maximization. The admission control algorithm of SARA uses an RL-agent to learn the NSLRs that increases the NSP profit. The RL-agent learns from the rewards produced by its iterative interactions with the environment (the substrate network). The resource allocation algorithm of SARA includes node mapping and link mapping steps to allocate resources in the substrate network to VNFs and virtual links composing an NSLR. Substrate resources are allocated by considering the service type (*i.e.,* eMBB, URLLC, and MIoT) to which NSLRs belong and differentiating core nodes from edge nodes for accomplishing QoS requirements.

Second, we propose DSARA, an implementation of SARA based on DRL. The use of DRL is motivated by the need for enhancing and extending SARA to cope with large scenarios in which its convergence time is longer. In DSARA, the DRL-agent approximates the admission policy function, which enables learning the most profitable NSLRs from a reduced number of interactions with the environment. This is known as generalization; the knowledge learned from past interactions is applied to similar situations. As a result, DSARA increases further the profit of NSPs while achieving fast convergence.

We assessed SARA and DSARA extensively in terms of profit, resource utilization, and acceptance ratio for different arrival rates of NSLRs. The evaluation of our solutions includes experiments on three topologies of different sizes: 16, 32, and 64 nodes. Evaluations results corroborate that SARA and DSARA outperform two heuristics, Always Admit Requests and Node Ranking, which demonstrates the convenience of leveraging RL and DRL for network management tasks such as admission control of NSLRs.

The remainder of this chapter summarizes the main contributions of this work (Section 1.1), the publication of results (Section 1.2), and the structure of this document (Section1.3).

1.1 Contributions

The main contributions of this thesis are:

- A solution that uses ML techniques to perform admission control and resource allocation jointly for 5G slice requests while guaranteeing the QoS of each request type.

- An RL-based algorithm for admission control of 5G NSLRs that increases the profit of service providers and network resource utilization.
- A DRL-based algorithm for admission control of 5G NSLRs to further improve the profit of providers in large scenarios while obtaining a fast convergence.
- An extensive performance evaluation of the proposed solutions in terms of profit, resource utilization, and acceptance ratio. The evaluation includes results for network topologies of 16, 32, and 64 nodes under different loads.

1.2 Publications

- Villota-Jácome W. F., Caicedo O. M., Fonseca, N.L.S. Admission Control for 5G Network Slicing based on Reinforcement Learning. Submitted to IEEE Transactions on Network and Service Management (TNSM).

1.3 Thesis Structure

This document is organized as follows. Chapter 2 reviews the main concepts of 5G, Network Slicing, and ML. Chapter 3 introduces the proposed RL-based admission control and resource allocation solution for 5G network slices. Chapter 4 presents the DRL-based approach to further improve the performance of our RL-based solution. Finally, chapter 5 provides conclusions and research directions.

Chapter 2

Background

In this chapter, we overview the concepts related to this thesis as follows. Section 2.1 presents the 5G, its features, use case families, and architecture. Section 2.2 introduces the notion of Network Slicing as well as its characteristics and components. Section 2.3 explores the main concepts in Machine Learning.

2.1 5G

5G is considered to be the generation of mobile networks that can support a wide range of devices and specific types of services to satisfy various customer demands simultaneously [2]. The services offered by 5G have different requirements, such as high data traffic volumes and a large number of devices. 5G provides such services by leveraging important advantages in terms of enhanced bandwidth, reduced latency, and extended coverage [54]. In order to support 5G specialized services and enable collaboration between industries and academia, eight major requirements are identified [4]:

- 1 - 10 Gbps data rates in real networks, which means a 10 times increment from LTE's data rate.
- 1 ms end-to-end round trip latency.
- Large number of connected devices with higher bandwidth for longer duration in a specific area.
- Provide connectivity to an enormous number (*i.e.,* thousands) of devices to realize the vision of IoT.
- Perceived network availability of 99.999%, this is practically always available.
- Almost 100% coverage. 5G need to ensure complete coverage independently of user's location.
- Almost 90% of reduction in energy usage. This is a remarkable reduction considering high data rates and massive connectivity.
- High battery life.

2.1.1 Use Cases

Several use cases are emerging for 5G, aiming at providing an end-to-end ecosystem with a fully mobile experience [2]. The Next Generation Mobile Networks (NGMN) describes eight use case families that can be used to define the requirements and building blocks of the 5G architecture [7]:

- Massive Internet of Things: provisioning of broadband for a massive number of devices (*e.g.*, sensors, actuators) with different characteristics and therefore demands. For example, myriads of ultra-light and/or low power sensors measuring different attributes, concerning environmental or health contexts. The overall management of this big quantity of devices and its associated applications is challenging.

- High user mobility: supporting broadband for mobile users in fast moving vehicles such as high speed trains. These trains are currently used for inter-city transport and it is anticipated they will further evolve after 2020, reaching speeds greater than 500 Km/h.

- Broadband access in dense areas: provisioning of service availability in densely-populated areas (*i.e.*, thousands of people per square kilometre) with up to 10Gbps bandwidth, for instance, ultra high definition video streaming.

- Broadband access everywhere: a minimum bandwidth of 50Mbps with ultra-low cost networks for digital inclusion of people living in low population areas.

- Extreme real-time communications: supporting of real-time interaction use cases such as tactile internet where ultra-low latency connectivity is essential. In the tactile internet system, humans control real and virtual objects in a wireless way. Examples of scenarios for Robotic control and interaction are manufacturing, medical care, and autonomous cars where reaction time must be within sub-millisecond.

- Lifeline communications: the mobile network as a lifeline enabling traffic peaks and high availability support in natural disaster and emergency cases.

- Ultra-reliable communications: low-latency, reliability, and availability for providing use cases such as automated traffic control and driving.

- Broadcast-like services: efficient distribution of information from one source to many destinations enabling a feedback channel for interactive services.

2.1.2 5G System Architecture

The 5G system architecture includes the 5G New Radio and the new 5G Core (5GC) [16]. The 3GPP has defined a service-based architecture (SBA) for the 5GC in [1]. The objective of SBA is to enable 5GC to be deployed in the cloud and leverage technologies like SDN and NFV as well as service-based interactions of network functions (NFs). SBA is depicted in Figure 2.1.

SBA design is guided by the next principles [16]:

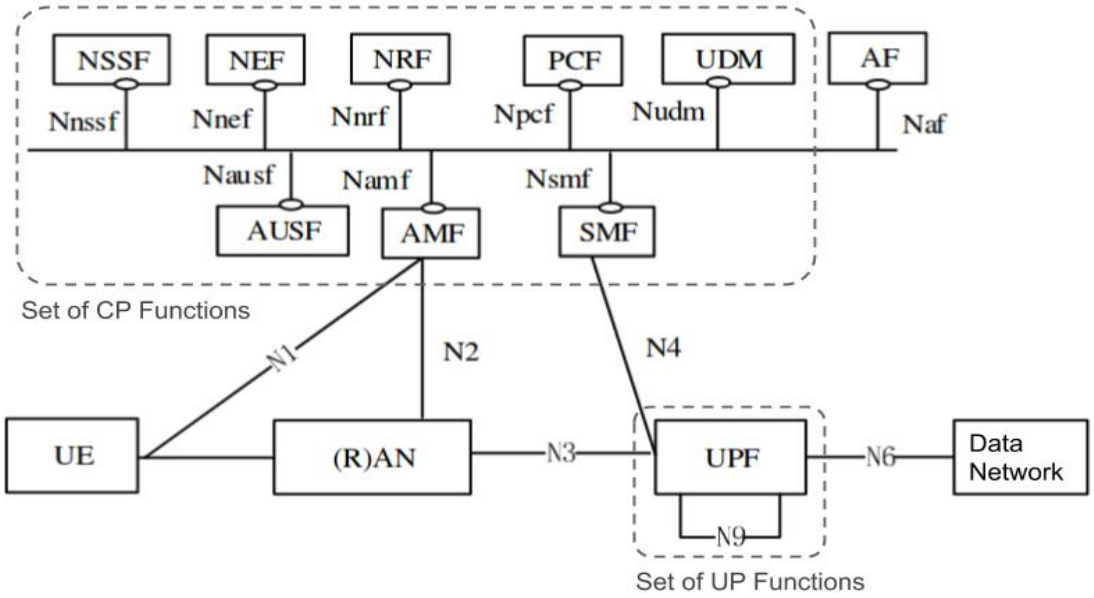


Figure 2.1 5G Service-based Architecture - adapted from [1]

- Control plane and user plane separation (CUPS). SBA distinguishes between control plane (CP) and user plane (UP) network functions to allow flexible deployments at centralized and edge locations, independent technical evolution, and scalability.
- A modular function design to enable implementing network slices with different requirements.
- Minimize dependencies between the access and the core networks aimed to permit operators to build a multi-access core network with common access-core interfaces.
- A unified authentication framework for the multi-access core to offer services independently of the access method.
- Support capability exposure.
- Concurrent access to local and centralized services. UP functions can be deployed close to the access to support low latency services.

5GC consists of various NFs. CP functions connect to each other over service-based interfaces (SBIs) with authorization to access each other's services. NFs are self-contained, independent, and reusable. Each NF service exposes its functionality through an SBI, which employs a well-defined REST interface using HTTP/2. NFs composing the SBA are:

- User Plane Function (UPF) performs packet routing and forwarding.
- Access and Mobility Management Function (AMF) handles connections and mobility tasks.
- Session Management Function (SMF) establishes, modifies, and releases PDU sessions.

- Policy Control Function (PCF) provides a policy framework for network slicing, roaming and mobility management.
- Unified Data Management (UDM) stores subscriber data and profiles.
- NF Repository Function (NRF), a database offering NF registration to allow NFs to discover each other.
- Network Exposure Function (NEF), an API for external users, such as enterprises or partner operators, to monitor, provision and enforce application policies.
- Authentication Server Function (AUSF) to manage security tasks.
- Network Slice Selection Function (NSSF) redirects traffic to the appropriate network slice according to the service type.
- Application Function (AF), an application that influences on traffic routing and accesses the NEF.

2.2 5G Network Slicing

Network Slicing is a concept strongly related to network softwarization, an emerging trend involving technologies like SDN and NFV [46]. Through the use of these technologies, network softwarization can provide programmability, flexibility, and modularity to build multiple logical networks (known as network slices) deployed over a single network infrastructure and each customized to accomplish particular needs. A network slice (NSL) is composed of virtual resources (*e.g.*, Virtual Machines) containing a set of Virtual Network Functions (VNFs), which may vary depending on the service requirements. For example, an NSL offering a real-time communication service will receive the appropriate resources to accomplish ultra-low latency constraints. NSLs are created on-demand according to the following principles [2]:

- *Isolation.* NSLs need independent control and management to reach performance and security guarantees. Isolation can be reached by using different physical resource, separating a shared resource through virtualization, and sharing a resource according to a policy that delineates the access rights.
- *Automation.* This principle enables the configuration of NSLs on-demand with low or no manual intervention. Automation can be accomplished by signaling-based mechanisms.
- *Customization.* NSL customization can be performed on the data plane by forwarding mechanisms, and on the control plane by programmable policies and protocols.
- *Elasticity.* This principle allows reshaping the allocated resource (*i.e.,* scaling up/down or relocating VNFs) in order to assure the required SLAs.

- *Programmability.* It denotes to the capability to control the allocated resources to NSLs by using open APIs.
- *End-to-end.* It refers to the service delivery throughout all the network (*i.e.,* RAN, core network, and transport) from the service providers to the end-users also known as customers.
- *Hierarchical abstraction.* Resources of an NSL allocated to a tenant can be further sliced to create new NSLs.

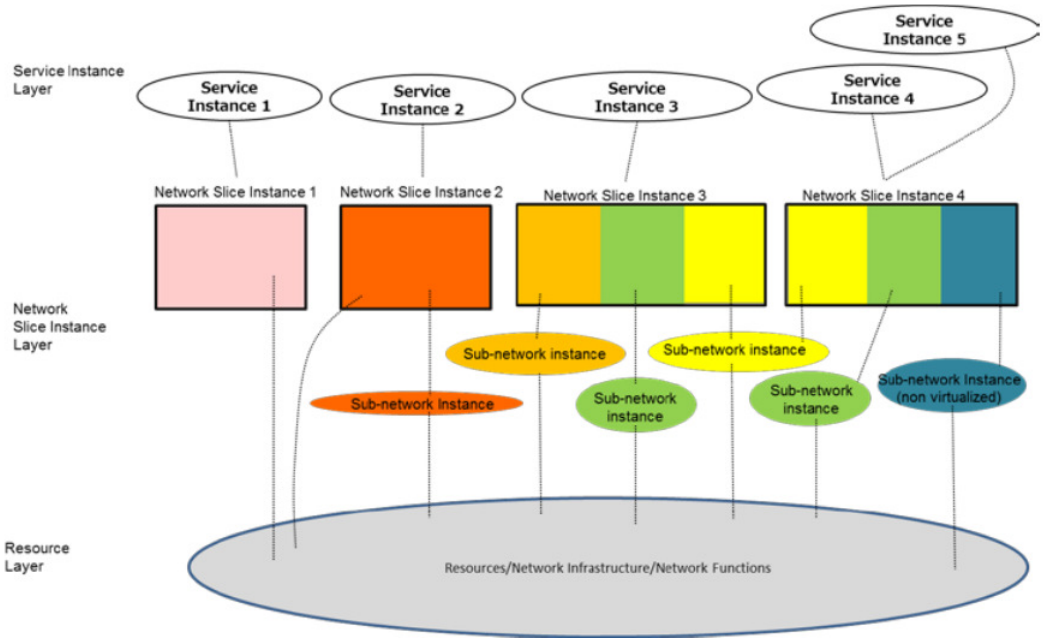


Figure 2.2 Network Slicing Layers [8]

As stated in [8], the network slicing concept is composed of three layers (see Figure 2.2): (*i*) Service Instance Layer, (*ii*) Network Slice Instance Layer, and (*iii*) Resource Layer. Service Instance Layer is composed of several service instances. Each of those instances represents a service provided by a network operator, an application provider or a vertical segment.

Network Slice Instance Layer consists of NSL instances. Each of them includes a set of customized resources (*i.e.*, physical and logical resources) aiming at meeting the performance requirements of specific services. None or several isolated or shared sub-network instances may compound an NSL instance. A network operator uses an NSL blueprint to create an NSL instance with specific characteristics, for example, ultra-low latency, ultra reliability, or value-added services for enterprises. This NSL instance may also be shared across multiple service instances generally belonging to the same type of service [2]. An NSL blueprint is a detailed description of the structure, configuration and the plan/work flows to control the life cycle of NSL instances [8].

The Resource Layer is the physical network topology that includes the Radio Access Network (RAN), transport network, and core network. This layer provides the physical network resources for NSLs.

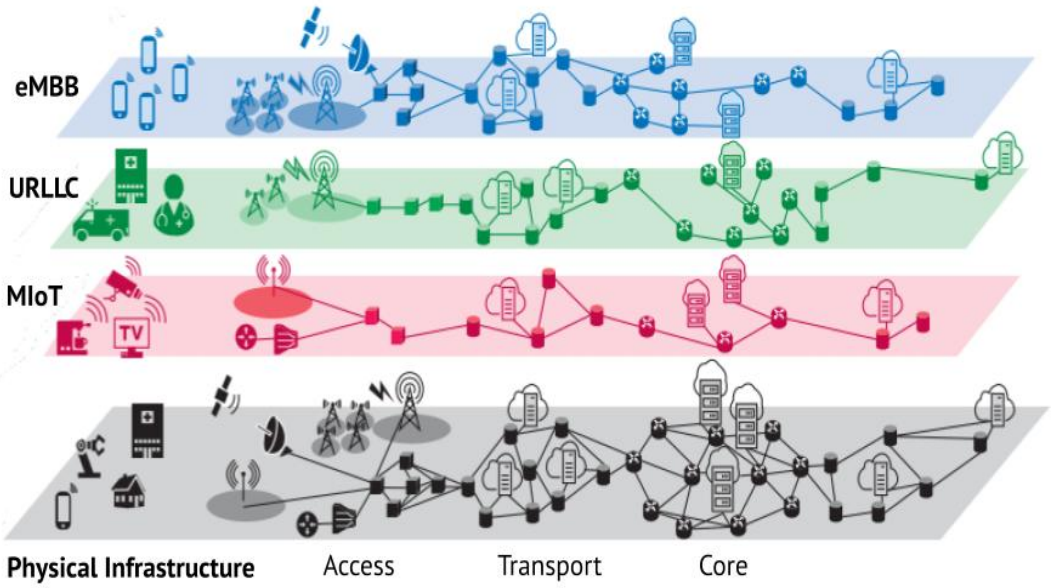


Figure 2.3 5G Network Slices sharing the same infrastructure - adapted from [46]

According to the The International Telecommunication Union, 5G use cases can be grouped into three general use case families: eMBB, URLLC, and MIoT [27]. Figure 2.3 shows an example of three 5G NSLs, each belonging to a different use case. Those NSLs are operated on the same physical infrastructure.

Each 5G use case has particular requirements. In eMBB, a wide range of VNFs are activated and distributed across the network (i.e, core and edge nodes) and high-capacity virtual links are instantiated. In URLLC, a very low end-to-end latency and high reliability are required. The former implies that some of the VNFs have to be deployed closer to the access nodes (i.e., at the edge cloud nodes) while the latter means that some of the VNFs have to be instantiated multiple times on different physical servers to serve as backup. In MIoT, the set of VNFs are limited compared to the previous use cases. MIoT fits a simplified slice without support for "always-on" connection, mobility handling, and low latency requirements [18, 7].

2.3 Machine Learning

Network Slicing tasks such as admission control and resource allocation require to cope intelligently with diverse and complex decisions for providing 5G NSLs to tenants. The application of ML, a subfield of Artificial Intelligence, has experienced considerable growth in carrying out such kind of tasks. ML's growing application is a consequence of an ever-increasing data availability and improvements in ML techniques as well as computing capabilities [15].

ML aims at extracting, utilizing, and improving knowledge over time with the help of experience. ML allows a computer software to learn on the basis of using previous experiences obtained from similar situations. The learning process begins when the system expected to learn takes a sample and learns certain information from it. Subsequently, it looks for a second sample to get more information [17]. This process helps to make

generalizations for the situation to be learned. With this aim, ML identifies and exploits hidden patterns in a set of data called training data [15]. These training data are used to construct the ML model.

Four paradigms are identified in ML: Supervised, Unsupervised, Semi-supervised, and Reinforcement Learning [15] [62]. Each of these paradigms differ from the others in the way of: (*i*) performing data collection, (*ii*) carrying out feature engineering intended to both pre-processing data, aiming at cleaning of noisy or incomplete instances, and feature extraction/selection; and (*iii*) determining ground truth which is about giving a formal description (*i.e.*, labels) to the classes [15].

Supervised Learning uses labeled training datasets (*i.e.*, data for which the output is already known) to create models. This approach is used in classification and regression problems. Classification problems intend to predict discrete valued outcomes while regression problems are for continuous outcomes. Semi-supervised techniques are applied when the training dataset has incomplete or missing labels [15]. Unsupervised Learning uses datasets without labels (*i.e.*, no outputs are given [42]) in the creation of models able to find patterns. Clustering problems belong to this approach [39].

RL is about learning by trial-and-error inspired by the behaviorist psychology [58]. This paradigm is appropriate for making cognitive choices such as decision making, planning, and scheduling problems [15]. RL differs from the other ML paradigms in that it does not use sample data from which to learn. Instead, RL is based on an agent that learns how to behave by iteratively interacting with an environment and observing the consequences of its actions [15] [42].

There are different classes of RL algorithms, such as value-based, policy-based, and actor-critic [10]. Value-based algorithms, useful for a finite set of actions, estimate the expected reward for different actions, *i.e.,* they learn a value function approximation from which the policy is inferred. Policy-based algorithms, appropriate for a continuous or stochastic action space, directly learn the policy function that maps state to action. Actor-critic algorithms are hybrid; they combine an actor (policy-based) that controls the algorithm behavior and a critic (value-based) that evaluates the actor's actions.

Q-learning is an example of value-based algorithms [58]. In Q-learning, the agent takes an action (a_t) under a state (s_t). After such an action is applied to the environment, the agent receives a reward (r_t) for its action and observes the next state (s_{t+1}) to repeat the cycle. The agent uses a lookup table, known as Q-table, to store the quality value (*i.e.,* the Q-value) of each action for a given state. The Q-table guides the agent to take the optimum action in every state.

Recently, Deep Reinforcement Learning (DRL), a technique that leverages both RL and Deep Learning (DL), has been widely used in several domains [40]. RL enables a self-learning agent to maximize long-term performance by interacting with the environment and receiving feedback from it. DL allows RL to manage problems with large state and action spaces by generalizing knowledge [25, 10]. Generalization allows to apply the knowledge learned from some already visited states to other similar states. Deep Q-learning (DQN) is an example of DRL algorithm. In DQN, the behavior of the agent is not guided by a table as in Q-earning. Instead, the agent uses a function approximator that estimates the Q-values for each action [10].

The solutions we introduce in this work are based on Q-learning and DQN. These techniques are discussed deeper in Chapter 3 and Chapter 4, respectively.

Chapter 3

Admission Control and Resource Allocation for 5G Network Slices based on Reinforcement Learning

5G networks support a myriad of services accessible on-demand by numerous customers and devices [2, 54]. The International Telecommunications Union defined several use cases with different QoS requirements for 5G services [18, 20]. Network Slicing, a 5G component, is crucial for enabling support of diverse QoS requirements since it provides flexibility, modularity, and programmability to the management of NSLs, customized and isolated logical networks [49]. The employment of SDN [38] and NFV technologies [32] allows to customize NSLs for accomplishing SLAs [46].

NSPs receive NSLRs to implement network slices of distinct types such as eMBB, URLLC, and MIoT, each possessing its own QoS requirements. NSPs need to implement an admission control mechanism to decide on the acceptance of NSLRs, considering the support of QoS requirements as well as the availability of physical resources. A resource allocation mechanism should guarantee the resources needed by an NSL. Resource allocation can be carried out as a Virtual Network Embedding problem that maps virtual networks onto a physical network [35, 57, 51, 6]. In virtualized 5G networks, physical resources are allocated to VNFs and virtual links. The acceptance of NSLRs depends not only on the availability of resources but also on the criteria used by NSPs such as maximization of monetary profit [48, 53, 14].

Several approaches have addressed admission control by employing different theoretical frameworks such as Queuing Theory [33], Big Data [50], Heuristics [37, 55], and RL [12, 49]. Such approaches propose making admission decisions considering individual NSLRs, which can lead to sub-optimal decisions since more profitable requests arriving in the short term can be rejected due to the unavailability of resources recently allocated [24]. Moreover, most of these proposals neither consider the QoS requirements of different service types (use cases) nor the allocation of resources in 5G core network nodes.

Furthermore, several other solutions based on heuristics [64, 41], Queuing Theory [3], and Complex Network Theory [27] have considered only resource allocation and neglected admission control, which prevents the achievement of NSP goals. On the other hand, performing admission control and resource allocation jointly and intelligently can

considerably improve the achievement of target objectives.

This chapter introduces a novel approach for admission control resource allocation of NSLRs in 5G networks, named SARA (network **S**lice requests **A**dmission and **R**esource **A**llocation), which employs an RL-based algorithm to explore, exploit, and learn to prioritize NSLRs collected in time windows.

The organization of this chapter is as follows: Section 3.1 presents related work. Section 3.2 introduces the architecture of SARA. Sections 3.3 and 3.4 describe the proposed mechanisms for admission control and resource allocation, respectively. Section 3.5 presents an evaluation of SARA. Lastly, section 3.6 concludes this chapter.

3.1 Related Work

In this section, we present research about Admission Control in 5G Network Slicing. Also, we introduce works addressing Resource Allocation in 5G Network Slicing.

3.1.1 Admission Control in 5G Network Slicing

Several investigations [33, 50, 55, 37, 12, 49] have addressed the admission control problem in 5G networks. Han et al. [33] proposed a utility-driven and multi-service based solution for network slicing based on Queuing Theory to maximize the network utility. This solution considers different queues for two types of requests and accounts for impatient customers. Raza et al. [50] propose an AC mechanism using Big Data Analytics for traffic prediction to increase the profit of infrastructure providers. This mechanism admits slice requests only when no service degradation is expected.

Jiang et al. [37] introduce a heuristic-based admission control mechanism to maximize users' Quality of Experience to meet the QoS requirements of NSLs. This mechanism decides about the acceptance of slice requests based on the solicited minimum and maximum data rates and available resources in the Radio Access Network. Furthermore, it dynamically changes the allocation of radio resources to different NSLs according to the traffic load.

Sciancalepore et al. [55] introduce a 5G network slice broker for radio resources based on RL that includes three modules: forecasting, admission control, and scheduling. The forecasting module predicts the network traffic used to dimension NSLs. The admission control module selects the NSLs by using a heuristic algorithm based on the Knapsack problem [9]. The scheduling module serves the tenant traffic of the granted radio NSLs. The broker uses RL to provide feedback to the forecasting module for resizing the slices, but RL is not used to make admission decisions of requested slices.

Bega et al. [12] introduce an analytical model based on Semi-Markov Decision Process, and a Q-learning based algorithm to perform admission control for individual slicing requests. Raza et al. [49] propose an admission control mechanism based on RL for maximizing the provider's profit. This solution considers a 5G flexible RAN and prioritized requests with different latency requirements and expected revenues. The proposed approach uses a Q-learning agent to decide on slice acceptance to maximize revenue.

The mechanisms in the aforecited papers [33, 50, 55, 37, 12, 49] make an admission decision for each request arrival, which prevents the selection of the NSLRs that potentially optimize an objective. These papers do not consider different types of requests according to 5G use cases, neglecting the diversity of QoS requirements of 5G service types. Moreover, they focus on edge nodes and neglects 5G core network nodes (see Table 3.1).

3.1.2 Resource Allocation in 5G Network Slicing

Several papers have addressed the resource allocation problem in NFV [22, 23, 61]. Solutions to RA in 5G adopt different techniques, such as heuristics [64, 41], Queuing Theory [3], and Complex Network Theory [27]. Zhang et al. [64] introduce a heuristic-based approach for placing VNFs on the core network for optimizing the acceptance ratio, the execution time, and throughput. When throughput degradation occurs, this heuristic changes the placement of VNFs.

Li et al. [41] propose a two-stage heuristic algorithm for slice provisioning that improves the revenue-to-cost ratio. The first stage performs node provisioning by considering the local and the global resource capacities and the topological attributes of physical nodes. The second stage performs link provisioning by using the k-shortest path algorithm [28].

Agarwal et al. [3] propose MaxZ, a solution to perform VNF placement, resource assignment, and traffic routing based on Queuing Theory to reduce service delay. MaxZ decouples the decision of VNF placement (*i.e.,* the physical hosts on which VNFs run) from CPU assignment (*i.e.,* how the VNFs share the computational capabilities). This solution makes placement and assignment decisions for each VNF.

Guan et al. [27] present a mapping algorithm for 5G network slices based on Complex Network (CN) theory [13] to analyze the topological characteristics of the network. The mapping process has two steps: VNF placement and VNF chaining. VNF placement selects the physical nodes to host VNFs by ranking them according to their topological properties (*i.e.,* degree and betweenness centrality), while VNF chaining chooses the physical paths by using the k-shortest path algorithm for the VNFs placed on the nodes.

The aforecited papers [64][41][3][27] focus on mapping NSLs. They map the arriving NSLRs without performing admission control. Performing admission control and resource allocation jointly in 5G core network slicing is critical to optimize resource usage and maximize NSP profit. RL is a well-suited candidate for such kind of problem. RL employs a Markov decision process (MDP), an efficient tool to solve decision-making problems. Furthermore, the Network Slicing process involves repetitive decisions and produces a large quantity of data that can be used to train RL-algorithms [43, 40].

Our approach, SARA, is based on a model-free RL, and does not make assumptions about the environment (*i.e.,* substrate network) but it learns while exploring such a environment without prior knowledge about it. SARA works on-line, therefore, it continuously learns from the environment. To the best of our knowledge, no other work has proposed a solution based on RL that jointly performs admission control and resource allocation, differentiates core and edge nodes and considers the typical types of 5G services (see Table

3.1).

Table 3.1 Related Work

Work	Technique	Focus		Time Window	Fifth Generation			Performance Metrics
		AC	RA		Use Cases	Edge Nodes	Core Nodes	
[49]	RL (Q-learning)	✓				✓		Provider profit
[33]	Queuing theory	✓				✓		Utility rate, admission rate, request waiting time
[50]	Big Data Analytics	✓				✓		Provider profit
[37]	Heuristic algorithm	✓	✓			✓		QoE, resource utilization
[55]	Heuristic algorithm	✓				✓		System resource utilization
[12]	RL (Q-learning)	✓				✓		Provider revenue
[64]	Heuristic algorithm		✓		✓		✓	Acceptance Ratio and Execution Time
[41]	Heuristic algorithm		✓				✓	Acceptance ratio, provider revenue
[3]	Queuing Theory		✓				✓	Running time
[27]	Complex Network Theory		✓		✓	✓	✓	Resource efficiency, acceptance ratio, execution time
SARA	RL (Q-learning)	✓	✓	✓	✓	✓	✓	Provider profit, resource utilization, acceptance ratio

3.2 Architecture of SARA

This section describes SARA architecture, which is illustrated in Figure 3.1.

3.2.1 5G Substrate Network

The substrate considered in this paper follows the European Telecommunications Standards Institute recommendation in [21], which defines that an NFV Infrastructure can span across several locations where Points of Presence (NFVI-PoPs) are operated. The network between these locations is part of the NFV Infrastructure and should be considered as well. NFVI-POPs are nodes that offer processing capacity and can be high capacity data centers (core nodes) or small data centers close to end users (edge nodes).

Core nodes are appropriate to the 5G Control Plane since it involves VNFs demanding high processing and bandwidth capacities [26][29]. Edge nodes are adequate to the 5G

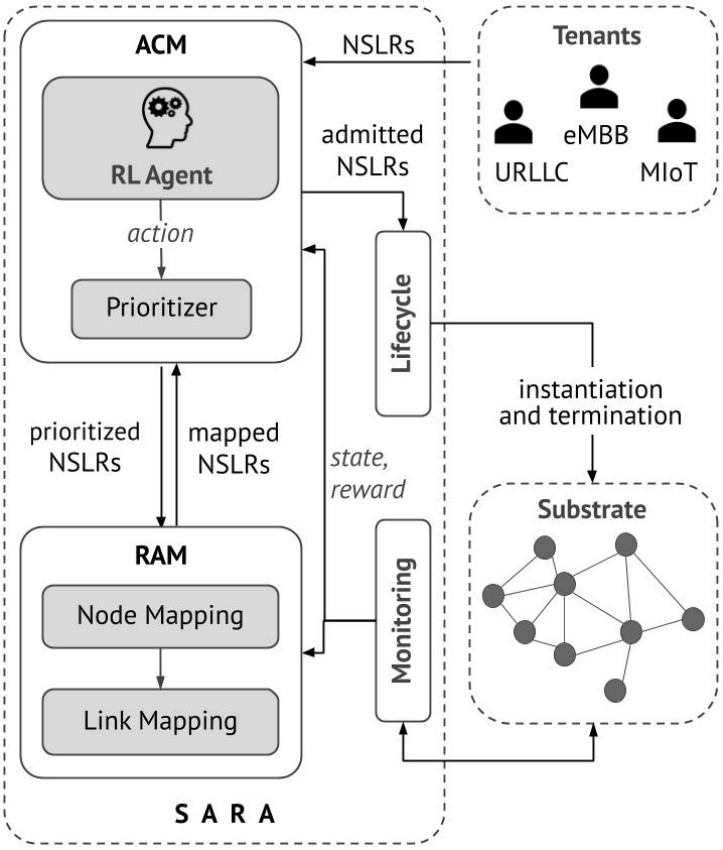


Figure 3.1 Architecture of SARA

User Plane since it may involve VNFs needing to be close to the end-users [18]. Moreover, edge nodes are adequate to a Fog-RAN involving remote radio heads and a centralized pool of virtual baseband units [47][30].

Each VNF that composes an NSL can be placed either on an edge node or a core node. The eMBB service type requires a wide range of VNFs activated and distributed across core and edge nodes. URLLC instantiations require VNFs deployed at the edge to support latency requirements. MIoT slices can have their VNFs on core nodes since there is no strict latency requirement for this type of service [18].

We model the 5G substrate network as a labeled and weighted undirected graph: $SN = \{N, L\}$, where N stands for the set of nodes, $N = \{n_1, n_2...n_m\}$, and L stands for the set of links, $L = \{(n_1, n_2), (n_1, n_3)...(n_l, n_m)\}$. Each node $n_i \in N$ has a processing capacity represented by $CPU(n_i)$. The bandwidth of a link (n_i, n_j) is given by $BW(n_i, n_j)$. Figure 3.2 depicts an example of a substrate network offering processing and bandwidth resources.

3.2.2 5G Network Slice Requests

An NSLR is described by $nslr = \{s\_type, T_o, G\}$. The s_type defines the 5G service: eMBB, URLLC, or MIoT. T_o defines the requested operational time (*i.e.,* the time the NSL will be active). $G = \{F, V\}$ is a labeled and weighted undirected graph representing an NSL, where F is the set of VNFs, and V is the set of virtual links connecting them. The labels on the nodes give the amount and type of resources demanded by a VNF.

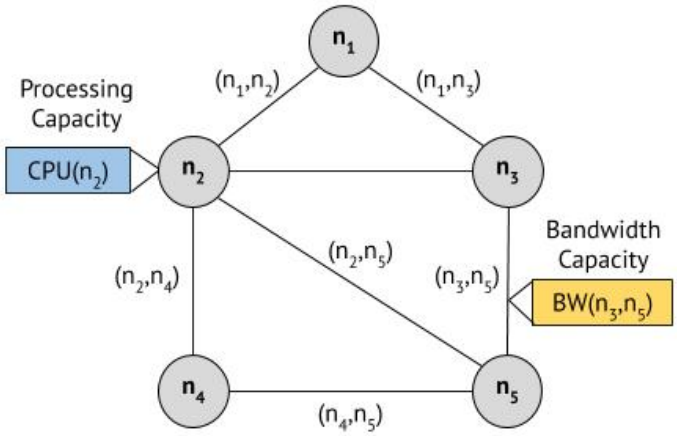


Figure 3.2 Example of a Substrate Network offering processing and bandwidth resources

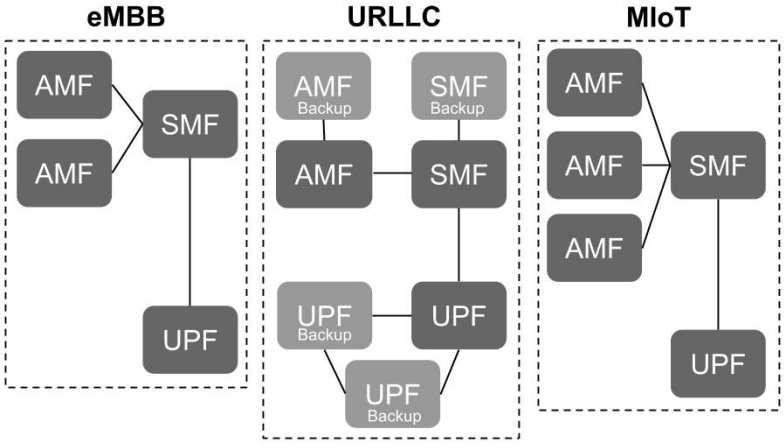


Figure 3.3 NSL graphs

The weight on the edges is the bandwidth requested by the virtual link. The processing capacity required is denoted by $cpu(vnf_i)$ and the node type a VNF requests by $type(vnf_i)$. Similarly, $bw(vnf_i, vnf_j)$ is the bandwidth demanded for the link (vnf_i, vnf_j).

Figure 3.3 depicts graphs for three types of use cases most common in 5G. They include essential VNFs such as Access and Mobility Management Function (AMF), Session Management Function (SMF), and User Plane Function (UPF) [16]. The URLLC graph considers backups for AMF, SMF, and UPF that should be instantiated at different nodes close to the end-users since URLLC must offer high reliability and low latency [18]. The eMBB and MIoT graphs do not include backups for SMF or UPF as these service types do not have ultra-high reliability requirements. The MIoT graph has more AMFs than the other types, as it must provide access to several type of devices [20]. The eMBB G has fewer AMFs than the MIoT graph since the former attends fewer devices than the latter.

3.2.3 Modules

SARA includes four modules: the *Monitoring Module*, the *Admission Control Module (ACM)*, the *Resource Allocation Module (RAM)*, and the *Lifecycle Module*. SARA processes the arriving NSLRs in batches collected in time windows. The Monitoring Module collects information about resource availability in the network substrate (nodes and links) and, periodically, delivers it to the ACM and RAM. Information is structured as states and rewards (see Section 3.3).

ACM performs the admission of NSLRs. It employs an RL-agent and a Prioritizer. The RL-agent determines a normalized weight value for each type of service. The Prioritizer uses these values to sort the NSLRs for establishing the order they should allocate resources. These weight values lead to the maximum profit, i.e., the RL-agent selects an action that, if taken, maximizes the profit. The RL-agent learns to select actions by considering the information on states and rewards got from the interaction with the environment. A state represents the available resources after the RL-agent executes an action, and a reward gives the profit generated by taking that action.

The Prioritizer enqueues NSLRs in batches of minimum size, obeying the proportion given by their weight values and arrival time. For instance, if weight values are 1.0, 0.5, and 0.5 for the eMBB, URLLC and MIoT, respectively, then batches with 2, 1, and 1 NSLRs of type eMBB, URLLC and MIoT are enqueued. NSLRs per class are enqueued in chronological order. If all the NSLRs of a particular type have been enqueued, the weight values of the other service type are used to determine the number of NSLRs in the subsequent batches. In the example just described, if there are 10 NSLRs per service type, in the queue, there will be a sequence of 5 batches with 2, 1, and 1 NSLRs of type eMBB, URLLC, and MIoT, followed by 5 batches composed by one NSLRs of URLLC, and MIoT type.

After assembling the priority queue, each NSLR is dequeued, and a request for allocation of resources is sent to RAM. If resources are successfully allocated, then the NSLR is accepted. Otherwise, it is rejected. The dequeuing of an NSLR and the attempt to allocate resources to it is repeated until the priority queue is empty.

RAM allocates resource on nodes for the VNFs composing an NSLR (node mapping), and then, allocates bandwidth in selected links (link mapping) connecting the allocated nodes. Decisions on node mapping consider not only if a node has resources available to support the demand but also the latency and reliability requirements of the NSLR type. Control Plane VNFs are mapped onto core nodes. User Plane VNFs of a URLLC NSLR are mapped onto edge nodes to satisfy strict latency requirements, while the User Plane VNFs of the other two types are mapped, preferably on core nodes [20, 16]. For complying with reliability requirements, a backup VNF is not placed onto the same node its primary VNF is placed on [18].

Link Mapping maps each virtual link onto the shortest substrate path that satisfies the required bandwidth by the virtual link. If Node Mapping and Link Mapping finish successfully, RAM sends a notification of successful allocation (mapped) to ACM. Otherwise, a non-mapped notification is sent.

Upon accepting an NSLR, the Lifecycle module instantiates its VNFs and virtual links,

creating, then, an NSL. When the lifetime of the NSL expires, resources are deallocated.

3.3 Admission Control based on Q-learning

In this section, we first summarize Q-learning. Then, we present the elements that specify the RL-agent of SARA. Finally, in subsection 3.3.3 we introduce the admission control algorithm.

3.3.1 Q-learning

ACM runs an admission control algorithm based on Q-learning, an RL technique [58] [40]. Mathematically, RL algorithms can be described by using Markov Decision Process (MDP), a framework to model decision-making problems. In MDP, the future state and reward of the environment depend on the current state and the action taken [40]. MDP can be represented with the tuple: $< S, A, P(s'|s,a), R >$ [10]. Where, S and A are the finite state space and action space, respectively. $P(s'|s,a)$ is the probability that action a under state s at slot t leads to state s' at slot $t+1$. R is an immediate reward function.

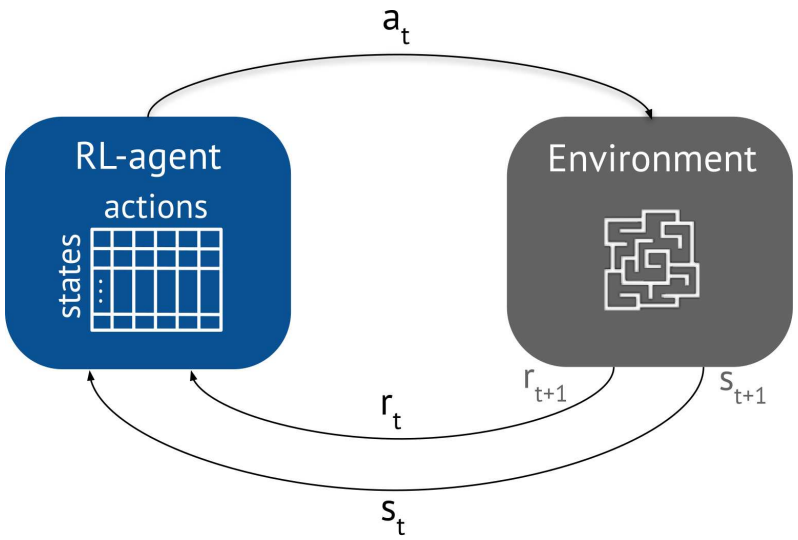


Figure 3.4 Q-learning operation

In Q-learning, the agent (see figure 3.4) takes an action (a_t) under a state (s_t). Such an action is applied on the environment. Subsequently, the agent receives a reward (r_t) from the environment, and observes the next state (s_{t+1}). In Q-learning, the agent uses a lookup table, known as Q-table, to store the quality value of each action (*i.e.,* the Q-value) for a given state [58]. The Q-learning agent updates the Q-values on the Q-table as follows:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \cdot [R_t + \gamma \cdot maxQ(s_{t+1}, a) - Q_t(s_t, a_t)] \tag{3.1}$$

where:

- $Q_{t+1}(s_t, a_t)$ is the new Q-value obtained after the update.
- $Q_t(s_t, a_t)$ is the old Q-value.
- α is the learning rate, a value between 0 and 1 that defines the Q-values the RL-agent keeps. If $\alpha = 1$, the RL-agent discards the old Q-values; if $\alpha = 0$, it discards the most recently learned Q-value (*i.e.,* the RL-agent learns nothing new).
- R_t is the reward observed by the RL-agent after taking action a_t under state s_t.
- γ is the discount factor used to balance the immediate reward and the maximum expected future reward $maxQ(s_{t+1}, a)$. Its value is in the interval $[0, 1]$. If the discount factor is 1, the RL-agent fully considers the expected future reward. A low discount factor emphasizes more the immediate reward.

Q-learning is classified as a model-free, off-policy, and temporal-difference RL algorithm. Model-free means that it does not use the transition probability distribution (also called model) related to the MDP, therefore, Q-learning does not know the next state before taken the action. Off-policy refers to the fact that Q-learning does not follow the same policy for sampling (*i.e.,* to select an action for the current state) and updating its value function (Q-values). Instead, for updating the value function, it chooses the action corresponding to the best Q-value [40]. Temporal-difference means that Q-learning updates its value function by comparing estimates at two consecutive steps; the current estimate, $R_t + \gamma \cdot maxQ(s_{t+1}, a)$, and the previous one, $Q_t(s_t, a_t)$.

3.3.2 RL-agent for Admission Control

This subsection introduces the elements that specify our RL-based solution: state space, action space, reward function, and exploration and exploitation method.

State Space

We model states to represent the amount of available resources in the substrate network. The State Space S is the set of all states the RL-agent can experience. In this work, a state $s \in S$ is defined by the tuple $\{cpu(E), cpu(C), bw(L)\}$, where $cpu(E)$ and $cpu(C)$ are the available processing capacity in the set of edge (E) and core nodes (C), respectively. $bw(L)$ is the available bandwidth in the set of links (L).

To exemplify the State Space, let us suppose the following state: $s_i = \{80, 50, 60\}$. s_i indicates that 80% and 50% of the total capacity of processing is available in E and C, respectively, and 60% of the total capacity of bandwidth is available in L.

A large State Space implies that Q-learning may not converge quickly enough because its RL-agent would need to explore many states in order to learn. Aiming at keeping the number of states small, the capacity of resources in the substrate is discretized in ten equal intervals. As our State Space considers three variables, and each can take ten values, the total number of states the RL-agent has to explore is 10^3.

Action Space

The Action Space A is the set of all actions the RL-agent can take. In every step, the RL-agent selects the weights for each type of service (*i.e.,* NSLR of eMBB, URLLC, and MIoT). An action $a \in A$ is denoted by $a = \{w_{embb}, w_{urllc}, w_{miot}\}$, where w_{embb}, w_{urllc}, and w_{miot} are the weights for each type of service. In every step, the RL-agent chooses the action a which returns the maximum profit.

To exemplify the Action Space, let us consider the following action: $a_i = \{1, 0.5, 0.5\}$. This action indicates that SARA should prioritize NSLRs of eMBB. Specifically, it has to admit one NSLR of URLLC and one of MIoT every two NSLRs of eMBB.

The weights of each service type enables the RL-agent to learn what is the best admission proportion of NSLRs that leads to the highest profit for the current state.

Reward Function

The reward guides the RL-agent to accomplish the proposed objective. In this work, the reward expresses the monetary profit obtained by taking an action on an individual state. The action taken implies the acceptance of NSLRs of different types and consumes resources having distinct costs. Typically, resources on core nodes are abundant and cheaper than on the edge nodes.

The RL-agent uses the reward to maximize the NSP profit while optimizing resource utilization. In this sense, the reward considers the profit generated after the RL-agent makes an action. The profit obtained by the acceptance and subsequently instantiation of an individual NSLR, $p(nsl_i)$, is the amount of money earned by NSP for selling the NSL minus the operational cost. $p(nsl_i)$ is given by:

$$p(nsl_i) = (rev_i - cst_i) \times T_o \tag{3.2}$$

$$cst_i = \sum_{j=0}^{m} cpu(vnf_j) \times f_{cpuj} + \sum_{j=0}^{n} bw(v_j) \times f_{bw} \times h \tag{3.3}$$

where:

- rev - is the income that an NSP receives for instantiating the nsl_i
- cst - the cost of running nsl_i on the substrate
- T_o - is the nsl_i operational time
- m - gives the number of VNFs in nsl_i
- n - gives the number of virtual links in nsl_i
- $cpu(vnf_j)$ - is the cpu demand of vnf_j in nsl_i
- f_{cpuj} is the processing cost of VNF vnf_j, which depends on the node type

- f_{bw} is the bandwidth cost
- h is the number of hops composing the path where virtual link v_j is allocated.

The reward function, R, is given by:

$$R = \frac{\sum_{i=0}^{k} p(nsl_i)}{maxP(SN, T)} \tag{3.4}$$

where $maxP(SN, T)$ is the maximum profit that NSP could receive when using all the resources in the substrate (SN) in a certain period (T).

Exploration and Exploitation Method

The Q-learning RL-agent uses the epsilon-greedy method (Equation 3.5) to explore and exploit to choose an action in each step [59]. The epsilon-greedy method allows the selection of either the current expected optimal action with probability $1-\varepsilon$ or a random action with probability ε.

To choose an action, first, the RL-agent generates a random number $rn \in [0, 1]$. If $rn > \varepsilon$, the RL-agent selects the action with the maximum Q-value; otherwise, it chooses a random action. A high ε value enables the RL-agent to select more random actions than optimal actions; as a result, it explores new actions more frequently. In contrast, a low ε value allows the RL-agent to take more optimal actions than random ones; *i.e.,* it exploits more the current knowledge.

$$a = \begin{cases} \max_{\mathrm{a}} Q_t(s_t, a), & if \quad rn > \varepsilon \\ random\, action, & otherwise \end{cases} \tag{3.5}$$

3.3.3 Admission Control Algorithm

Algorithm 1 presents the RL-based Algorithm employed by SARA to perform admission control. The aim is to admit the most profitable NSLRs. Time is discretized. The algorithm receives as input a set of NSLRs arrived in a time window, $RS = \{nslr_1, ...nslr_n\}$, and produces as output weight values for each type of service.

Data employed by the algorithm include the Action Space (A), the State Space (S), the parameters learning rate (α), the discount factor (γ), the exploration-exploitation factor (ε), and the number of learning episodes (n). A learning episode comprises a sequence of m-steps; and each step comprises a state, an action, and a reward received.

Algorithm 1 starts by creating the Q-table (*Line 1*) with as many rows as the number of states ($|S|$) the RL-agent can experience, and as many columns as the number of

Algorithm 1: RL-based Admission Control Algorithm

Data :
Learning rate α
Discount factor γ
Exploration parameter ε
Number of learning episodes n
State space S
Action space A
Input : Sets of NSLR (RS) collected during time windows
Result: Admitted NSLRs that generates the maximum profit

Initialize $Q : Q(S, A) = 0, \forall s \in S, \forall a \in A$
for *episode* $\leftarrow 1$ **to** n **do**
    The agent observes the initial state s_i `// when 100% of substrate resources are available`;
    while *next state* s_{t+1} *is not the final state* **do**
        The agent chooses a_t using ε-greedy exploration method (equation 3.5) `// Selection of a random or an optimal action that leads to increase the profit`;
        The agent invokes the Prioritizer to sort the NSLRs into a priority queue PR;
        for *each* $nslr \in PR$ **do**
            The agent invokes RAM that runs algorithm 2 to map $nslr$;
            if $nslr$ *is mapped* **then**
                The agent admits $nslr$ and sends it to Lifecycle;
            end
            else
                The agent rejects $nslr$;
            end
        end
        The agent observes the reward R_t that is calculated by using equation 3.4;
        The agent observes the new state s_{t+1} `// The current resource availability`;
        The Q-table is updated according to Equation 3.1;
        The current state is updated $s_t \leftarrow s_{t+1}$;
    end
end

actions the RL-agent can take ($|A|$). This means, Q-table has the dimension $|S| \times |A|$. Initially, the entries of the Q-table are set to zero. The Q-table is a lookup table where the RL-agent updates the Q-values for each state-action pair.

The algorithm has an outer loop (*Line 2*) that goes through episodes and an inner loop that goes through m-steps (*Line 4*). In the inner loop, the RL-agent uses the ε-greedy method, defined in Equation 3.5, to select either a random action or an optimal action a_t. The optimal action allows the RL-agent to exploit the learned knowledge. As Q-values depend on the past rewards (*i.e.,* the normalized profit), the RL-agent learns to choose actions that increase the profit.

The optimum solution, a_t, is passed to the prioritizer (*Line 6*) which will sort all the NSLRs received in a priority queue according to their arrival time and the weight value of their type of service. Each NSLR in the priority queue is sent to RAM. If resources are allocated, the NSLR is mapped (*Line 8*) onto the substrate, *i.e.* is considered admitted. Information on the acceptance of an NSLR is passed to the Lifecycle module (*Line 10*).

Then, the RL-agent receives the reward for performing a_t (*i.e.,* the profit generated by PR, computed by Equation 3.4). The RL-agent also observes the new state s_{t+1} (*i.e.,* new available processing and bandwidth capacities in the substrate after executing a_t). In line 18, the RL-agent updates the Q-table by using Equation 3.1 considering the reward and the new state. Finally, the state s_{t+1} becomes the current state s_t and the RL-agent begins a new iteration.

3.4 Service-aware Resource Allocation

RAM aims at finding and allocating the appropriate resources to NSLRs by performing the mapping (embedding) $M : G = \{F, V\} \rightarrow SN' = \{N', L'\}$, while meeting the latency, bandwidth, processing and reliability requirements of each type of service. In the mapping M, N' is a subset of N and L' is a subset of L.

Algorithm 2 describes SARA resource allocation which is carried out in two steps, namely node mapping and link mapping. The input of Algorithm 2 is an $nslr = \{s\_type, T_o, G\}$ provided by the ACM. The output is a notification on either successful or unsuccessful allocation of resources.

The allocation information for accepted NSLRs, describing the substrate nodes and links where VNFs and virtual links must be instantiated, is finally passed to the Lifecycle module. The following subsections present node and link mapping steps and their operation.

3.4.1 Node Mapping

The Node Mapping step in Lines 1 to 15 of Algorithm 2, aims at mapping the VNFs of an NSLR onto nodes in the substrate network ($F \rightarrow N'$), while meeting processing, latency and reliability requirements. Each element in F represents a VNF and its processing requirement is given by $cpu(vnf)$. The service type s_type is used to choose the type of node in SN.

Algorithm 2: Resource Allocation Algorithm

Input :
An NSLR
Result: Mapped NSLR

for *each substrate node* $n \in N$ **do**
  Calculate embedding potential (equation 3.6);
end
Rank the substrate nodes N according to the embedding potential value in descending order
for *each* $vnf \in F$ **do**
  for *each node* n *in the ranked list* **do**
    if $match(type(n), type(vnf), s\_type) == True$ *and* $isAllowed(n, vnf) == True$ *and* $\frac{CPU(n)}{cpu(vnf)} \geq 1$ **then**
      Map vnf onto n;
      Break;
    end
  end
  if vnf *is not mapped* **then**
    Return *non-mapped* notification;
  end
end
for *each virtual link* $v \in V$ **do**
  Obtain source src and destination dst from v;
  Compute all simple paths from src to dst;
  Sort the simple paths into the list $CandidatePaths$ regarding their number of hops `// The first path in the list has the least number of hops`;
  for *each path* CP *in* $CandidatePaths$ **do**
    if *each link* $l \in CP$ *satisfies* $\frac{BW(l)}{bw(v)} \geq 1$ **then**
      Map v onto CP;
      Break;
    end
  end
  if v *is not mapped* **then**
    Return *non-mapped* notification;
  end
end
Return mapped $NSLR$;

VNFs are mapped according to their $type(vnf)$ (Control Plane or User Plane) and the service type (s_type) it belongs to. A VNF belonging to the Control Plane is always mapped to a core node. VNFs belonging to the User Plane are mapped depending on the service type:

- For the URLLC type, VNFs are mapped onto edge nodes to satisfy the strict latency requirements.
- For MIoT, User Plane VNFs are always mapped to core nodes since this service type is not latency-sensitive.
- For eMBB, User Plane VNFs are preferentially mapped to core nodes. However, they may be mapped to edge nodes when core nodes are not available.

To perform the node mapping, substrate nodes, $n \in SN$, are ordered according to their embedding potential value, EP, (*Lines 1 to 3*) given by Equation 3.6). The EP metric determines the capacity of a substrate node n_i to embed a VNF considering its available processing capacity $CPU(n_i)$ and available bandwidth on its adjacency $BW(adj(n_i))$. $BW(adj(n_i))$ is computed as the sum of the available bandwidth in all the links connected to n_i [34] [41].

$$EP(n_i) = CPU(n_i) \times \sum_{l \in adj(n_i)} BW(l_j) \tag{3.6}$$

The periodic computation of EP value allows balancing the distribution of VNFs, which avoids the saturation of a few nodes. RAM sorts the substrate nodes in decreasing order of EP value (*i.e.,* from highest to lowest EP). In this way, nodes with a lot of processing resources available and strongly connected to their neighbors are in the top of the ranked list.

In the second loop, *Lines 5 to 15*, an attempt is made to map the VNFs in F onto the ranked substrate nodes. Three conditions need to be satisfied for a successful mapping (*Line 7*): i) the node type ($type(n)$) needs to match with the service type of the NSLR (s_type) and the VNF type ($type(vnf)$) to guarantee the latency requirement, ii) a backup of a VNF should not be placed on the same node that the primary VNF in order to satisfy reliability requirements (method $isAllowed(n, vnf)$); and iii) the available processing capacity in the node is higher than the processing required by the VNF ($\frac{CPU(n)}{cpu(vnf)} \geq 1$).

If all conditions are satisfied, the vnf is mapped onto n, and the node resources are allocated. Otherwise, RAM visits the next node in the ranked list to verify if the mapping conditions hold. If RAM cannot map at least one VNF, it returns to ACM a *non-mapped* notification for *nslr* and deallocate all the nodes that have been allocated so far to the NSLR. In case all the VNFs are successfully mapped, RAM starts the Link Mapping step that is described in the following subsection.

3.4.2 Link Mapping

The Link Mapping step attempts to map virtual links onto paths (composed of substrate links), $V \rightarrow L'$, consuming the least amount of bandwidth. Paths in the substrate network with available bandwidth satisfying the bandwidth demanded and with the lowest number of hops is sought to minimize the network bandwidth utilization. If the number of hops is high, virtual links may occupy a lot of physical links. This situation evidently will run out bandwidth resources fast and could avoid admitting new incoming NSLRs.

In *Lines 16 to 29*, the last loop of Algorithm 2, RAM carries out the Link Mapping. For each virtual link, $v \in V$, an attempt is made to map it onto a substrate path. RAM starts obtaining the source *src* and destination *dst* nodes of v (*line 17*). RAM uses the depth-first search to compute all the simple paths (loop-free paths) between *src* and *dst* (*Line 18*). In *Line 19*, RAM sorts the simple paths, in descending order of their number of hops and put them in the $CandidatePaths$ queue.

Subsequently, RAM takes from $CandidatePaths$ the first path CP (*i.e.,* the shortest path in terms of number of hops). RAM verifies if the bandwidth availability in all links along CP satisfies the NSLR bandwidth requirement $\frac{BW(l)}{bw(v)} \geq 1$. If all links meet this condition, RAM maps v onto the links of CP; otherwise, RAM considers the next shortest path.

If RAM cannot map at least one virtual link, it returns a *non-mapped* notification to ACM, and nodes are deallocated. Otherwise, Node Mapping and Link Mapping steps finish successfully, and RAM sends a mapped notification to ACM.

3.5 Performance Evaluation

This section describes the evaluation of SARA. First, we introduce the metrics used for assessing SARA performance. Then, we detail the setup for the experiments conducted, and finally, results obtained in the experiments are discussed.

3.5.1 Metrics

The profit, resource utilization, and acceptance ratio are considered as the metrics for SARA evaluation. The profit P is calculated according to Equation 3.2, The resource utilization is given by:

$$U = \frac{\frac{\sum_j cpu(nsl_j)}{CPU(SN)} + \frac{\sum_j bw(nsl_j)}{BW(SN)}}{2} \tag{3.7}$$

where:

- $CPU(SN)$ - is the total processing capacity in SN,
- $\sum_j cpu(nsl_j)$ - is the processing resource utilized by all NSLRs instantiated in SN,

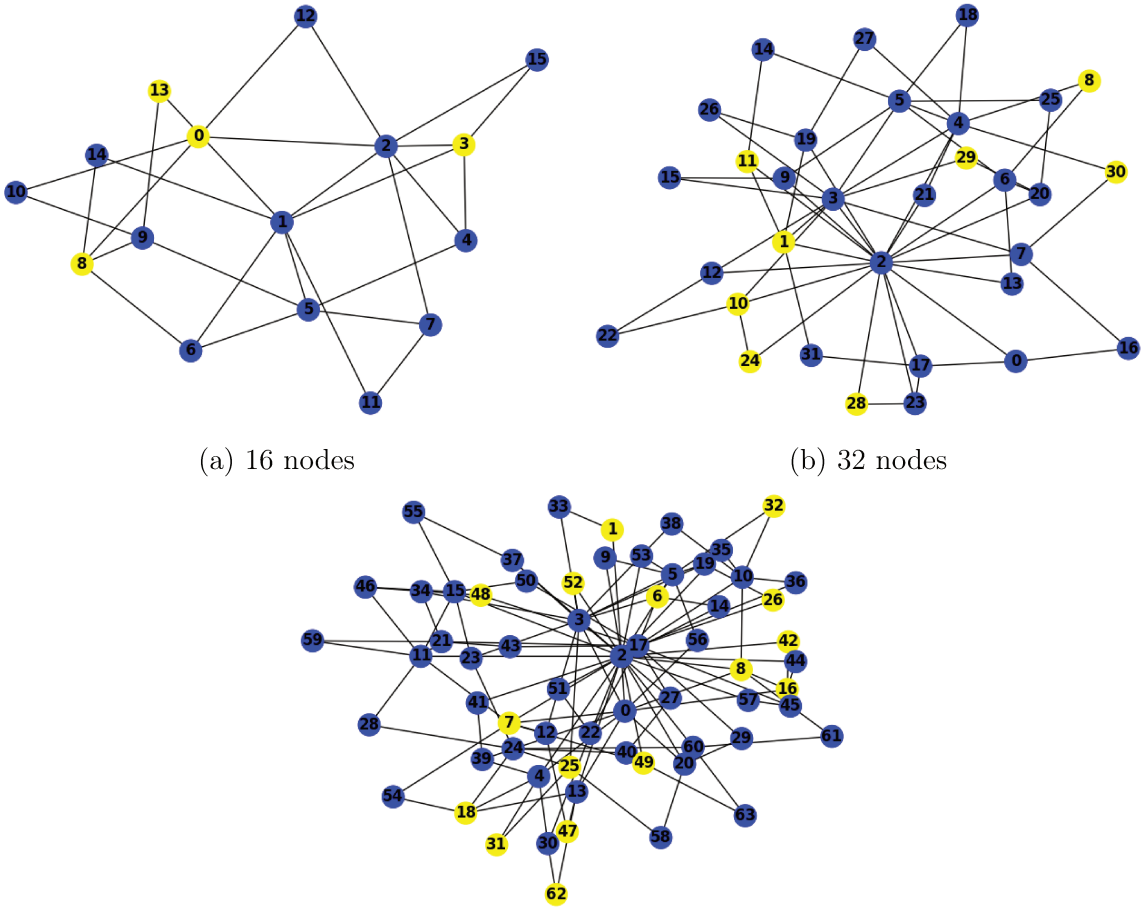
- $BW(SN)$ is the total network bandwidth,
- $\sum_j bw(nsl_j)$ is the bandwidth utilized by all NSLRs instantiated.

The acceptance ratio is the ratio between the number of admitted NSLRs (req_a) and the number of NSLRs (req_t) requested.

$$AR = \frac{req_a}{req_t} \tag{3.8}$$

3.5.2 Experiment Setup

SARA modules were developed using Python 3. We developed a Python-based discrete event simulator to test SARA and executed it on an Ubuntu 16.04 LTS desktop with Intel Core i5-4570 CPU and 15.5 GB RAM. The simulator uses the NetworkX library [31] to create and manipulate the graphs of NSLRs and the substrate network topology.



(a) 16 nodes

(b) 32 nodes

(c) 64 nodes

Figure 3.5 Substrate Network Topologies. Core nodes: yellow. Edge nodes: blue.

Experiments were conducted with different topology sizes. 16-node, 32-node, and 64-node topologies depicted in Figure 3.5 were generated by using the Barabasi-Alberth algorithm [5]. The 16-node topology composed by 4 core nodes (*i.e.,* yellow nodes) and 12 edge nodes (*i.e.,* blue nodes). The 32-node topology composed by 8 core nodes and 24 edge nodes. The 64-node topology has 16 core and 48 edge nodes. The core and edge nodes capacity is of 300 and 100 processing units, respectively. All substrate links have a capacity of 100 bandwidth units.

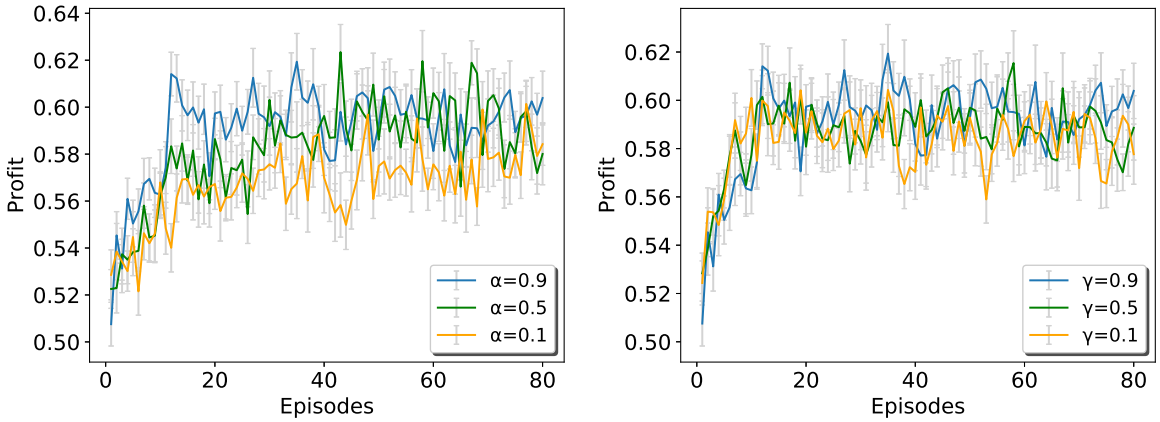
Parameter	**Value**
Substrate	16, 32, and 64-node topologies
Capacity of nodes (cpu units)	core: 300, edge: 100
Capacity of links (bw units)	100
Mean operational time (time units)	12
Total load (requests per time unit)	$1, 3, 5, 7, 10, 15, 20, 25, 30, 40, 60, 80, 100$
Time window (time units)	2
α (learning rate)	0.9
γ (discount factor)	0.9
ϵ (greedy factor)	0.1

Table 3.2 Simulation Parameters

The computational demand of each VNF is 5 processing units. The virtual links in the eMBB, URLLC, and MIoT service graphs require 3, 2, and 1 bandwidth units, respectively. The operational time of arriving NSLRs follows an exponential distribution with a mean of 12 time units. The arrivals of each of the three types of NSLR follows a Poisson process and have the same arrival rate [63]. The total arrival rate varied from 1 to 100 requests per time unit to assess SARA performance under different load conditions. The window duration is 2 time units. We carried out 33 repetitions to obtain results in 95% confidence intervals. Table 3.2 summarizes the simulation parameters.

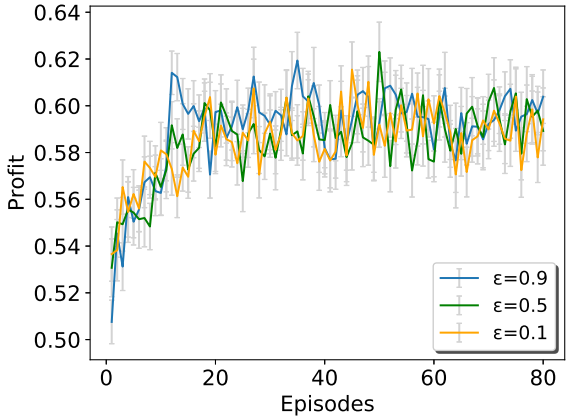
We trained the RL-agent in an episodic setting. Each episode consists of 15 steps. In order to define the values for the learning parameters, we conducted preliminary tests. Figures 3.6a, 3.6b, and 3.6c depict the profit results when we varied α (learning rate), γ (discount factor), and ϵ (exploration factor), respectively. From Figure 3.6a, we found that a learning rate of 0.9 makes SARA to converge faster than lower values such as 0.5 or 0.1. Results in Figure 3.6b show that SARA seems to converge a little faster with low γ values (0.5 and 0.1), however, it gets the highest profit values with γ fixed to 0.9. Regarding ϵ in Figure 3.6c, when we set it to high values such as 0.5 or 0.9, SARA takes longer to reach its convergence than when we set it to 0.1. In accordance with the above, we fixed α and γ to 0.9 and ϵ to 0.1.

SARA performance was compared to those of two heuristics; the Always Admit Requests algorithm (AAR) and the Node Ranking algorithm (NR). NR ranks the substrate nodes according to the embedding potential defined in [35] and [41], while ARR does not differentiate nodes for allocation. These heuristics admit NSLRs as they arrive if there is enough available capacity to do so. Unlike SARA, these heuristics do not use any strategy to reach target goals.



(a) Varying learning rate

(b) Varying discount factor



(c) Varying exploration factor

Figure 3.6 Profit for different learning parameters

3.5.3 Results for a 16-node topology

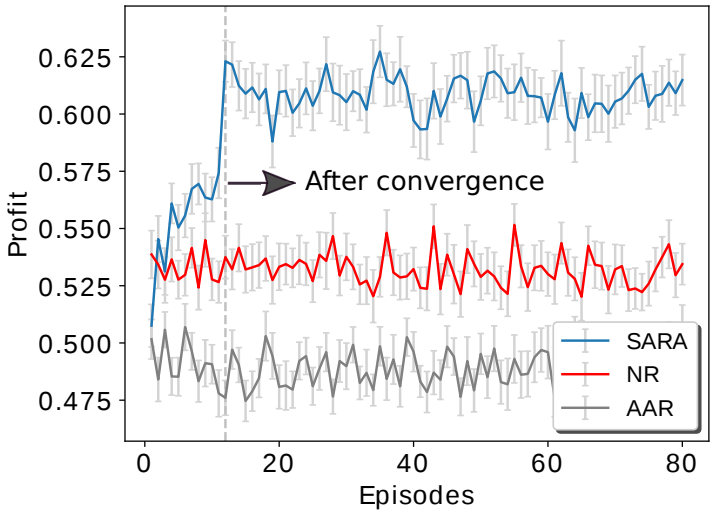


Figure 3.7 Profit comparison on a 16-node topology

Figure 3.7 depicts the profit as a function of time for an arrival rate of 20 requests per time unit. SARA increased the profit from episodes 1 to 12 when it converged. SARA

produced the highest profit values even before converging (except for the first episode). Overall, SARA's profit is 7% and 11.3% higher than those obtained by NR and AAR, respectively. These profit values are a consequence of SARA using an RL-algorithm that quickly learns the most profitable combination of NSLRs.

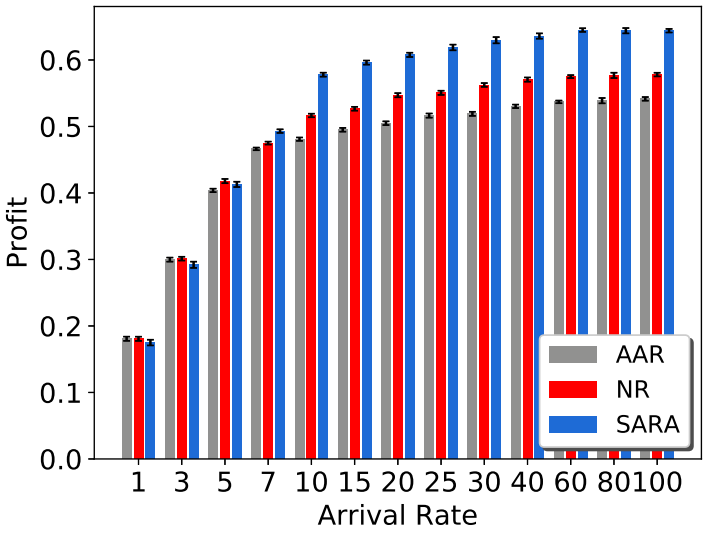


Figure 3.8 Profit vs Arrival Rate on a 16-node topology

Figure 3.8 shows the profit as a function of the total arrival rate. Under low loads (*e.g.,* 1 requests per time unit), SARA obtained lower profit values than those given by the other two heuristics. If the arrival rate is too low, the number of arrivals of NSLRs is not enough to provide useful information to the RL-agent. On the other hand, for arrival rates equal to or greater than 7 requests per time unit, SARA obtained the highest profit. For example, for 7 requests per time unit, the RL-agent obtained 2.4% higher profit than the NR profit, and for 60 requests per time unit, the difference raised to 7.9%. Summarizing, SARA outperforms AAR and NR under medium to high loads because the RL-agent learns to admit the proper proportion of each NSLR type to increase the profit.

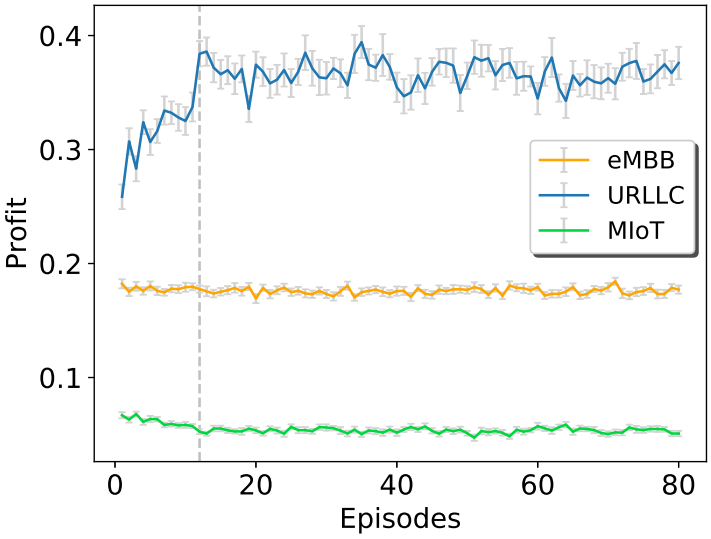


Figure 3.9 SARA - Profit per NSLR Type on a 16-node topology

Figure 3.9 presents the contribution of each type of service of admitted NSLR to the total profit obtained by SARA for a load of 20 requests per time unit. The contribution of URLLC NSLRs to the total profit grew from episodes 1 to 12 when SARA obtained the maximum profit. Conversely, the profit of NSLRs of eMBB and MIoT slightly decreased from episodes 1 to 12. SARA learns to identify the NSLRs that generate high profit.

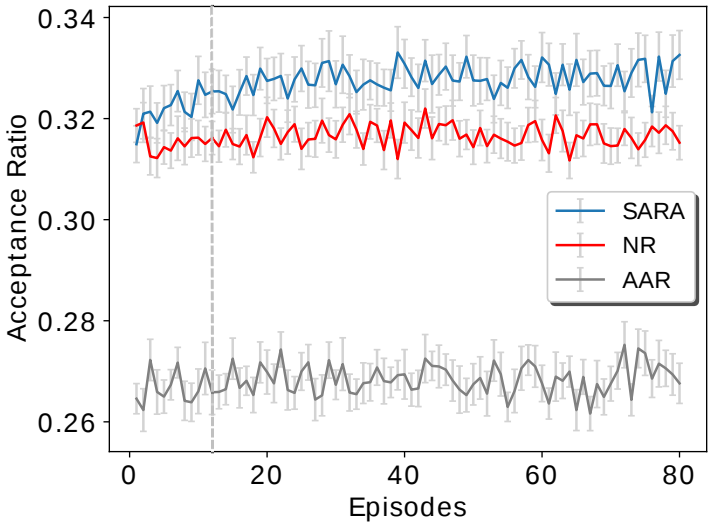


Figure 3.10 Acceptance Ratio on a 16-node topology

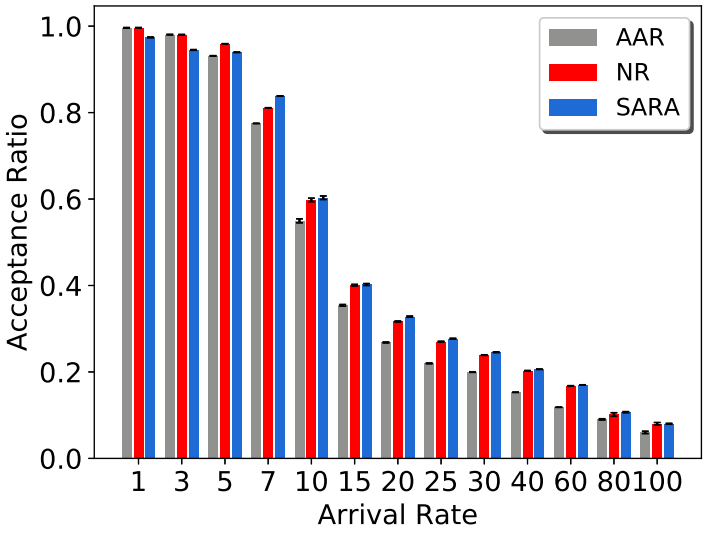


Figure 3.11 Acceptance Ratio vs Arrival Rate on a 16-node topology

Figure 3.10 shows the acceptance ratio as a function of time. Before converging, SARA produced acceptance ratios greater than those produced by AAR and slightly higher than those given by NR. After converging, SARA produced acceptance ratios 1% and 6% higher than NR and AAR, respectively. The difference is not significant since SARA prioritizes NSLRs that increase the profit which may cause the rejection of some of the less profitable NSLRs.

Figure 3.11 depicts the acceptance ratio as a function of the arrival rate. The three algorithms produced low acceptance ratios when the arrival rate is high because of the

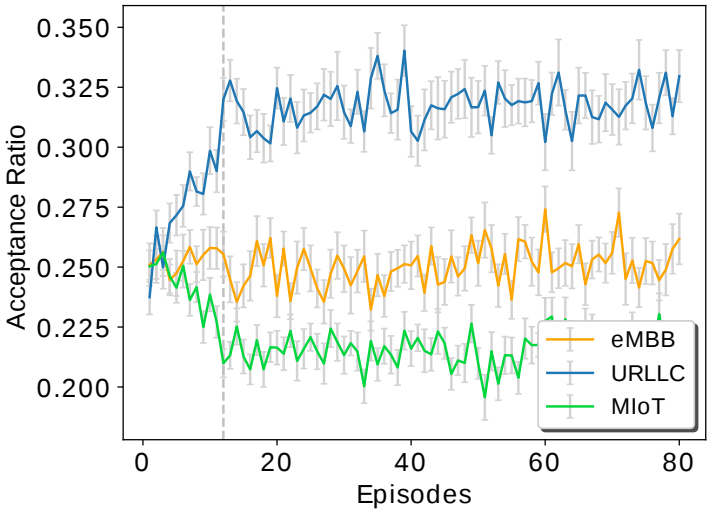


Figure 3.12 SARA - Acceptance per NSLR Type on a 16-node topology

limited resources in the substrate. When the arrival rate is higher than 7 requests per time unit, SARA produced the highest acceptance ratio, but the difference is still negligible. The reason for such difference is that the primary goal of SARA is to increase the profit and not the acceptance ratio. Several factors influence the profit: the cost, the quantity of accepted NSLR, and the operational time of resources consumed by these NSLRs. If SARA accepts NSLRs that need many resources and have long operational times, the substrate becomes busy for a long time, which prevents the acceptance of incoming NSRL. In general, the acceptance ratio results do not negatively impact on the profit since the RL-agent learns to prioritize the requests that increase the profit (see Figure 3.8).

Figure 3.12 presents SARA's acceptance ratio per service type. For URLLC NSLR, SARA's acceptance ratio grew from the first episode to the convergence point, achieving the maximum value. After that, the number of admitted eMBB and MIoT NLSR decreased over time because the RL-agent learns that these types of requests are less profitable (mainly the MIoT requests). Figures 3.12 and 3.9 corroborate that SARA accepts a higher proportion of NSLRs of type URLLC than the other types of service which give less profitability.

Figure 3.13 presents the network resource utilization as a function of time for an arrival rate of 20 requests per time unit. The resource utilization produced by NR and AAR did not evolve because these heuristics make decisions without learning from the environment. Conversely, the utilization produced by SARA grew up fast from the episode 1 to 12. After converging, the RL-agent obtained the maximum resource utilization, which is 5% higher than that achieved by NR.

Figure 3.14 depicts the network resource utilization obtained by SARA for the two types of nodes, and links. The green, orange, and brown lines correspond to the utilization of core nodes, edge nodes, and links, respectively. The blue line corresponds to the total resource utilization.

The utilization of core nodes slightly decreased after convergence. This happened because the mean utilization of edge nodes increased from 37% on the episode 1 to 49% after episode 12. The RL-agent accepts more NSLRs of URLLC than NSLRs of type

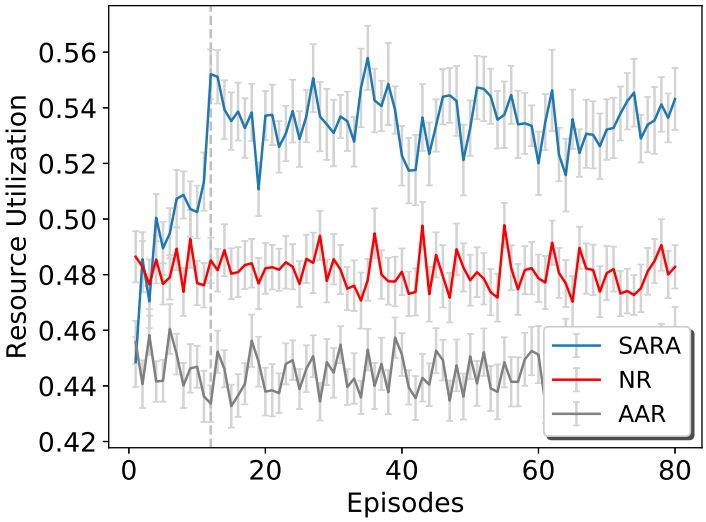


Figure 3.13 Resource Utilization on a 16-node topology

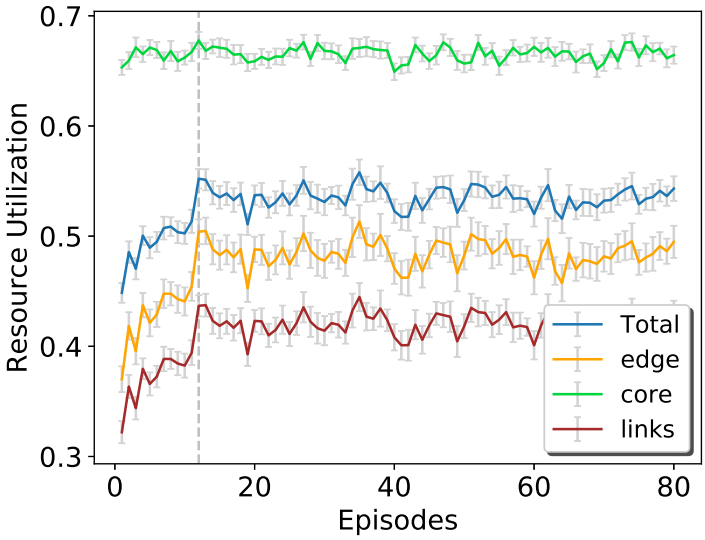


Figure 3.14 SARA - Utilization per Node Type on a 16-node topology

MIoT and eMBB. Remember that URLLC requests use many edge resources to meet latency requirements. As a result, profit increases (Figure 3.7).

3.5.4 Results for a 32-node topology

Figure 3.15 depicts the profit obtained in the 32-node topology as a function of time for an arrival rate of 20 requests per time unit. SARA increased the profit from episodes 1 to 10 when it converged. SARA produced higher profit values than NR and AAR even before converging (for most of episodes). Overall, the profit produced by SARA is 4.1% and 7.4% higher than those obtained by NR and AAR, respectively. Again, these profit values are due to SARA uses an RL-algorithm that quickly learns the most profitable combination of NSLRs. It is noteworthy to mention that on the 16-node topology with 20 requests per time unit, SARA reached a higher difference compared to the benchmark (*i.e.,* 7% respect to NR and 11.3% respect to AAR). This can be explained by the fact

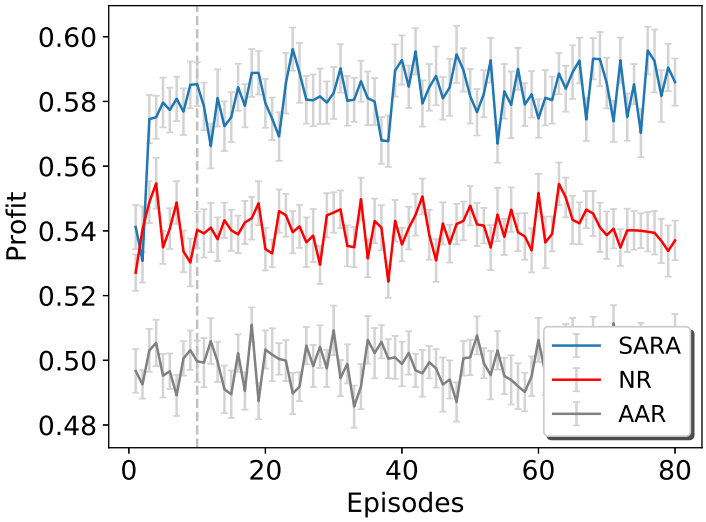


Figure 3.15 Profit comparison on a 32-node topology

that the topology with 32 nodes has a larger amount of resources than the topology of 16 nodes, thus, the former needs a higher quantity of instantiated NSLs (*i.e.,* triggered by higher arrival rates) to generate similar profit results than the latter.

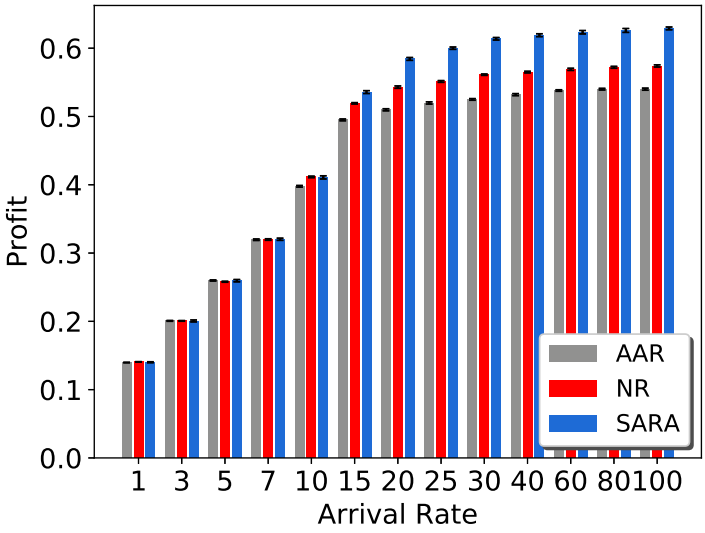


Figure 3.16 Profit vs Arrival Rate on a 32-node topology

Figure 3.16 shows the profit obtained as a function of the total arrival rate on the 32-node topology. Under low loads (*e.g.,* 1 requests per time unit), SARA obtained lower profit values than those given by the other two heuristics. If the arrival rate is too low, the number of arrivals of NSLRs is not enough to provide useful information to the RL-agent. On the other hand, for arrival rates upper than 10 requests per time unit, SARA obtained the highest profit. For example, for 15 requests per time unit, the profit of SARA is 1.6% higher than the profit produced by NR and for 100 requests per time unit, the difference raised to 5.6%. Anew, SARA outperforms AAR and NR under medium to high loads because it learns to accept the proper proportion of each NSLR type to increase the profit.

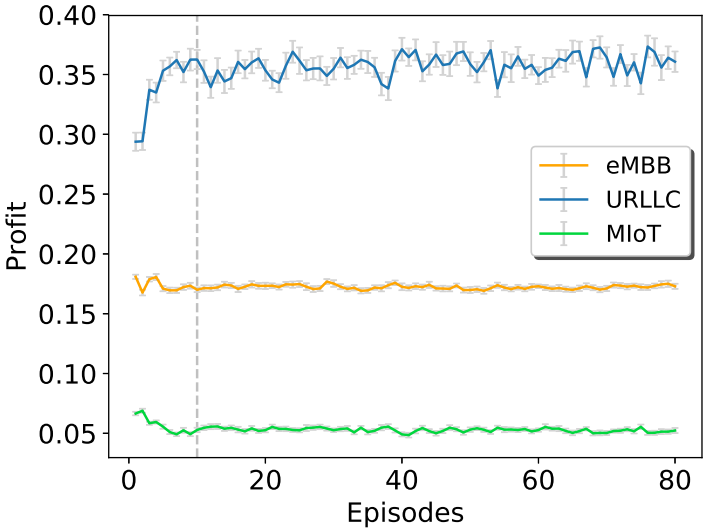


Figure 3.17 SARA - Profit per NSLR Type on a 32-node topology

Figure 3.17 presents the contribution of each type of admitted NSLR to the total profit obtained in the 32-node topology by SARA for a load of 20 requests per time unit. The contribution of URLLC NSLRs to the total profit grew from episodes 1 to 10, when SARA obtained the maximum profit. Conversely, the profit of NSLRs of eMBB and MIoT slightly decreased from episodes 1 to 10. These results confirm SARA learns to identify the NSLRs that generate high profit.

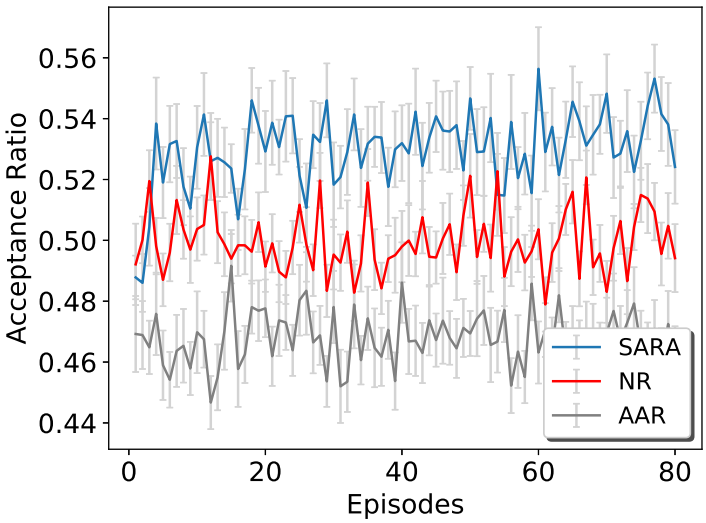


Figure 3.18 Acceptance Ratio on a 32-node topology

Figure 3.18 shows the acceptance ratio obtained in the 32-node topology as a function of time for a load of 20 requests per time unit. Before converging, SARA produced acceptance ratio greater than those produced by AAR and NR. After converging, SARA produced acceptance ratio 3.2% and 7.1% higher than the obtained by NR and AAR, respectively. As in the 16-node topology, the difference is low since SARA does not aim to increase the acceptance ratio but the profit.

Figure 3.19 depicts the acceptance ratio obtained in the 32-node topology as a function

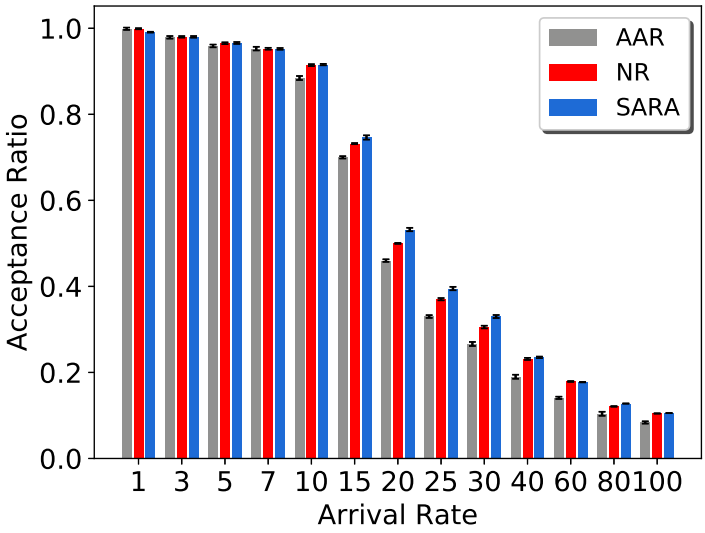


Figure 3.19 Acceptance Ratio vs Arrival Rate on a 32-node topology

of the arrival rate. The three algorithms produced low acceptance ratios when the arrival rate is high due to the limited resources in the substrate. When the arrival rate is higher than 10 requests per time unit, SARA produced the highest acceptance ratio but the difference is still negligible. As mentioned earlier, this happens because SARA targets to increase the profit and not the acceptance ratio.

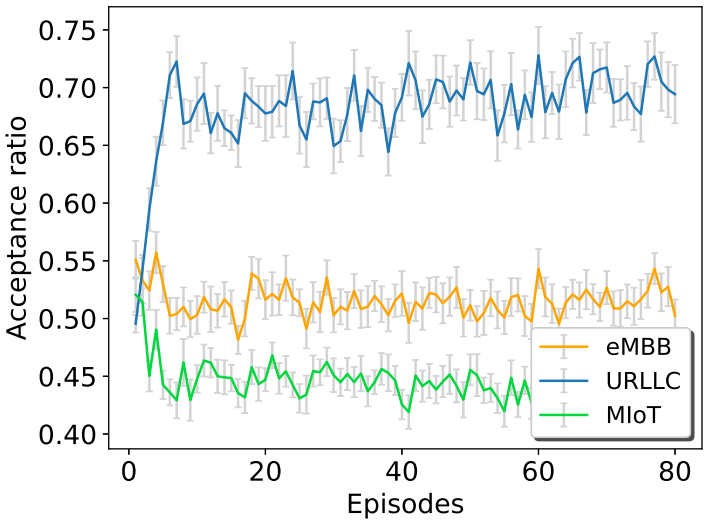


Figure 3.20 SARA - Acceptance per NSLR Type on a 32-node topology

Figure 3.20 presents the acceptance ratio per service type obtained by SARA with a load of 20 requests per time unit. For URLLC NSLRs, this ratio grew from the first episode to the convergence point, achieving the maximum value. After converging, the number of admitted eMBB and MIoT NSLRs diminished over time because the RL-agent learns that these types of requests are less profitable. Figures 3.9, 3.12, 3.17, 3.20 confirm that SARA accepts a higher proportion of NSLRs of type URLLC than the other types of service which give less profitability.

Figure 3.21 presents the network resource utilization results obtained in the 32-node

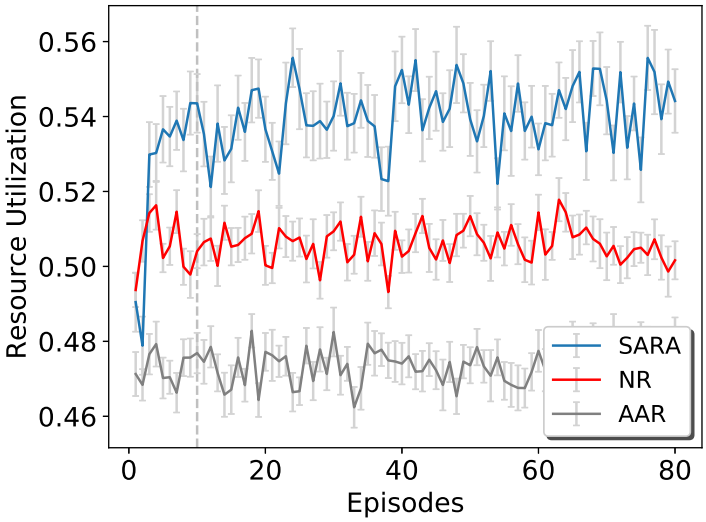


Figure 3.21 Resource Utilization on a 32-node topology

topology by the three algorithms as a function of time for an arrival rate of 20 requests per time unit. As expected the resource utilization produced by NR and AAR did not evolve because these heuristics make decisions without learning from the environment. Contrarily, the utilization produced by SARA grew up fast from the episode 1 to 10. After converging, the RL-agent obtained the maximum resource utilization, which is 3% higher than the achieved by NR.

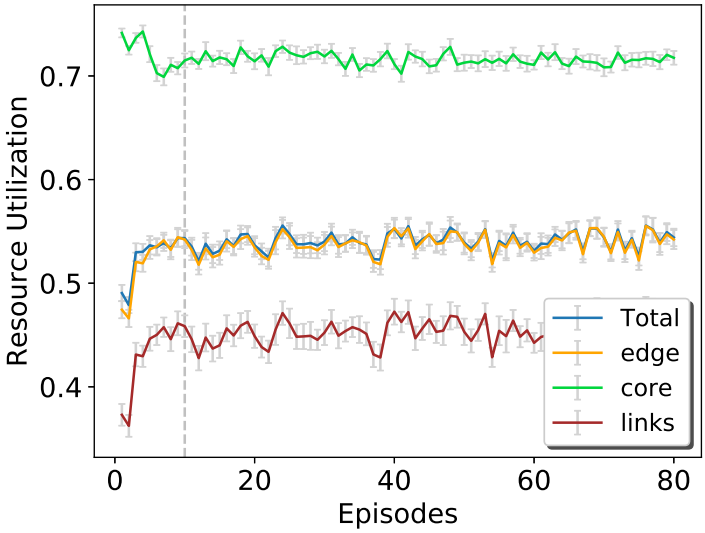


Figure 3.22 SARA - Utilization per Node Type on a 32-node topology

Figure 3.22 depicts the network resource utilization obtained by SARA in the 32-node topology for different types of nodes, and links. The green, orange, and brown lines correspond to the utilization of core nodes, edge nodes, and links, respectively. The blue line corresponds to the total resource utilization. The utilization of core nodes slightly decreased after convergence due to the mean utilization of edge nodes increased from 49% on the episode 1 to 54% after episode 10. The RL-agent prioritizes more NSLRs of URLLC than NSLRs of type MIoT and eMBB in order to increase the profit (Figure

3.15).

3.5.5 Results for a 64-node topology

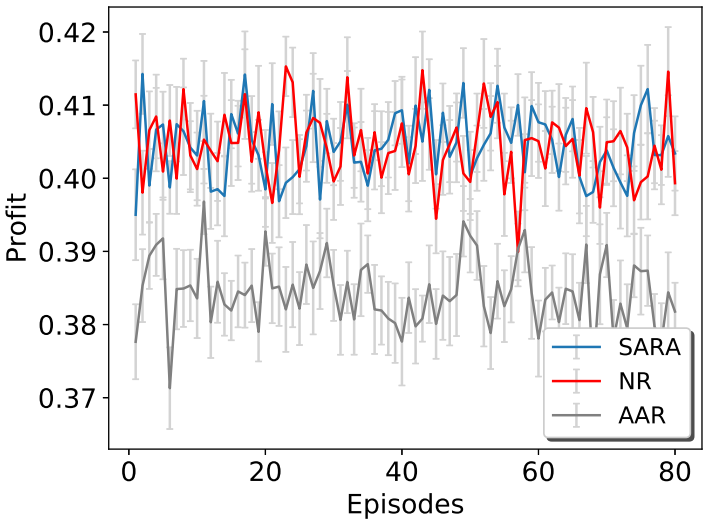


Figure 3.23 Profit comparison for 20 requests per time unit on a 64-node topology

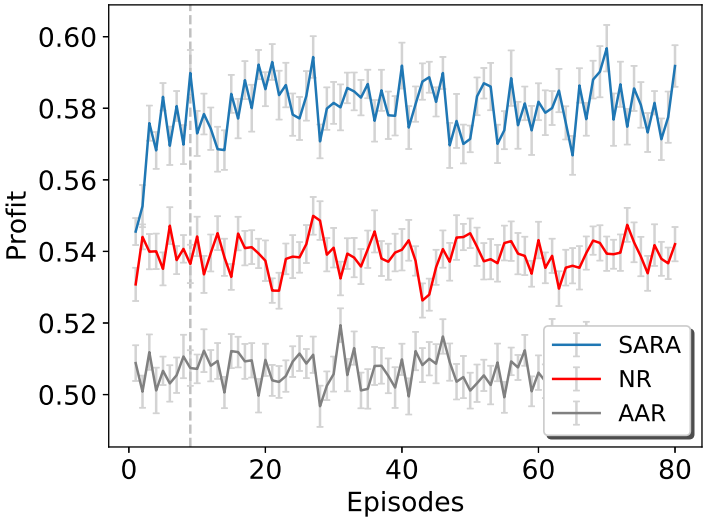


Figure 3.24 Profit comparison for 40 requests per time unit on a 64-node topology

Figure 3.23 depicts the profit results on the 64-node topology with an arrival rate of 20 requests per time unit. Here, the profit of SARA does not show evolution along the episodes and its values are similar to those produced by NR. This can be explained since a 64-node topology has a larger amount of resources than a 16-node or a 32-node topology and therefore, 20 requests per time unit are not enough for producing strong rewards that allow the RL-agent to learn. In this sense, we show in Figure 3.24 the profit obtained by SARA for an arrival rate of 40 requests per time unit. SARA increased the profit from episodes 1 to 9 when it converged. It produced higher profit values than NR and AAR even before converging. Overall, the profit produced by SARA is 4.6% and 8.4% higher

than those obtained by NR and AAR, respectively. Once again the RL-based algorithm of SARA quickly learned the most profitable combination of NSLRs.

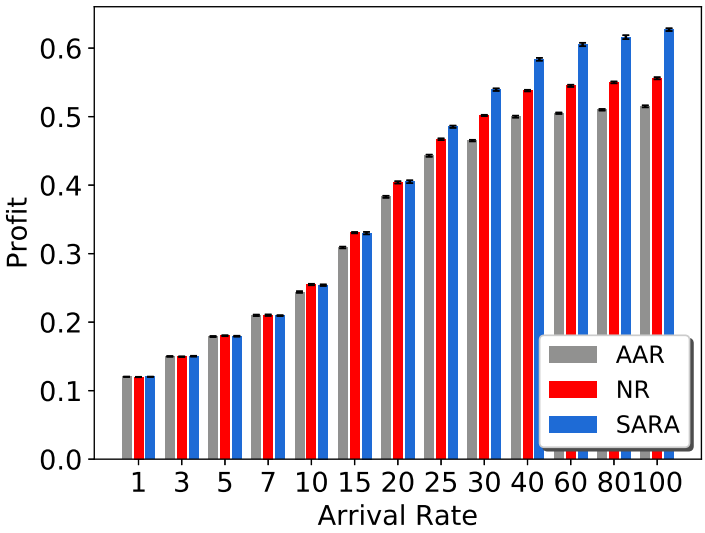


Figure 3.25 Profit vs Arrival Rate on a 64-node topology

In figure 3.25, we show the profit obtained by SARA, NR, and AAR on the 64-node topology when varying the load from 1 to 100 requests per time unit. SARA exhibits marked higher profit values than the benchmark after 20 requests per time unit where the difference is 1.8% in relation to NR and 4.2% in relation to AAR. For an arrival rate of 100 requests per second, such differences raised to 7.1% with respect to NR and 11.2% compared to AAR.

It is important to highlight that for 16-node and 32-nodes topologies SARA overcomes the benchmark after 5 and 10 requests per time unit, respectively. This behaviour can be explained since for the case of topologies with lower number of nodes, and hence, less amount of resources than larger topologies, low loads allow occupying enough resources to produce sufficiently strong rewards to favor the learning. In this sense, large topologies need a higher load to generate profit results similar to those produced by small topologies with a lower load.

Figure 3.26 presents the evolution, along the episodes, of the profit contributed by each type of admitted NSLR to the total profit obtained on the 64-node topology with a load of 40 requests per time unit. Before the convergence point, URLLC profit grows while the profit of eMBB and MIoT slightly decreases. These results confirm that SARA learns to identify the type of NSLRs that generate high profit.

In figure 3.27, we show the acceptance ratio obtained in the 64-node topology as a function of time for a load of 40 requests per time unit. After converging, SARA produced a higher acceptance ratio than NR and AAR, in specific, 1% and 4%, respectively. As mentioned in the evaluations for 16-node and 32-node topologies, such a difference is low since SARA does not aim to increase the acceptance ratio but the profit.

Figure 3.28 depicts the acceptance ratio obtained in the 64-node topology when varying the arrival rate. The three algorithms produced high acceptance ratios for low loads. When the arrival rate grows, such ratios decrease. For arrival rates higher than 25 requests

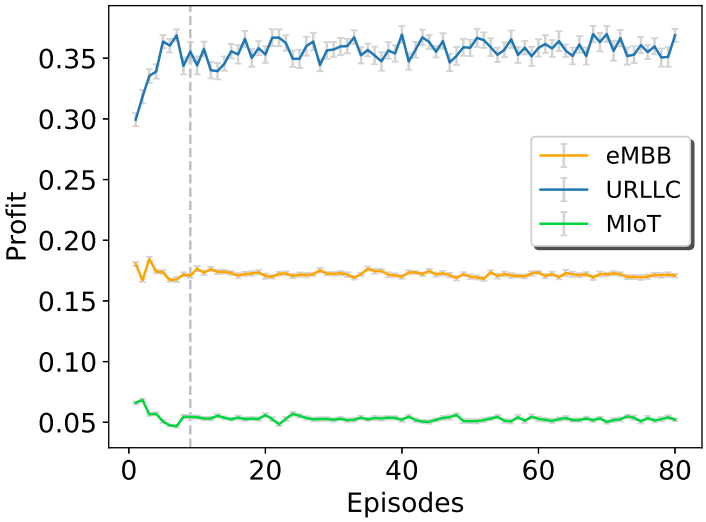


Figure 3.26 SARA - Profit per NSLR Type on a 64-node topology

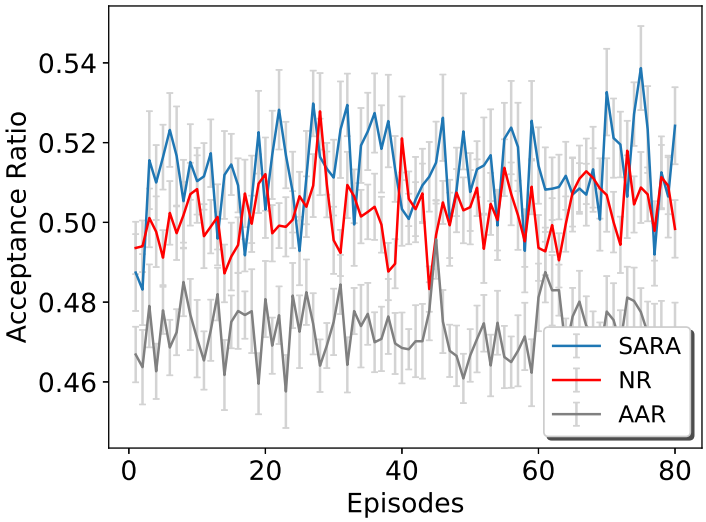


Figure 3.27 Acceptance Ratio on a 64-node topology

per time unit, SARA produced the slightly higher acceptance ratios than the benchmark, however, the difference is negligible. As mentioned earlier, this is explained since the goal of SARA is to increase the profit and not the acceptance ratio.

In figure 3.29, we present the acceptance ratio of each service type obtained by SARA with a load of 40 requests per time unit. Before the convergence, the acceptance ratio of URLLC NSLRs grows until achieving the maximum value while the ratios of eMBB and MIoT NSLRs decrease since the RL-agent learns that these types of requests are less profitable. Figures 3.9, 3.12, 3.17, 3.20, 3.26, and 3.29 confirm that SARA accepts a higher proportion of NSLRs of type URLLC than the other types which give less profit.

Figure 3.30 presents the network resource utilization results obtained in the 64-node topology by the three algorithms as a function of time for an arrival rate of 40 requests per time unit. As expected the resource utilization produced by NR and AAR did not evolve because these heuristics make decisions without learning from the environment. In contrast, the utilization produced by SARA grew up fast in the episodes before conver-

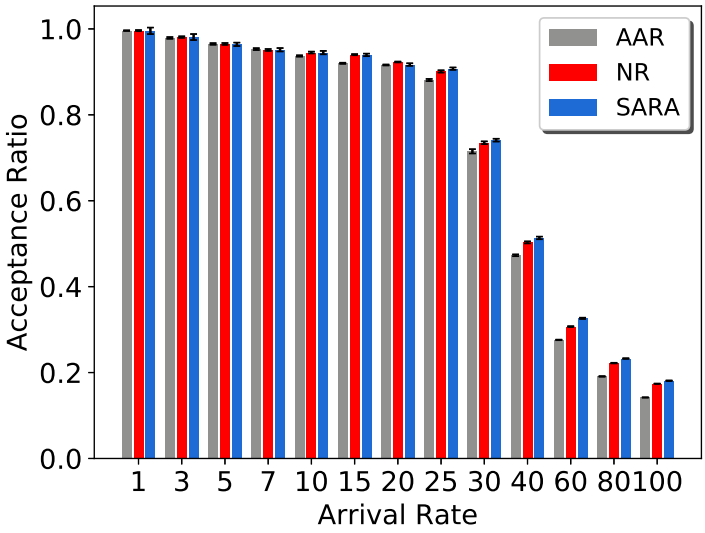


Figure 3.28 Acceptance Ratio vs Arrival Rate on a 64-node topology

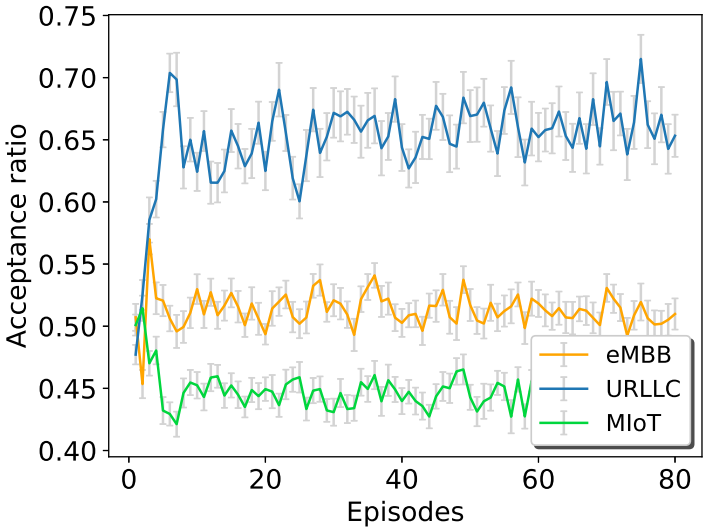


Figure 3.29 SARA - Acceptance per NSLR Type on a 64-node topology

gence. After converging, the SARA obtained a profit 3% higher than the achieved by NR and 7% greater than AAR.

In figure 3.31, we depict the network resource utilization obtained by SARA in the 64-node topology for each type of node, and links. The green, orange, and brown lines correspond to the utilization of core nodes, edge nodes, and links, respectively. The blue line corresponds to the total resource utilization. After convergence, the utilization of core nodes has decreased while the mean utilization of edge nodes has increased and reaches 54%. SARA prioritizes more NSLRs of URLLC than those of MIoT and eMBB in order to increase the profit as shown in (Figure 3.15).

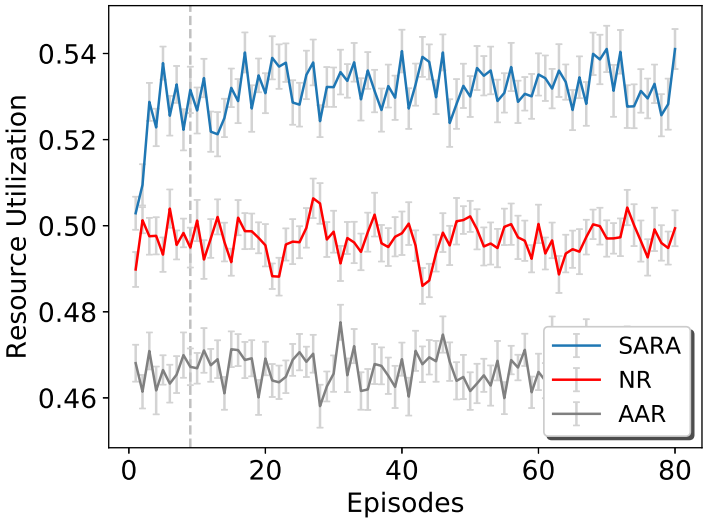


Figure 3.30 Resource Utilization on a 64-node topology

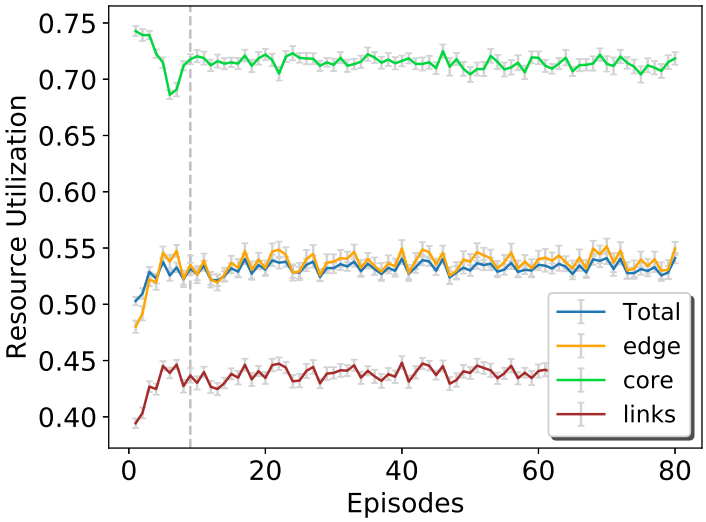


Figure 3.31 SARA - Utilization per Node Type on a 64-node topology

3.6 Summary

In this chapter, we introduced SARA, an approach that jointly performs admission control and resource allocation for NSLRs of eMBB, URLLC, and MIoT service types in 5G. The RL-based algorithm of SARA prioritizes the most profitable NSLRs in a set of them that arrived in a time window. SARA employs Q-learning, a model-free RL algorithm, meaning that SARA does not make assumptions about the substrate network as do optimization-based approaches. SARA learns while exploring the environment without a prior-knowledge about the substrate. SARA adapts to changes in the environment since its online operation (*i.e.,* input data is provided sequentially while the algorithm is running) allows learning continually from the interaction with the environment.

The evaluation results show that SARA outperforms the benchmark when testing on topologies of different sizes. For the 16-node topology, SARA achieved up to 7% and 11.3% higher profit than the profit given by the NR and AAR algorithms, respectively.

Moreover, SARA utilized 5% and 10% more resources than did NR and AAR, respectively. Utilization results are consistent with profit results since SARA learned to admit the most profitable requests, which are those that use more resources. Results on the 32-node topology show that SARA reached up to 5.6% and 9% higher profit than the profit given by the NR and AAR algorithms, respectively. SARA utilized 3.5% and 7.2% more resources than did NR and AAR, respectively. For tests on the 64-node topology, SARA obtained up to 7.1% and 11.2% higher profit than the profit generated by the NR and AAR, respectively. Furthermore, SARA reached 3.2% higher utilization than NR and 6.8% higher than AAR.

Results presented in this chapter corroborate that the RL-based strategy adopted by SARA, to manage admission and resource allocation for 5G NSLRs, is suitable for optimizing the profit to NSPs.

Chapter 4

Admission Control for 5G Network Slices based on Deep Reinforcement Learning

5G is conceived to support myriads of specialized services accessible on-demand from a vast number of devices by multiple customers [2, 54]. Network Slicing is the technology that allows 5G to build and run several isolated and customized NSLs over a single network infrastructure to support such specialized services [49]. NSPs will receive several NSLRs from multiple tenants. Since each NSLR has particular QoS requirements associated to its service type (eMBB, URLLC, or MIoT) and substrate resources are finite, NSPs face the challenge of admitting NSLRs and allocating resources to them, aimed at increasing its overall profit and optimize the network resource utilization.

In the literature, admission control and resource allocation for slicing requests have been addressed by employing different techniques such as Queuing Theory [33, 3], Big Data [50], Heuristics [37, 55, 64, 41], Complex Network Theory [27], Reinforcement Learning [49, 12], and more recently, Deep Reinforcement Learning [11, 40]. However, on the one hand, some of those proposals have considered only resource allocation and neglected admission control, which prevents the achievement of NSP goals.

On the other hand, approaches that consider admission control make decisions on individual NSLRs, which can lead to sub-optimal decisions since more profitable requests arriving in the short term can be rejected due to the unavailability of resources recently allocated [24]. Moreover, most of these proposals neither consider the QoS requirements of different service types (use cases) nor the allocation of resources in 5G core network nodes. Performing admission control and resource allocation jointly and intelligently can considerably improve the achievement of target objectives.

In Chapter 3, we proposed SARA, a Q-learning based solution for admission control of NSLRs and their resource allocation. We demonstrated that SARA outperforms two heuristics in terms of profit and resource utilization. SARA can produce even higher profit values if a larger and more accurate scenario is provided, for instance, more features to represent the states and more actions for the agent to explore. However, such enhancement is achieved at the cost of a longer convergence time since the agent has to experience more state-action pairs many times before obtaining a reliable estimation of their Q-values.

Furthermore, strong memory characteristics are needed for storing a larger amount of Q-values. These limitations are typical of Q-learning solutions due to the use of the Q-table to store Q-values [11].

Deep Q-learning (DQN), a Deep Reinforcement Learning (DRL) technique, arises to deal with Q-learning limitations. DQN uses a function approximator for generalizing the knowledge learned from some already visited states to other similar states. Such generalization allows the agent to learn from a reduced number of interactions with the environment, and therefore, the algorithm converges fast [58].

Aimed at improving further the results achieved by SARA, this chapter introduces DSARA (DRL-based network **S**lice requests **A**dmission and **R**esource **A**llocation). DSARA addresses admission control and resource allocation for 5G NSLRs. DSARA employs a DRL-agent to explore, exploit, and learn to admit NSLRs collected in time windows to increase the profit of NSPs.

The organization of this chapter is as follows. Section 4.1 presents the related work. Section 4.2 introduces the architecture of DSARA. Section 4.3 presents and describes in detail the DRL-based admission control algorithm of DSARA. Section 4.4 presents the evaluation of DSARA. Finally, section 4.5 summarizes this chapter.

4.1 Related Work

In this section, we present research related to admission control and resource allocation in 5G Network Slicing. Table 4.1 summarizes approaches and its features.

Multiple works have addressed admission control and resource allocation in 5G network slicing by different techniques including Queuing Theory [33, 3], Big Data [50], Heuristics [37, 55, 64, 41], Complex Network Theory [27], and RL [49, 12]. Recently, DRL has also gained attention in this field [11, 40]. Bega et al. [11] propose a DRL-based algorithm that performs admission control for individual slicing requests, focused on RAN, aimed at maximizing the monetization of the infrastructure provider. Li et al. [40] investigate the application of DRL in 5G network slicing for a radio resource slicing scenario. In particular, authors propose an algorithm that allocates the bandwidth resource of a slice to the users within the slice.

Approaches [33], [50], [55], [37], [49], [12] and [11] use mechanisms for admission control that make decisions on individual requests, which prevents the selection of the NSLRs that potentially optimizes an objective such as the profit of NSPs. These papers do not consider different types of requests according to 5G use cases, neglecting the diversity of QoS requirements of 5G service types. Moreover, they focus on edge nodes and neglects 5G core network nodes (Table 4.1).

In [64], [41], [3], [27], and [40], authors focus only on mapping NSLs. They map the arriving NSLRs without considering admission control which prevents the achievement of NSP goals. Performing both admission control and resource allocation jointly in 5G Network Slicing is critical to optimize resource utilization and maximize NSP profit. This task is challenging since it involves making highly repetitive decisions [43]. RL algorithms are well-suited candidates for decision-making problems; RL employs Markov Decision

Table 4.1 Related Work

Work	Technique	Focus		Time Window	Fifth Generation			Performance Metrics
		AC	RA		Use Cases	Edge Nodes	Core Nodes	
[49]	RL (Q-learning)	✓				✓		Provider profit
[33]	Queuing theory	✓				✓		Utility rate, admission rate, request waiting time
[50]	Big Data Analytics	✓				✓		Provider profit
[37]	Heuristic algorithm	✓	✓			✓		QoE, resource utilization
[55]	Heuristic algorithm	✓				✓		System resource utilization
[12]	RL (Q-learning)	✓				✓		Provider revenue
[11]	DRL (Deep Q-learning)	✓				✓		Provider revenue
[64]	Heuristic algorithm		✓		✓		✓	Acceptance Ratio and Execution Time
[41]	Heuristic algorithm		✓				✓	Acceptance ratio, provider revenue
[3]	Queuing Theory		✓				✓	Running time
[27]	Complex Network Theory		✓		✓	✓	✓	Resource efficiency, acceptance ratio, execution time
[40]	DRL (Deep Q-learning)		✓			✓		Spectrum efficiency and QoE
DSARA	DRL (Deep Q-learning)	✓	✓	✓	✓	✓	✓	Provider profit, resource utilization, acceptance ratio

Process, an efficient tool to solve such kind of problem [40]. Moreover, such repetitive decisions produces a large quantity of data that can be used to train RL-agents.

Our approach, DSARA, learns from a set of NSLRs collected during a time window, the most profitable ones. DSARA is model-free since it does not make assumptions about the environment but learns while exploring it without a prior-knowledge. DSARA operates online which allows to learn continually from the environment and therefore adapt to changes. DSARA enables to generalize knowledge from a few past experiences to converge fast. To the best of our knowledge, no other work has proposed a solution for 5G Network Slicing based on DRL that jointly performs admission control and resource allocation, considers typical use cases in 5G, and differentiates core and edge nodes.

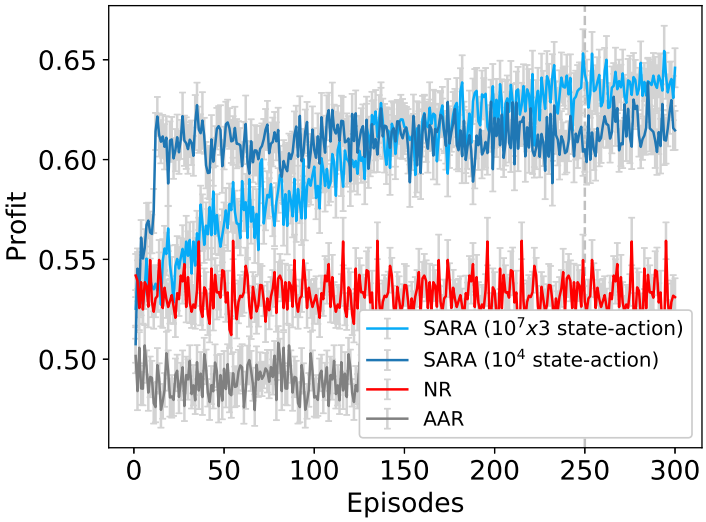


Figure 4.1 Profit of SARA for a large scenario

4.2 Architecture of DSARA

This section first present a brief motivation to use Deep Q-learning, and second, it describes the architecture of DSARA, illustrated in Figure 4.2.

4.2.1 From Q-learning to Deep Q-learning

In Chapter 3, we proposed SARA, a Q-learning based solution for admission control of NSLRs and their resource allocation. We demonstrated that SARA outperforms two heuristics in terms of profit and resource utilization. Nonetheless, the profit of SARA can be further increased if we consider: additional features to represent states (*e.g.,* the number of NSLs of each type in operation) to get a more accurate model of the environment, and a larger Action Space, *i.e.,* a higher number of actions to encompass more options that lead the agent to find more optimal solutions.

Figure 4.1 depicts the scenario mentioned above. The dark-blue line shows the profit results obtained by SARA in Chapter 3, *i.e.,* for a scenario with 10^3 states and 10 actions (10^4 state-action pairs). Results correspond to tests on the 16-node topology with a load of 20 requests per time unit. The light-blue line presents the profit results achieved by SARA for a scenario in which we increased the states to 10^6 and the number of actions to 30 (3×10^7 state-action pairs). Results show that in the new scenario, SARA reaches a maximum profit that is 3% higher than the obtained with the previous scenario. However, its convergence takes place at 238 episodes after. NSPs are missing the opportunity to generate higher profits in such initial episodes.

The performance of SARA can be improved for large scenarios (*i.e.,* scenarios with a high amount of state-action pairs) by facing Q-learning limitations. In Q-learning, the agent selects an action (a_t) under a state (s_t) and receives from the environment a reward (R_t) and a new state (s_{t+1}) [58, 40]. Q-learning stores the quality (Q-value) of an action, in a given state, by using a lookup table (Q-table). Q-learning behaves very stably for moderate amounts of state-action pairs. For large scenarios, Q-learning approaches may

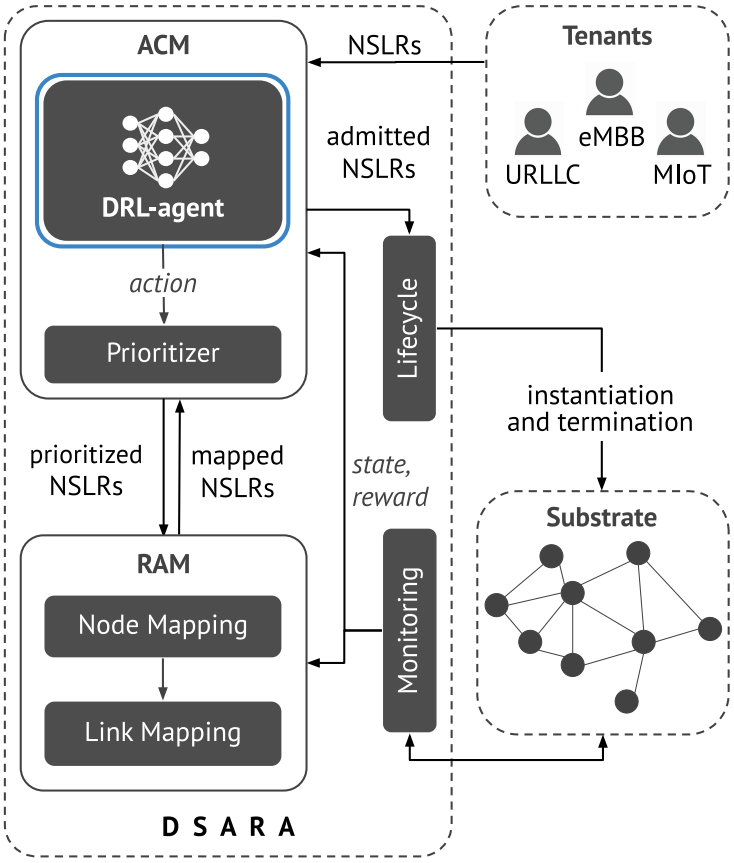


Figure 4.2 Architecture of DSARA

present serious limitations derived from the use of the Q-table to store the Q-values [11]. Such limitations are: *(i)* large convergence time since the agent needs to experience many times the same state-action pairs before obtaining a reliable estimation of their Q-values; and *(ii)* strong memory characteristics needed to store such a large amount of Q-values.

In this regard, DQN is a well-known alternative to deal with the limitations of Q-learning. DQN enables the agent to learn from a reduced number of interactions with the environment by applying the knowledge learned from some already visited states to other similar states. This is known as generalization, improves convergence, and can be performed by Artificial Neural Networks (NN), strong approximators for non-linear functions [36].

4.2.2 Modules

The environment on which DSARA operates consists of a substrate network and several incoming 5G NSLRs processed in batches collected in time windows. These elements follow the same considerations as in SARA described in Chapter 3.

DSARA is composed of: the *DRL-based Admission Control Module (ACM)* that performs the admission of NSLRs, the *Resource Allocation Module (RAM)* that assigns substrate resources to VNFs and virtual links of an NSLR, the *Monitoring Module* in charge of collecting and delivering information about substrate resources availability to ACM and RAM, and the *Lifecycle Module* that creates and terminates NSLs.

ACM consists of a DRL-agent that selects an action indicating a normalized weight

value for each service type and a Prioritizer that sorts the NSLRs according to the values provided by the agent. The DRL-agent learns to select the actions that generate the highest profit to NSPs by looking at previous experiences. Each experience consists of a past decision (*i.e.,* the action selected) made in a particular situation and its consequence (*i.e.,* the profit resulted by taking the selected action). Experiences are stored into a database (replay memory) and then revisited to extract knowledge from them to be applied to similar situations in the future. The operation of ACM and its specificities are formally discussed in Section 4.3.

The Prioritizer is in charge of enqueueing NSLRs in batches of minimum size obeying the arrival time and the proportion given by the weight values provided by the agent. For instance, if weight values are 1.0, 0.5, and 0.5 for the eMBB, URLLC and MIoT, respectively, then batches with 2, 1, and 1 NSLRS of type eMBB, URLLC and MIoT are enqueued. NSLRs per class are enqueued in chronological order. If all the NSLRs of a particular type have been enqueued, the weight values of the other service types are used to determine the number of NSLRs in the subsequent batches. In the example just described, if there are 10 NSLRs per service type, in the queue, there will be a sequence of 5 batches with 2, 1, and 1 NSLRs of type eMBB, URLLC, and MIoT, followed by 5 batches composed by one NSLR of URLLC and one of MIoT type.

Once the priority queue has been assembled, each NSLR is dequeued, and the ACM sends a request for allocation of resources to RAM. If resources are successfully allocated, then the NSLR is accepted. Otherwise, it is rejected. The dequeuing of an NSLR and the attempt to allocate resources to it is repeated until the priority queue is empty.

RAM allocates resources on nodes for the VNFs composing an NSLR (node mapping), and then, allocates bandwidth in selected links (link mapping) connecting the allocated nodes. Decisions on node mapping consider not only if a node has resources available to support the demand but also the latency and reliability requirements of the NSLR type. Control Plane VNFs are mapped onto core nodes. User Plane VNFs of a URLLC request are mapped onto edge nodes to satisfy strict latency requirements, while the User Plane VNFs of the other two types are mapped, preferably on core nodes. For complying with reliability requirements, a backup VNF is not placed onto the same node on which its primary VNF is placed.

Link Mapping maps each virtual link onto the shortest substrate path that satisfies the required bandwidth by the virtual link. If Node Mapping and Link Mapping finish successfully, RAM sends a notification of successful allocation (mapped) to ACM. Otherwise, a non-mapped notification is sent. RAM in DSARA has no modifications with respect to its counterpart in SARA.

Upon accepting an NSLR, the Lifecycle module instantiates its VNFs and virtual links, creating, then, an NSL. When the lifetime of the NSL expires, resources are deallocated.

4.3 Admission Control based on Deep Q-learning

ACM in DSARA is based on DQN. In this section, we firstly review DQN. Then, we describe our DRL-agent by defining the elements that specify it. Finally, we introduce

and detail our admission control algorithm. For allocating resources to NSLRs, we use the algorithm described in Chapter 3.

4.3.1 Deep Q-learning

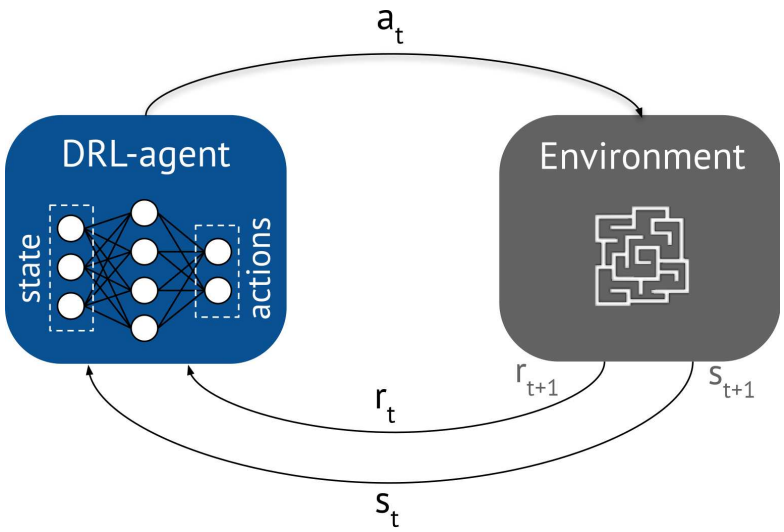


Figure 4.3 Deep Q-learning operation

DQN is a DRL algorithm, a category of Machine Learning that takes advantage of both RL and Deep Learning [40] (see Figure 4.3). On the one hand, RL enables a self-learning agent to maximize long-term performance by interacting with the environment and receiving feedback from it. On the other hand, inspired by biological neural networks, Deep Learning allows RL to manage problems with large state and action spaces by generalizing knowledge [25, 10]. Generalization allows to apply the knowledge learned from some already visited states to other similar states. Linear function approximation is the most straightforward generalization technique [44]. However, it could not accurately model the estimated policy function [40]. In this regard, researches have used NNs, strong approximators for non-linear functions, to perform generalization [36].

DQN uses an NN to generalize the experience learned. NNs are structures formed by a set of highly interconnected units known as neurons organized into layers. Each neuron is basically a mathematical operation that applies a weighted sum to the input, and then, it passes its output to the next neurons. Neurons learn by adjusting their weights based on a large number of examples containing inputs and desired outputs (*i.e.,* labeled datasets) [12]. In DQN, the NN estimates the Q-value (Q_i) of each action in the action space A. Then, the action taken will be the one with the best estimated Q-value. One of the advantages of using DQN is that its NN only needs to store a limited number of variables, *i.e.,* the weights and biases of the network structure, to accurately estimate Q-values [11].

Enhancements in DQN

Two techniques have been proposed for enhancing the learning stability in DQN [10, 45]:

- **Experience Replay.** The agent stores the past experience (s_t, a_t, s_{t+1}, R_t) at step t into a dataset (replay memory) to later take from it a mini-batch of experiences and train the NN. This allows to reduce the amount of interactions with the environment needed to learn compared to Q-learning.
- **Evaluation and Target Networks.** The agent uses two NNs to learn which action to take at every step. The evaluation NN estimates the Q-values ($Q(s_t, a_t)$) and is trained at each step. A second network, called fixed target NN (*i.e.,* its weights are temporarily fixed to enhance learning stability) is used for estimating the target Q-values (see Equation 4.1) that serves as a label to calculate the loss for training the evaluation network. Target NN is updated with the weights of the evaluation NN after a fixed number of steps since the latter is being trained iteratively.

$$Q^{+}(s_t, a_t) = R_t + \gamma \cdot maxQ(s_{t+1}, a) \tag{4.1}$$

Gradient Descent and Backpropagation

Training an NN involves reducing the loss by adjusting the NN's parameters (*i.e.,* weights and biases). The loss is the difference between the estimated output and the target output, also called error or cost function. In DQN, the loss (Equation 4.2) is calculated by the difference between the Q-value estimated by evaluation NN and the target Q-value obtained by target NN. The adjustment of weights and biases can be performed by using a gradient descent and backpropagation approach [52].

$$Loss = (Q^{+}(s_t, a_t) - Q(s_t, a_t))^2 \tag{4.2}$$

The Gradient descent algorithm is used to calculate new weights and biases that reduces the loss by moving it in the direction of the steepest descent. Gradient descent and backpropagation allow to adjust weights and biases of each layer in the NN by back-propagating the error calculated at the output layer.

4.3.2 DRL-agent for Admission Control

Figure 4.4 shows the internals of our DRL-agent that includes target and evaluation NNs, a memory for storing experiences (Replay Memory), and loss calculation and parameters update operations. This subsection describes the elements that specify our DRL-based solution: state space, action space, reward function, exploration and exploitation method, and finally the NNs used by the DRL-agent.

State Space

The State Space S is the set of all states the DRL-agent can observes. Each state describes the substrate resource availability as well as the amount of NSLs of each type in operation.

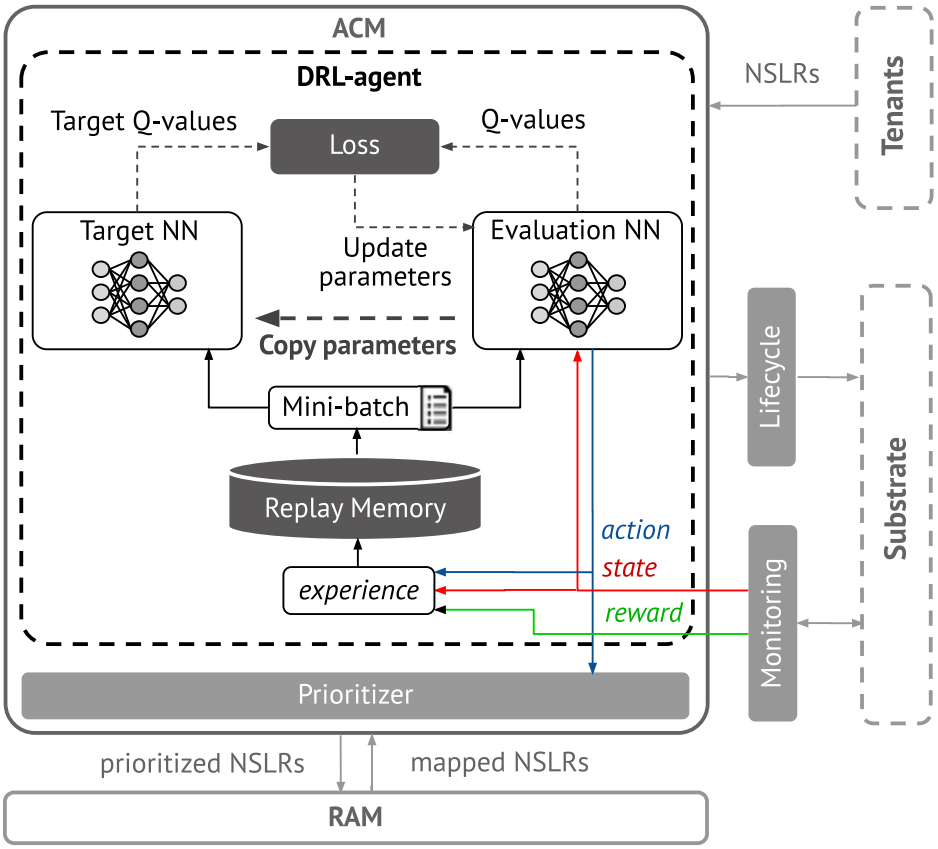


Figure 4.4 DRL-agent of DSARA

Every state $s \in S$ is denoted by the tuple: $\{cpu(E), cpu(C), bw(L), n_e, n_u, n_m\}$, where $cpu(E)$ and $cpu(C)$ are the available processing capacity in the set of edge (E) and core nodes (C), respectively. $bw(L)$ is the available bandwidth in the set of links (L). n_e, n_u, and n_m indicate the number of NSLs in operation of type eMBB, URLLC, and MIoT, respectively.

With the aim of keeping tractable the number of states, the capacity of substrate resources and the number of instantiated NSLs of each type are discretized in ten equal intervals. As our State Space considers six variables, and each can take ten values, the total number of states the DRL-agent can explore is 10^6.

To exemplify the State Space, consider the following state: $s_i = \{80, 50, 60, 30, 60, 10\}$. s_i indicates that 80% and 50% of the total capacity of processing is available in E and C, respectively, and 60% of the total capacity of bandwidth is available in L. Such a state also indicates that 30% of NSLRs currently running are of eMBB type. Similarly, 60% of NSLRs are of URLLC and 10% belongs to MIoT type.

Action Space

The Action Space A is the set of all actions the DRL-agent can take. In every step, the DRL-agent selects the action describing the weights of each type of service (*i.e.,* NSLR of eMBB, URLLC, and MIoT). Those weights enables the DRL-agent to learn what is the best admission proportion of NSLRs that leads to the highest profit for the current state. Every action $a \in A$ is denoted by $a = \{w_{embb}, w_{urllc}, w_{miot}\}$, where w_{embb}, w_{urllc}, and w_{miot}

are the weights for each type of service. In every step, the DRL-agent chooses the action a that allows to achieve the maximum profit.

To exemplify the Action Space, let us consider the following action: $a_i = \{1, 0.75, 0.5\}$. This action indicates that the DRL-agent of DSARA should prioritize NSLRs of eMBB. Specifically, it has to admit two NSLRs of MIoT and three of URLLC every four NSLRs of eMBB. We consider 30 different actions in the Action Space, *i.e.,* 30 different admission solutions the DRL-agent can explore to determine the optimum one for every state.

Reward

The reward guides the DRL-agent to get proper solutions for the proposed target. As we target to maximize the NSP profit, our DRL-agent receives a reward that contains the profit generated after it takes an admission action. Equation 4.3 indicates the profit p generated by each NSL accepted and instantiated during its operational time T_o. $p(nsl)$ is the difference between the revenue rev_i and the cost cst_i.

$$p(nsl_i) = (rev_i - cst_i) \times T_o \tag{4.3}$$

$$cst_i = \sum_{j=0}^{m} cpu_j(vnf_j) \times f_{cpuj} + \sum_{j=0}^{n} bw(v_j) \times f_{bw} \times h \tag{4.4}$$

where:

rev - is the income that an NSP receives for instantiating the nsl_i

cst - the cost of running nsl_i on the substrate

T_o - is the nsl_i operational time

m - gives the number of VNFs in nsl_i

n - gives the number of virtual links in nsl_i

$cpu(vnf_j)$ - is the cpu demand of $vnf_j \in nsl_i$

$bw(v_j)$ - is the bandwidth demand of $v_j \in nsl_i$

f_{cpuj} - is the cost of processing cost node j, which depends on the node type

f_{bw} - is the bandwidth cost

h - is the number of hops composing the path where virtual link v_j is allocated.

The reward R the DRL-agent receives at each step is given by:

$$R = \frac{\sum_{i=0}^{k} p(nsl_i)}{maxP(SN, T)} \tag{4.5}$$

where $maxP(SN, T)$ is the maximum profit that NSP could receive when using all the resources in the substrate (SN) in a certain period (T).

Exploration and Exploitation Trade-off

The DRL-agent of DSARA needs a method to choose an action at each step. We use an epsilon-greedy method that reduces the ε parameter progressively which allows the DRL-agent to explore less and less as episodes go on. The goal of this progressive reduction is to move the DRL-agent from a more explorative policy at the beginning to a more exploitative policy after a certain number of episodes [58].

In epsilon-greedy, the parameter ε determines how much the DRL-agent explores-exploits [60]. A small ε value enables the DRL-agent to select more optimal actions than random actions, as a result, it exploits more the current knowledge. In contrast, a large ε allows the DRL-agent to take more random actions than optimal ones; *i.e.,* it explores new actions more frequently.

$$\varepsilon_t = \varepsilon_{max} - (nsteps_t \times dr) \tag{4.6}$$

To choose an action at each step, our DRL-agent first decays ε by Equation 4.6, where ε_{max} is the maximum probability of exploration at the beginning of training. $nsteps_t$ is the number of steps the DRL-agent has experienced so far at step t. dr is the decay rate, a constant that allows reducing ε linearly over time.

$$a = \begin{cases} \max\limits_{\text{a}} Q_t(s_t, a), & with\ probability\ 1-\varepsilon_t \\ random\,action, & with\ probability\ \varepsilon_t \end{cases} \tag{4.7}$$

After decaying ε, the DRL-agent generates a random number $rn \in [0, 1]$, and then, compares rn against ε. If $rn > \varepsilon$, the DRL-agent selects the action with the maximum Q-value; otherwise, it chooses a random action (see Equation 4.7).

Neural Networks

The traditional DQN uses a single NN to estimate both Q-values and target Q-values, which leads to unstable learning caused by the correlations between such values [45]. To avoid such a learning instability, the DRL-agent of DSARA uses two NNs to learn the optimal action for every state: an evaluation NN that estimates the Q-value ($Q(s_t, a_t)$), and a fixed target NN that estimates the target Q-value ($Q^+(s_t, a_t)$) which is used as a label to calculate the loss for training the evaluation NN (see Figure 4.4). The evaluation NN is trained by using a mini-batch of experiences from the replay memory. Weights and

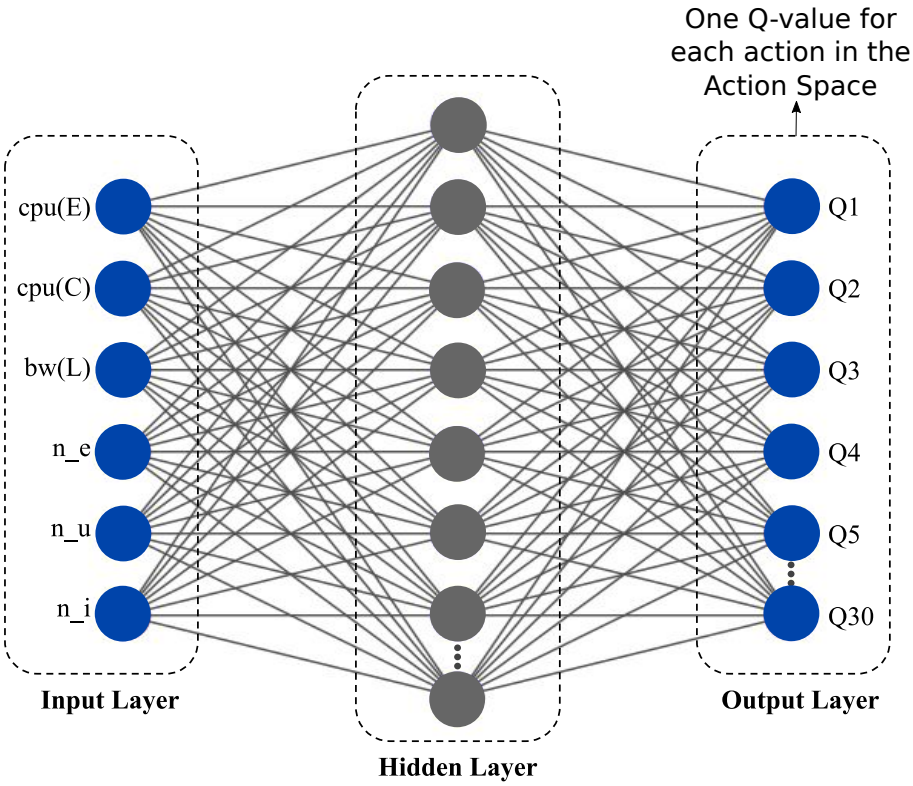


Figure 4.5 Structure of evaluation NN and target NN of DSARA

biases are adjusted by a gradient descent and backpropagation approach [52]. Target NN is fixed temporarily, for stability reasons as mentioned above, and updated only periodically with the trained parameters of evaluation NN [10].

The structure of an NN consists of an input layer, one or more hidden layers, and an output layer. Figure 4.5 depicts the structure of the NNs used by our DRL-agent. There are 6 neurons in the input layer, *i.e.,* one input neuron per variable in the tuple that represents an state (*i.e.,* $\{cpu(E), cpu(C), bw(L), n_e, n_u, n_m\}$). The output layer has 30 neurons, *i.e.,* one neuron for each of the 30 actions in the Action Space A. Each neuron in the output layer estimates the Q-value Q_i associated with action $a_i \in A$. Finally, we consider a single hidden layer since it is enough to approximate any function. Moreover, a larger amount of layers may require longer training periods [36].

4.3.3 DRL-based Admission Control Algorithm

The DRL-based procedure of ACM, described in Algorithm 3, uses experience replay, evaluation and target NNs, and gradient descent and backpropagation. The input of this algorithm is the set of NSLRs collected in a time window. The output at each step is a set of granted NSLRs that maximizes the profit for NSPs and optimizes the resource utilization of the substrate network.

Algorithm 3 starts by initializing evaluation NN, target NN, and replay memory (*line 1*). The algorithm consists of an outer loop that goes through episodes and an inner loop that goes through steps. A step is every admission decision taken over a set of NSLRs collected in a time window by the DRL-agent after receiving a new state. An episode

Algorithm 3: DRL-based Admission Control Algorithm

Data :
Discount factor (γ)
Maximum Exploration (ε)
Decay rate
Training start
Mini-batch size
Target NN update (C)
Hidden layer size
Number of learning episodes (n)
State space (S)
Action space (A)

Input : Sets of NSLR (RS) collected during time windows

Result: Admitted NSLRs that generates the maximum profit

Initialize: Evaluation NN Q with weights θ, Target NN $\hat{Q}$ with weights $\hat{\theta}$, and Replay Memory D

for *episode* $\leftarrow 1$ **to** n **do**
  The agent observes the initial state s_i;
  while *next state* s_{t+1} *is not final state* **do**
    Update exploration probability ε by using equation 4.6;
    The agent selects action a_t according to equation 4.7;
    The agent invokes the Prioritizer to sort the NSLRs into a priority queue PR;
    for *each* $nslr \in PR$ **do**
      The agent invokes RAM that runs algorithm 2 to map $nslr$;
      if $nslr$ *is mapped* **then**
        The agent admits $nslr$ and sends it to Lifecycle;
      end
      else
        The agent rejects $nslr$;
      end
    end
    The agent receives reward R_t (calculated by equation 4.5) and observes new state s_{t+1};
    Store experience $e_t = (s_t, a_t, s_{t+1}, R_t)$ into D;
    Sample random mini-batch of experiences from D;
    Estimate $Q(s_t, a_t)$ and calculate $Q^+(s_t, a_t) = R_t + \gamma \cdot maxQ^+(s_{t+1}, a)$, if state is final state, $Q^+(s_t, a_t) = R_t$;
    Minimize loss (Equation 4.2) by gradient descent and updates the weights θ of the evaluation network Q;
    The current state is updated $s_t \leftarrow s_{t+1}$;
  end
  Every C episodes, the agent clones the evaluation NN parameters to the target NN ($\hat{Q} = Q$)
end

consists of a set of steps. The outer loop (*line 2*) allows the DRL-agent to start a learning episode by observing the initial state provided by the environment (*i.e.,* the substrate network). After C episodes, the DRL-agent replaces the target NN weights and biases by making a copy of such values from the evaluation NN. Remember, target NN is fixed only temporarily to enable stable training. After C episodes, it can be updated with the trained parameters of evaluation NN.

The inner loop (*line 4*) is performed until m steps. In this loop, the DRL-agent executes the instructions described in the following. First, the DRL-agent decays its exploration probability ε according to equation 4.6. Second, the DRL-agent uses the exploration probability and the exploration method described in Equation 4.7 to select either a random action or an optimal action a_t for the current state s_t. The optimal action allows the DRL-agent to exploit the learned knowledge. Third, selected action a_t is passed to the prioritizer that will sort all the NSLRs received in a priority queue according to their arrival time and the weight value of their type of service. Fourth, each NSLR in the priority queue is sent to RAM. If resources are allocated, the NSLR is mapped onto the substrate, *i.e.* is considered admitted. Information on the acceptance of an NSLR is passed to the Lifecycle module.

Fifth, the DRL-agent receives a reward R_t for the taken action a_t, calculated by Equation 4.5, and observes a new state s_{t+1} provided by the environment. Sixth, the DRL-agent builds the experience ($e_t = (s_t, a_t, s_{t+1}, R_t)$) and stores it into the replay memory D.

Seventh, the DRL-agent randomly takes a set of experiences (mini-batch) from the replay memory aimed to train the evaluation NN. This enables our DRL-agent to learn from a reduced number of interactions with the environment compared to an RL-agent. Eighth, by taking the fields s_t, R_t and s_{t+1} from the experiences, the DRL-agent first estimates the Q-values $Q(s_t, a_t)$ by using the evaluation NN, and then, it calculates the associated target values $Q^+(s_t, a_t)$ by using the target NN (see Equation 4.1).

Ninth, the DRL-agent uses gradient descent and backpropagation to back-propagate the error for adjusting the weights and biases of the evaluation NN. This is an important step to minimize the loss function presented in Equation 4.2; a low loss means the estimations of the DRL-agent are accurate. Lines 19, 20, and 21 are only executed after the DRL-agent has stored a certain quantity of experiences. Finally, the new state becomes the current state, and the DRL-agent begins a new iteration.

4.4 Performance Evaluation

4.4.1 Metrics

We evaluated DSARA regarding the profit, resource utilization, and acceptance ratio. The profit P is calculated according to Equation 4.3, The resource utilization is given by:

$$U = \frac{\frac{\sum_j cpu(nsl_j)}{CPU(SN)} + \frac{\sum_j bw(nsl_j)}{BW(SN)}}{2} \tag{4.8}$$

where:

- $CPU(SN)$ - is the total processing capacity in SN,
- $\sum_j cpu(nsl_j)$ - is the processing resource utilized by all NSLRs instantiated in SN,
- $BW(SN)$ is the total network bandwidth,
- $\sum_j bw(nsl_j)$ is the bandwidth utilized by all NSLR instantiated.

The acceptance ratio is the ratio between the number of admitted NSLRs (req_a) and the number of NSLRs (req_t) requested.

$$AR = \frac{req_a}{req_t} \tag{4.9}$$

4.4.2 Experiment Setup

Parameter	**Value**
Substrate	16, 32, 64-node topologies
Capacity of nodes (cpu units)	cloud: 300, edge: 100
Capacity of links (bw units)	100
Mean operational time (time units)	12
Total load (requests per time unit)	$1, 3, 5, 7, 10, 15, 20, 25, 30, 40, 60, 80, 100$
Time window (time units)	2

Table 4.2 Simulation Parameters

The modules of DSARA were developed by using Python3. To test DSARA, we developed a Python-based discrete event simulator and executed the experiments on an Ubuntu 16.04 LTS desktop with Intel Core i5-4570 CPU and 15,5 GB RAM. The simulator uses the NetworkX library [31] to create and manipulate NSLRs graphs, substrate network topologies, and resources.

The evaluation of DSARA includes experiments with 16, 32, and 64-node topologies generated randomly by using the Barabasi-Alberth algorithm [5]. The 16-node topology is composed of 4 core nodes and 12 edge nodes. The 32-node topology includes 8 core nodes and 24 edge nodes. The 64-node topology has 16 core and 48 edge nodes.

Aimed to differentiate core and edge nodes, we assume 300 and 100 processing units the capacities in each core and edge nodes, respectively. We assume 100 bandwidth units in each substrate link. The computational demand of each VNF is 5 processing units. The virtual links in the eMBB, URLLC, and MIoT service graphs require 3, 2, and 1 bandwidth units, respectively. The operational time of NSLRs follows an exponential distribution with a mean of 12 time units. The arrival of each type of NSLR follows a Poisson process. The three arrivals rates are independent and identically distributed random variables. Poisson process is proper to model random events as the NSLRs arrivals [63].

We assess DSARA under different load conditions by varying the total load (*i.e.,* the sum of the three arrival rates) from 1 to 100 requests per time unit. The time window duration is 2 time units. We carried out 33 repetitions of the tests to obtain results with a 95% confidence interval. Table 4.2 summarizes the simulation parameters.

DSARA performance is compared to SARA and the algorithms NR and AAR. AAR is a simple algorithm that admits NSLRs by verifying if the available resources in the substrate meet the requests demand. NR extends AAR by ranking the substrate nodes according to its embedding potential, similar to [35] and [41], which allows to accept more NSLRs and therefore achieve higher profits. SARA learns to select the most profitable NSLRs by using an RL-based agent. Unlike DSARA and SARA, NR and AAR do not leverage any ML technique to make admission decisions.

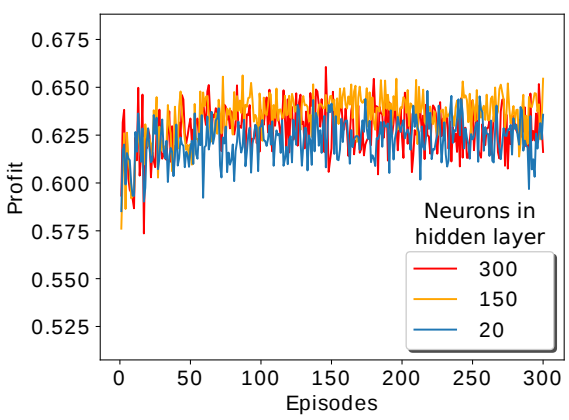
DRL-agent training and setup

Parameter	**Value**
Neurons in hidden layer	150
γ (discount factor)	0.9
Maximum Exploration ε_{max}	1
Decay rate	1/1000
Training start	300 steps
Mini-batch size	15
Target NN update	every 150 steps

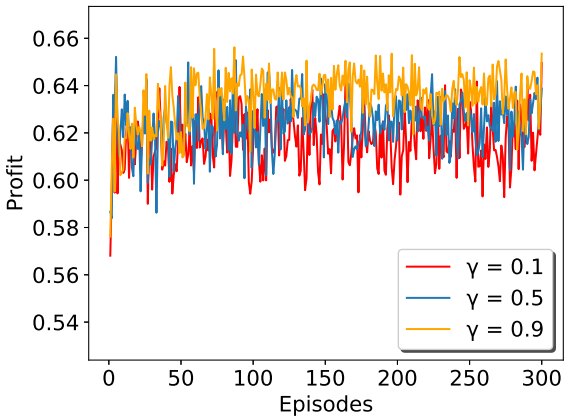
Table 4.3 DRL setup

We trained the DRL-agent in an episodic setting. Each episode consists of 15 steps of 2 time units; recall a step corresponds to the action the agent takes over a set of NSLRs collected in a time window. We conducted several preliminary tests to define the values of the learning parameters (see Figure 4.6). For the sake of clarity, we depict three results in each figure.

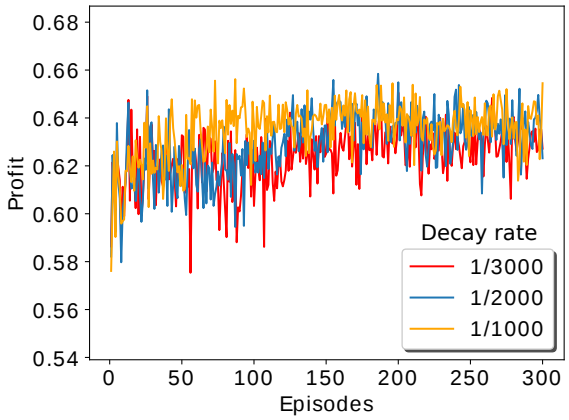
Evaluation NN and target NN have one hidden layer of 150 neurons. A single hidden layer is proven to approximate any function [36]. A larger amount requires a longer training period. A configuration with 150 neurons in the hidden layer seems to approximate more accurately the policy function than lower or higher values like 20 and 300 neurons



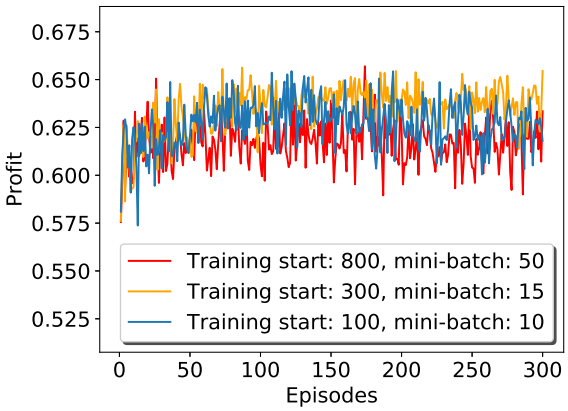
(a) Varying number of neurons in hidden layer



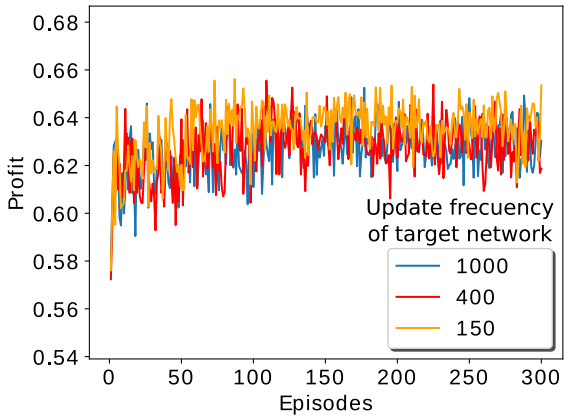
(b) Varying discount factor



(c) Varying decay rate



(d) Varying training start and mini-batch size



(e) Varying update frequency of target NN

Figure 4.6 Profit Results when varying DRL parameters

(see Figure 4.6a). Regarding γ (discount factor), DSARA obtained higher and more stable profit values when we set γ to 0.9 than to lower values (see Figure 4.6b).

We set ϵ (exploration factor) to a maximum of 1, *i.e.,* initially, the agent's exploration probability is 100%. Figure 4.6c shows that 1/1000 is the most appropriate value of decay rate for our approach. With lower decay rates, such as 1/2000 and 1/3000, DSARA takes longer to reach maximum profit values since it explores for longer periods.

Training starts at 300 steps (*i.e.,* 300 experiences stored in replay memory) and is

perform with mini-batches of 15 samples. Different configurations can lead to suboptimal solutions as depicted in 4.6d. The weights of evaluation NN are copied to target NN every 100 steps after training starts. Other frequency values result in slightly lower performance (4.6e). In accordance with the above, Table 4.3 summarizes the setup of our DRL-agent.

4.4.3 Results for a 16-node topology

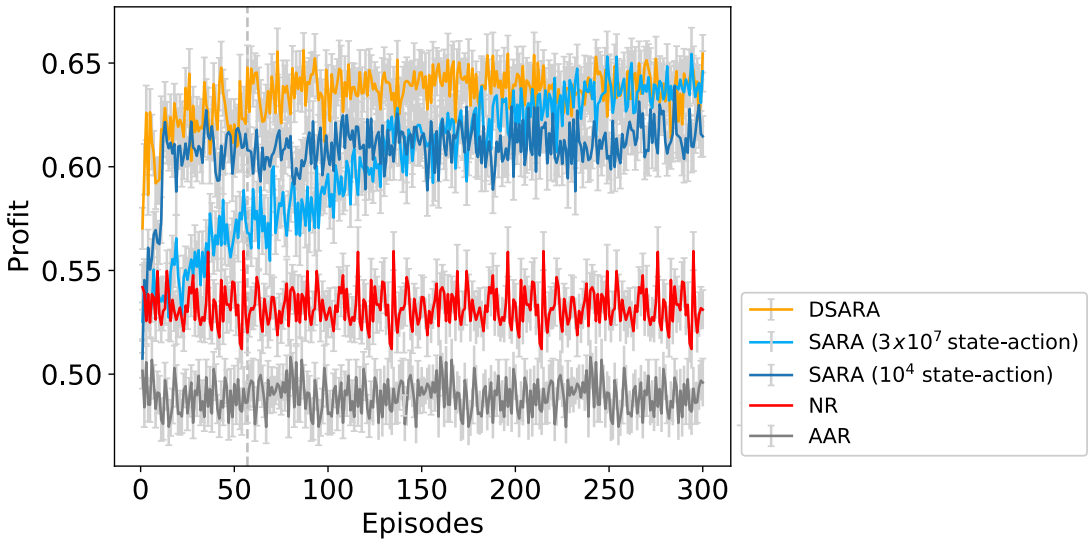


Figure 4.7 Profit on a 16-node topology

Figure 4.7 depicts the profit achieved by DSARA during 300 episodes for a load of 20 requests per time unit on a topology of 16 nodes. The dark-blue line shows the results achieved by SARA in Chapter 3 (*i.e.,* for a scenario with 10^4 state-action pairs). The light-blue line shows the profit reached by SARA in a larger scenario with 3×10^7 state-action pairs. Remember, this new scenario enables SARA to reach a maximum profit higher than it achieved in the first scenario; however, the convergence is delayed.

DSARA overcame the convergence problem of light-blue SARA while reaching the same maximum profit. This earlier convergence implies DSARA gets higher profits than those obtained by light-blue SARA during almost 200 episodes. Even though DSARA converges after does dark-blue SARA, its profit values before this convergence (episodes 1 to 55) are already higher in most of the cases. After converging, DSARA maintains a 3% higher profit than the reached by dark-blue SARA. Moreover, DSARA produced 10% higher profit than did NR and 14.3% than AAR.

We have proven that for a larger scenario, the DRL-agent of DSARA enables a much faster convergence than SARA. Hereinafter, we compare the performance of DSARA to the performance showed by SARA in Chapter 3 (*i.e.,* scenario with 10^4 state-action pairs).

In Figure 4.8, we show the results of resource utilization for a load of 20 requests per time unit on the topology with 16 nodes. As in 4.7, DSARA outperforms SARA both before and after convergence. DSARA achieved a maximum resource utilization that is 4% higher than the one obtained by SARA. The acceptance ratio results, depicted in figure 4.9, show DSARA reached slightly higher values than did SARA. These results are

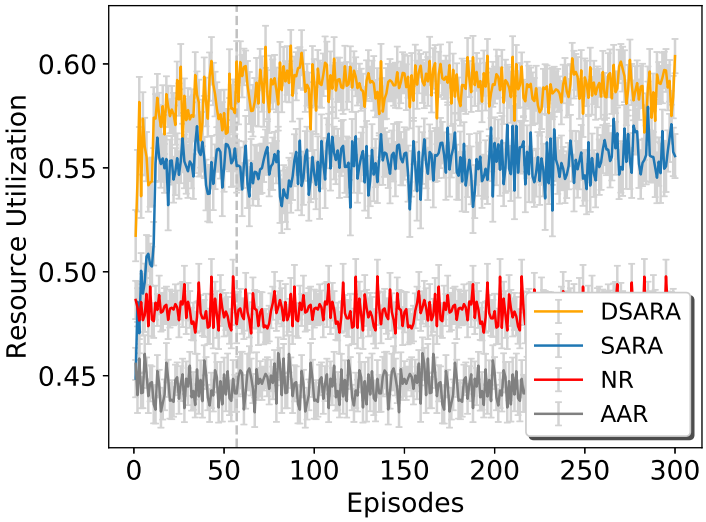


Figure 4.8 Utilization on a 16-node topology

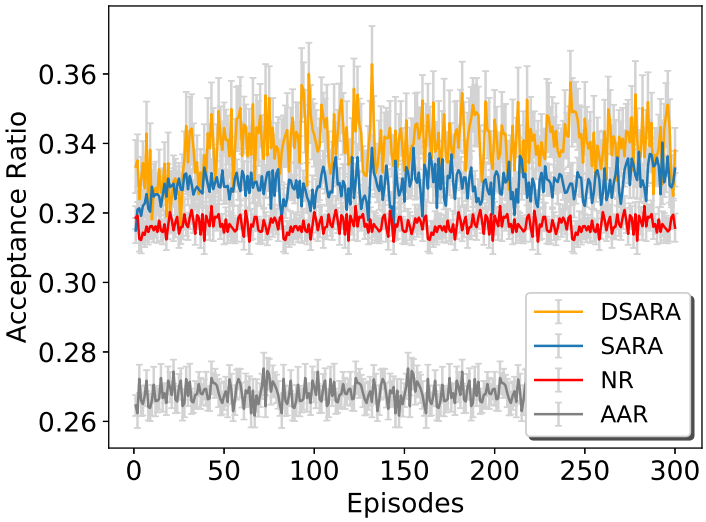


Figure 4.9 Acceptance ratio on a 16-node topology

explained since our objective is not the optimization of acceptance but profit. Moreover, acceptance results do not negatively impact profit results (see Figure 4.7)

In Figures 4.10 (profit) and 4.11 (acceptance ratio), we compare the results obtained by DSARA, SARA, NR, and AAR when varying the arrival time for tests carried out on the 16-node topology. Results for SARA and DSARA are the average values calculated after their respective convergences. From Figure 4.10, we conclude that medium and high loads favor the performance of DSARA in terms of profit. DSARA starts to overcome SARA from 10 request per time unit where the difference is 1.14% and it gets a maximum difference in 25 requests per time unit (3.2%). From this load, the difference begins to decrease slightly. This could happen due to a possible saturation of substrate resources which avoids admitting more NSLRs and hence, generating additional profit. For 100 requests per time unit, the difference is 2%.

Acceptance ratio results in Figure 4.11 show DSARA obtained values that are similar to those SARA reached. This behaviour is normal, as mentioned before, the goal of this

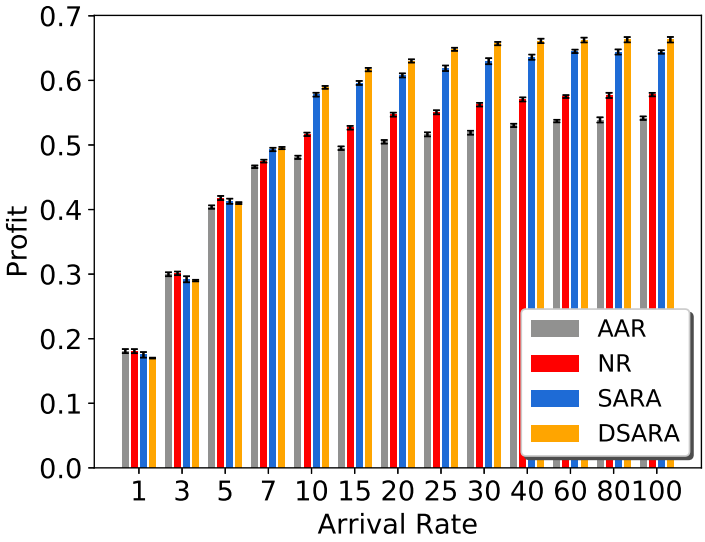


Figure 4.10 Profit vs Arrival Rate on a 16-node topology

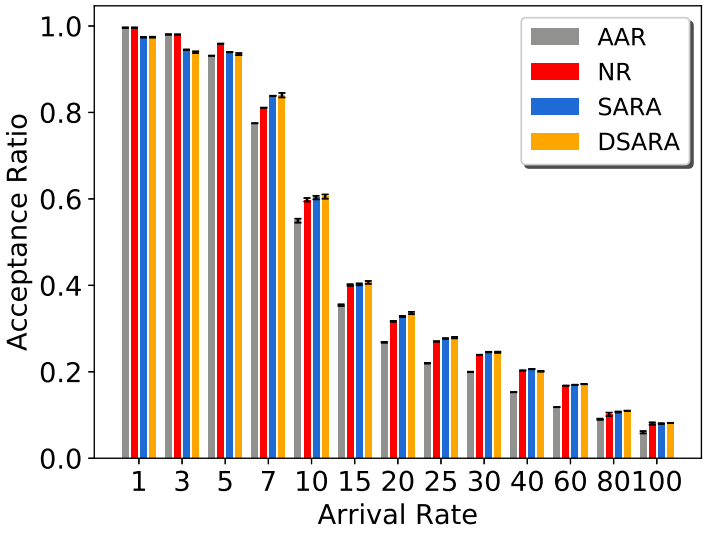


Figure 4.11 Acceptance Ratio vs Arrival Rate on a 16-node topology

work is to optimize profit and not acceptance ratio. Remember that acceptance is not the only factor that influences profit, but also the prioritization given in each state since the distinct types of NSLRs can generate different profits.

4.4.4 Results for a 32-node topology

Figure 4.12 depicts DSARA performance compared to SARA, NR, and AAR in terms of profit for a load of 20 requests per time unit and tested on the 32-node topology. Before converging at episode 54, DSARA exhibits similar profit values than the produced by SARA. After its convergence, DSARA produced 1%, 5.1%, and 8.4% higher profit than the produced by SARA, NR, and AAR, in the order given. DSARA found a more optimal action for each state.

Resource utilization results in Figure 4.13 show DSARA used 1.5% more resources than the achieved by SARA when tested on the 32-node topology and a load of 20 requests

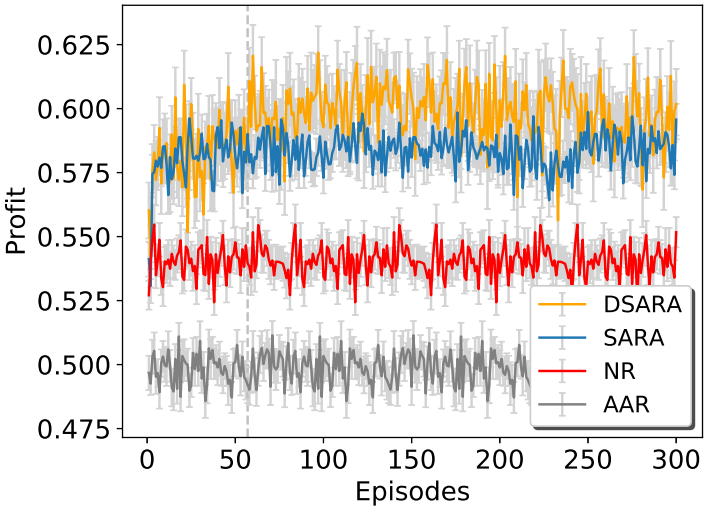


Figure 4.12 Profit on a 32-node topology

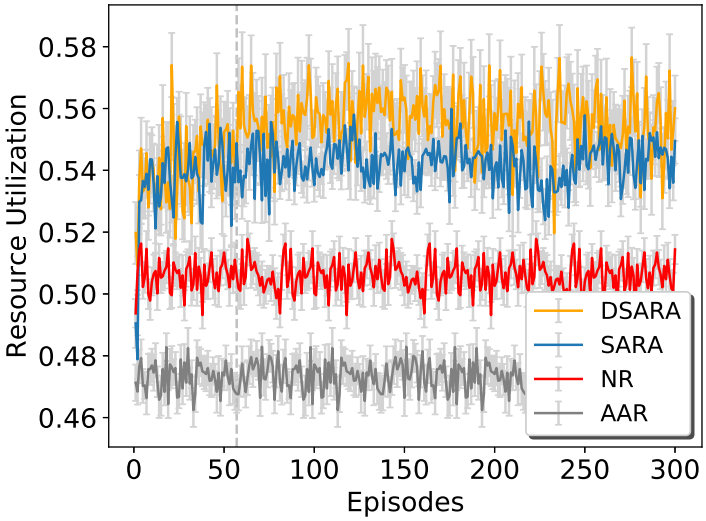


Figure 4.13 Utilization on a 32-node topology

per time unit. Moreover, DSARA reached 4.5% and 8.5% higher utilization than NR and AAR, respectively.

Figure 4.14 depicts acceptance ratio obtained by DSARA, SARA, NR, and AAR for the 32-node topology and 20 requests per time unit. DSARA obtained 1% lower acceptance ratio than the obtained by SARA. These results do not negatively impact profit results, see 4.12.

Profit and acceptance ratio results obtained by DSARA, SARA, NR, and AAR on the 32-node topology for different loads are depicted in in Figures 4.15 and 4.16. Results showed for SARA and DSARA are the average values calculated after their corresponding convergences.

Profit results, Figure 4.15, show DSARA overcomes SARA from 20 request per time unit where the difference is just 1%. However, such a difference increases as arrival rate increases; the highest difference is 3% and is obtained with a load of 30 requests per time unit. From a load of 40 requests, similar to results in the 16-node topology (Figure 4.7),

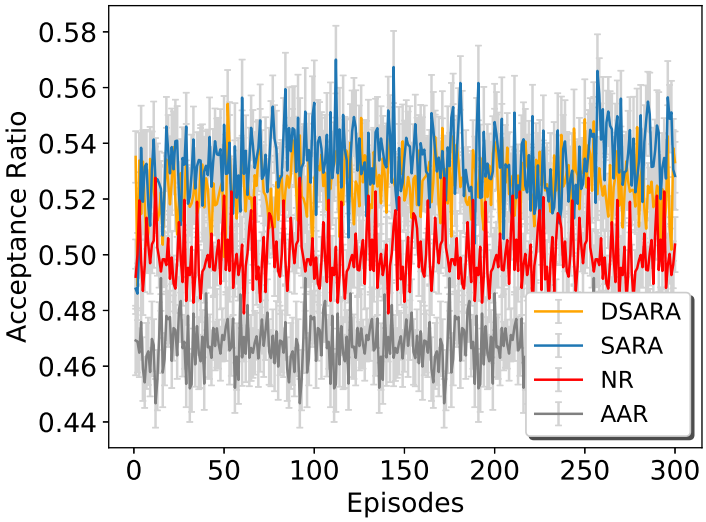


Figure 4.14 Acceptance ratio on a 32-node topology

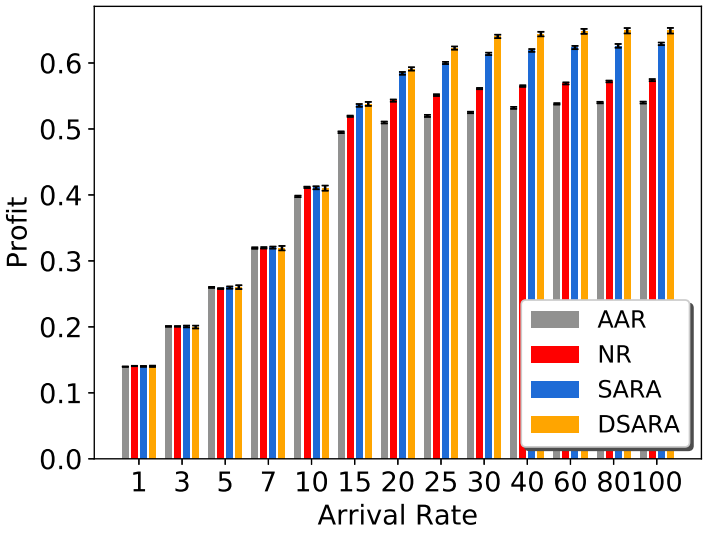


Figure 4.15 Profit vs Arrival Rate on a 32-node topology

the profit difference between DSARA and SARA begins to shorten. For 100 requests per time unit, the difference is 2%. This could be explained since substrate resources are saturated which avoids admitting more NSLRs and hence, generating additional profit. Regarding acceptance ratio results in Figure 4.16, DSARA performs similar to SARA. As explained before, the goal of this work is to optimize only the profit of NSPs.

4.4.5 Results for a 64-node topology

Up to this point, we have showed the performance of DSARA for 16-node and 32-node topologies with a load of 20 requests per time unit (Figures 4.7 and 4.12). However, since a 64-node topology has a more considerable amount of resources than 16-node or 32-node topologies, 20 requests per time unit are not enough to produce strong rewards that allow the agent to learn. In this sense, the results depicted in this subsection correspond to tests performed with a load of 40 requests per time unit.

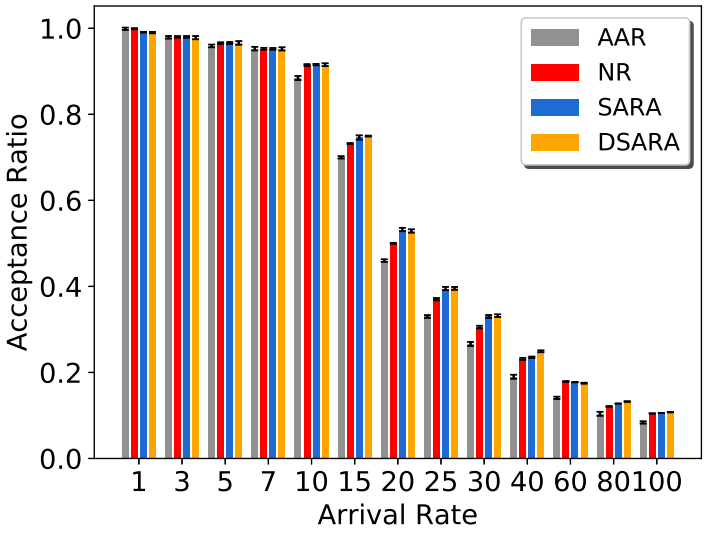


Figure 4.16 Acceptance Ratio vs Arrival Rate on a 32-node topology

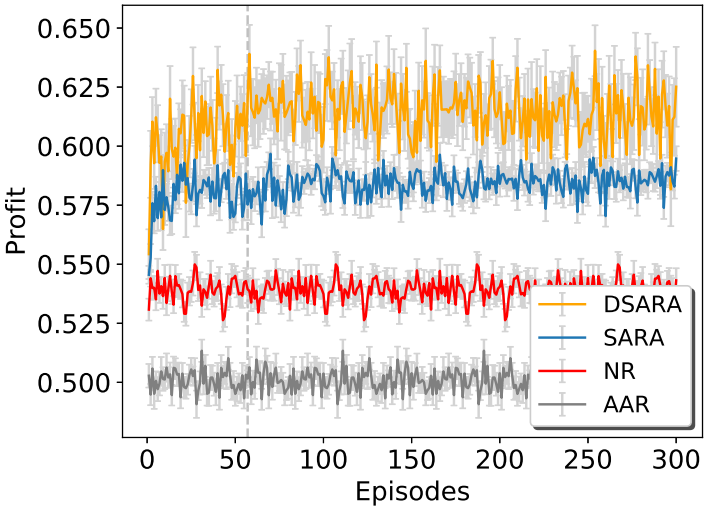


Figure 4.17 Profit on a 64-node topology

In Figure 4.17, we depict the profit obtained by DSARA, SARA, NR, and AAR. Before converging at episode 55 approximately, the profit values of DSARA are higher than the values of SARA in most of the episodes. This can be explained since our DRL-agent works with a decreasing exploration configured to start with full exploration that helps to find optimal actions faster. After converging, DSARA achieved 3% higher profit than SARA, 7.6% higher than NR and 11.4% higher than AAR.

Figure 4.18 presents the network resource utilization results. The utilization produced by DSARA grew up fast and it is higher than the achieved by SARA even before convergence (in most of the cases). After converging, DSARA achieved a utilization 2.9% higher than the achieved by SARA, 5.9% higher than NR, and 9.9% greater than AAR. In Figure 4.19, we show the acceptance ratio results. DSARA produced a slightly higher acceptance ratio than the reached by SARA. As mentioned in the evaluations for 16-node and 32-node topologies, our goal is to optimize profit.

In Figures 4.20 and 4.21, we depict profit and acceptance ratio results obtained by

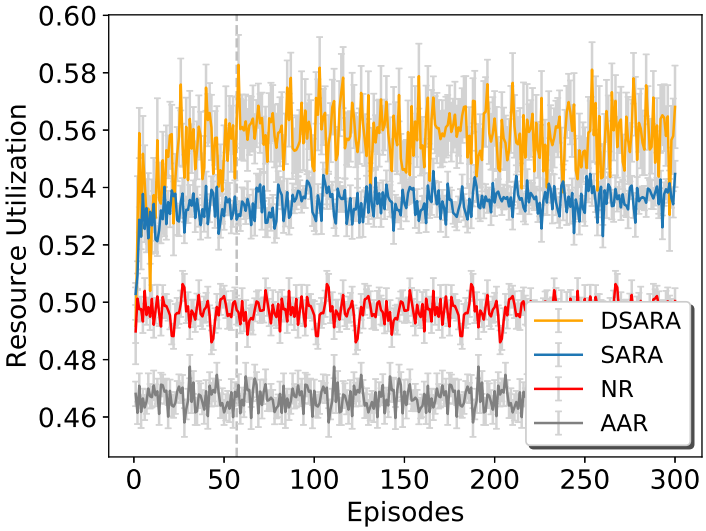


Figure 4.18 Utilization on a 64-node topology

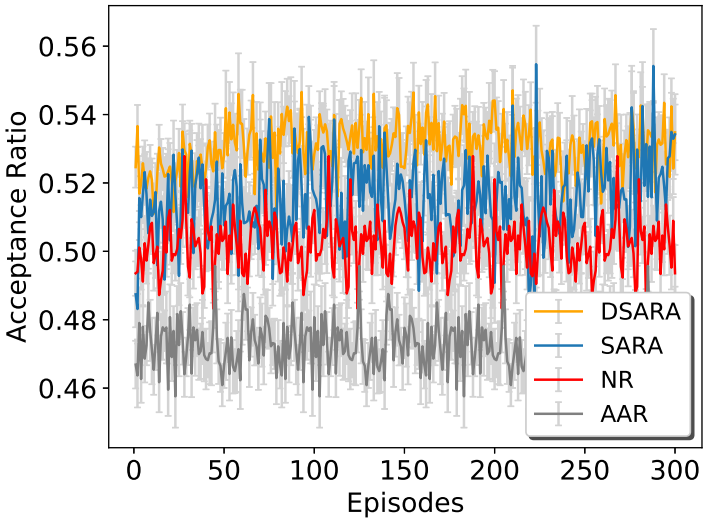


Figure 4.19 Acceptance ratio on a 64-node topology

DSARA, SARA, NR, and AAR on the 64-node topology for different loads from 1 to 100 requests per time unit. Results plotted for SARA and DSARA correspond to the average values calculated after their respective convergences.

Profit results, Figure 4.20, show DSARA's profit starts to overcome the achieved by SARA from 25 requests per time unit where the difference is only 0.5%. However, such a difference increases as arrival rate increases and it achieves the highest in 80 requests per time unit, *i.e.,* 3.6%. For 100 requests per time unit, the difference has decreased to 3.1%. This could be explained by the possible saturation of substrate resources which avoids admitting more NSLRs and hence, generating additional profit. In relation to the acceptance ratio in Figure 4.21, the results of DSARA are very close to the values reached by SARA. This is because we aim to optimize only the profit of NSPs.

Figure 4.22 depicts the profit results obtained by DSARA, SARA, NR, and AAR for different topology sizes and arrival rates. For the sake of visualization, we show results for only 7 of the total 13 arrival rates showed in previous tests.

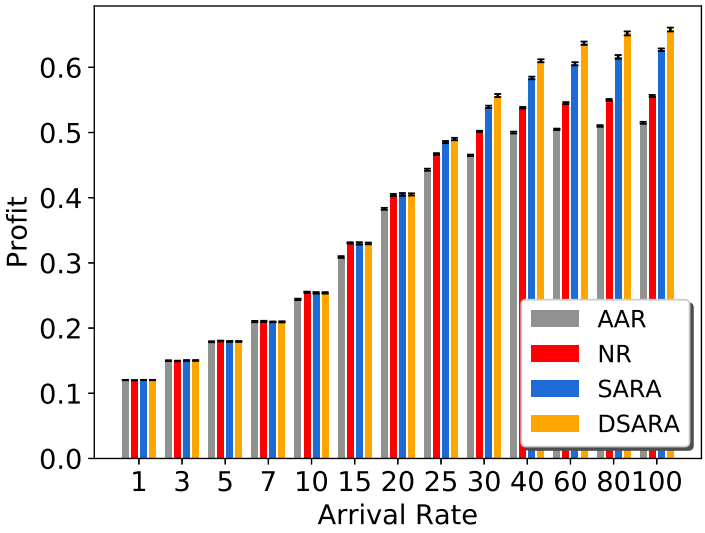


Figure 4.20 Profit vs Arrival Rate on a 64-node topology

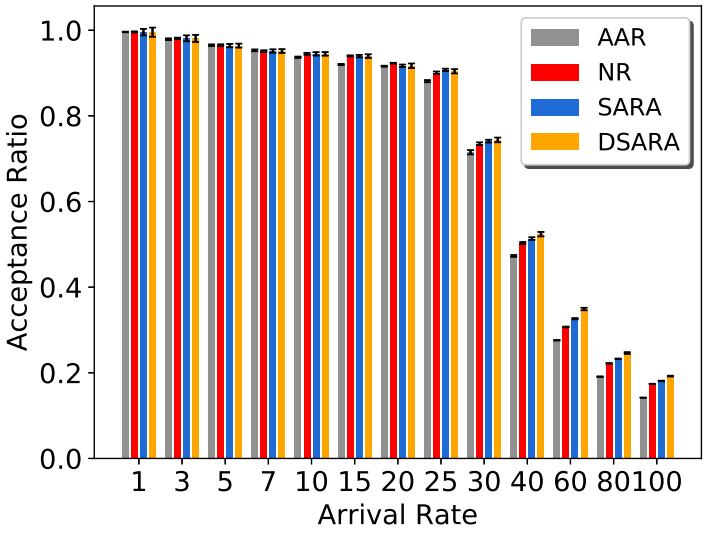


Figure 4.21 Acceptance Ratio vs Arrival Rate on a 64-node topology

For the three topologies, both SARA and DSARA performs better as the arrival rate increases. DSARA results start to be higher than the obtained by SARA from 10, 20, and 30 requests per time unit for 16, 32, and 64-node topologies, respectively. Similarly, SARA performance is better than NR from 7, 15, and 25 requests for 16, 32, and 64-node topologies, respectively. This behavior responds to the fact that DSARA and SARA learn with a reward provided by the substrate network that is basically the profit normalized according to its resource capacity. Since large substrate networks have many resources, they require a larger amount of requests to reach strong profits/rewards useful for the agents. As a result, for a small topology such as 16-node, a load of 7 requests per time unit produces rewards strong enough for the agent to learn (Figure 4.10), while for a large topology such as 64-node, a load of 25 requests per time unit was needed (Figure 4.20).

The most significant gains in profit achieved by DSARA with respect to SARA were 3.2%, for a load of 25 requests per time unit on the 16-node topology, 3% for 30 requests per time unit on the 32-node topology, and 3.6% for 80 requests on the 64-node topology.

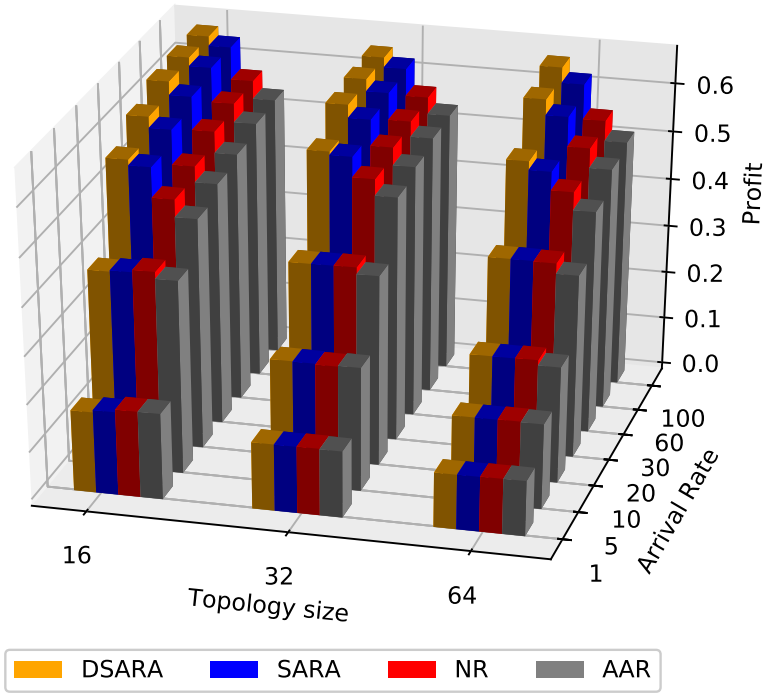


Figure 4.22 Profit results for different loads and topology sizes

These gains may not seem significant; however, in terms of money, they can make a big difference. In all cases, lower loads to those mentioned produce lower differences since they also produce lower profits. At higher loads, the differences in profit are slightly lower. This behavior could be explained by the possible saturation of substrate resources that takes place at higher loads when substrate resources are more abundant. For instance, it is observed from 30, 40, and 100 requests per time unit on the 16-node, 32-node and 64-node topologies, in the order given.

4.5 Summary

In this chapter, with the aim of increasing further the performance reached by SARA, we have proposed DSARA that jointly performs admission control and resource allocation for 5G network slice requests. DSARA uses a DRL-agent that selects the most profitable requests in a set of them that arrived within a time window. Requests of eMBB, URLLC, and MIoT service types were considered as well as edge and core nodes in the substrate network.

DSARA benefits from Deep Q-learning, and in general, from RL. First, DSARA learns to select from a set of NSLRs collected during a time window, the most profitable ones by receiving feedback from the environment (*i.e.,* the substrate network). Second, since DSARA is model-free, it does not make assumptions about the environment; DSARA learns while exploring the environment without a prior-knowledge about the substrate

network. Third, DSARA adapts to changes in the environment since it operates online, meaning that it continually learns from the interaction with the environment. Fourth, thanks to the generalization provided by its NNs, DSARA learns quickly (see Figure 4.7). This enables DSARA to cope with larger scenarios than those SARA can face. Moreover, NNs only need to store a limited number of variables (*i.e.,* the weights and biases of the network structure) to accurately estimate the policy function.

From a quantitative point of view, DSARA outperforms SARA when evaluating with different loads and on distinct topology sizes. Tests on the 16-node topology show that the maximum profit reached by DSARA is 3.2% higher than the profit achieved by SARA. For 32-node topology, DSARA achieved up to 3% higher profit values than those SARA achieved. Results on the 64-node topology show DSARA reached up to 3.6% higher profit than did SARA.

Evaluation results evidence DSARA as a suitable DRL-based approach to overcome the performance reached by SARA for managing the admission of 5G NSLRs and their resource allocation.

Chapter 5

Conclusion and Future Work

NSPs will receive myriads of network slice requests generated by multiple tenants. Considering that substrate resources are finite and 5G use cases have particular QoS requirements as well as different deployment costs, NSPs face the challenge of controlling the admission of such requests and managing their resource allocation. In this thesis, we explored the use of machine learning techniques, such as Reinforcement Learning and Deep Learning, to cope with the admission of 5G NSLRs and their resource allocation while improving the profit of NSPs and network resource utilization.

In Chapter 3, we introduced SARA, an approach that jointly performs admission control and resource allocation for NSLRs of eMBB, URLLC, and MIoT service types in 5G. The RL-based algorithm of SARA allows to prioritize the most profitable NSLRs from a set of them collected in a time window. SARA employs Q-learning, a model-free RL algorithm, meaning that SARA does not make assumptions about the substrate network as do optimization-based approaches. SARA learns while exploring the environment without a prior-knowledge about the substrate. SARA is adaptable since it operates online, which implies it continually learns from the interaction with the environment. The resource allocation algorithm of SARA permits to map NSLRs onto the substrate network by considering their 5G service type and differentiating core nodes from edge nodes in order to accomplish QoS requirements.

SARA outperforms the benchmark. For the 16-node topology, SARA achieved up to 7% and 11.3% higher profit than the profit given by the NR and AAR algorithms, respectively. Results on the 32-node topology show that SARA reached up to 5.6% and 9% higher profit than the profit given by the NR and AAR algorithms, respectively. For the 64-node topology, SARA obtained up to 7.1% and 11.2% higher profit than the profit generated by the NR and AAR, respectively. These results corroborate that SARA is a suitable solution for managing the admission of NSLRs and their resource allocation aimed at optimizing the profit of NSPs.

In Chapter 4, we introduced DSARA, an implementation of SARA based on DRL for further improving the performance obtained by SARA. DSARA showed to converge faster than SARA in a scenario with large state space and action space. The use of a DRL-agent enables DSARA to face large scenarios since it learns from a reduced number of interactions with the environment; our DRL-agent generalizes knowledge from past experiences and applies it to similar states. Such a generalization is achieved thanks to

the use of NNs that allow to accurately estimate the policy function by storing a limited number of variables compared to a Q-table.

DSARA overcomes SARA performance on tests with different loads and topology sizes.In particular, DSARA achieved up to 3.2%, 3%, and 3.6% higher profit values than SARA did for 16, 32, and 64-node topologies, respectively. Results of this chapter evidence the DRL-agent of DSARA is a suitable strategy to cope with large scenarios where the use of typical RL-based approaches as SARA may become impractical.

As future work, we will enrich the admission control mechanism with the latest enhancements to RL and DRL. The RA mechanism can be polished with more sophisticated strategies like the coordinated node and link mapping as proposed in [56] [19]. Furthermore, we will extend our solutions to perform admission control and resource allocation for end-to-end slices, *i.e.,* involving both 5G radio access and core network. Also, we plan to provide an adaptive scaling mechanism for instantiated network slices aimed to support their QoS variations at run time.

Bibliography

[1] 3GPP. 5G;System architecture for the 5G System (5GS). Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 03 2020. Version 15.9.0.

[2] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys Tutorials*, 20(3):2429–2453, thirdquarter 2018.

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

[4] M. Agiwal, A. Roy, and N. Saxena. Next generation 5g wireless networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 18(3):1617–1655, thirdquarter 2016.

[5] Réka Albert and Albert-László Barabási. Topology of evolving networks: local events and universality. *Physical review letters*, 85(24):5234, 2000.

[6] Gustavo Prado Alkmim, Daniel Macêdo Batista, and Nelson L. S. da Fonseca. Mapping virtual networks onto substrate networks. *J. Internet Serv. Appl.*, 4(1):3:1–3:15, 2013.

[7] NGMN Alliance. 5g white paper. *Next generation mobile networks, white paper*, pages 1–125, 2015.

[8] NGMN Alliance. Description of network slicing concept. *NGMN 5G P*, 1, 2016.

[9] Esther M Arkin, Samir Khuller, and Joseph SB Mitchell. Geometric knapsack problems. *Algorithmica*, 10(5):399–427, 1993.

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[11] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, and X. Costa-Pérez. A machine learning approach to 5g infrastructure market optimization. *IEEE Transactions on Mobile Computing*, 19(3):498–512, 2020.

[12] Dario Bega, Marco Gramaglia, Albert Banchs, Vincenzo Sciancalepore, Konstantinos Samdanis, and Xavier Costa-Perez. Optimising 5g infrastructure markets: The business of network slicing. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9, Atlanta, GA, USA, 2017. IEEE.

[13] Stefano Boccaletti, Vito Latora, Yamir Moreno, Martin Chavez, and D-U Hwang. Complex networks: Structure and dynamics. *Physics reports*, 424(4-5):175–308, 2006.

[14] Juliana Freitag Borin and Nelson Luis Saldanha da Fonseca. Admission control for wimax networks. *Wireless Communications and Mobile Computing*, 14(14):1409–1419, 2014.

[15] Raouf Boutaba, Mohammad Ali Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada Solano, and Oscar Mauricio Caicedo Rendon. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9:1–99, 2018.

[16] GABRIEL Brown. Service-based architecture for 5g core networks. *A Heavy Reading white paper produced for Huawei Technologies Co. Ltd. Online verfügbar unter: https://www. huawei. com/en/press-events/news/2017/11/HeavyReading-WhitePaper-5G-Core-Network, letzter Zugriff am*, 1:2018, 2017.

[17] H. I. Bulbul and Ö. Unsal. Comparison of classification techniques used in machine learning as applied on vocational guidance data. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, volume 2, pages 298–301, Dec 2011.

[18] B. Chatras, U. S. Tsang Kwong, and N. Bihannic. Nfv enabling network slicing for 5g. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 219–225, Paris, France, March 2017.

[19] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Trans. on networking*, 20(1):206–219, 2011.

[20] Rachid El Hattachi and Javan Erfanian. Ngmn 5g white paper. *NGMN Alliance, February*, 2015.

[21] N Etsi. Etsi gs nfv 002 v1. 1.1 network functions virtualization (nfv). *Architectural Framework. sl: ETSI*, 2013.

[22] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann. Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–6, Kyoto, Japan, June 2011.

[23] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann. Adaptive-vne: A flexible resource allocation for virtual network embedding algorithm. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 2640–2646, Anaheim, CA, USA, Dec 2012.

[24] Nelson L. S. Da Fonseca and Roberto De A. Façanha. The look-ahead-maximize-batch batching policy. *IEEE Trans. Multimedia*, 4(1):114–120, 2002.

[25] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. *An Introduction to Deep Reinforcement Learning*. 2018.

[26] Fabio Giust, Gianluca Verin, Kiril Antevski, Joey Chou, Yonggang Fang, Walter Featherstone, Francisco Fontes, Danny Frydman, Alice Li, Antonio Manzalini, et al. Mec deployments in 4g and evolution towards 5g. *ETSI White Paper*, 24:1–24, 2018.

[27] Wanqing Guan, Xiangming Wen, Luhan Wang, Zhaoming Lu, and Yidi Shen. A service-oriented deployment policy of end-to-end network slicing based on complex network theory. *IEEE Access*, 6:19691–19701, 2018.

[28] Michael Günther, Johann Schuster, and Markus Siegle. Symbolic calculation of k-shortest paths and related measures with the stochastic process algebra tool caspa. In *Proceedings of the First Workshop on Dynamic Aspects in Dependability Models for Fault-Tolerant Systems*, pages 13–18, New York, NY, USA, 2010.

[29] B. Görkemli, S. Tatlıcıoğlu, A. M. Tekalp, S. Civanlar, and E. Lokman. Dynamic control plane for sdn at scale. *IEEE Journal on Selected Areas in Communications*, 36(12):2688–2701, 2018.

[30] M. A. Habibi, M. Nasimi, B. Han, and H. D. Schotten. A comprehensive survey of ran architectures toward 5g mobile communication system. *IEEE Access*, 7:70371–70421, 2019.

[31] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploriponerng network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[32] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, Feb 2015.

[33] B. Han, V. Sciancalepore, X. Costa-Pérez, D. Feng, and H. D. Schotten. Multiservice-based network slicing orchestration with impatient tenants. *IEEE Transactions on Wireless Communications*, pages 1–1, 2020.

[34] Juliver Gil Herrera. *Resource Allocation in an NFV/SDN-based Network Architecture*. PhD thesis, Universidad de Antioquia, august 2018.

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

[36] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[37] Menglan Jiang, Massimo Condoluci, and Toktam Mahmoodi. Network slicing management & prioritization in 5g mobile systems. In *European Wireless 2016; 22th European Wireless Conference*, pages 1–6, Oulu, Finland, 2016. VDE.

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

[39] R. A. Kulkarni. Scrutinizing action performed by user on mobile app through network using machine learning techniques: A survey. In *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pages 860–863, Jan 2018.

[40] Rongpeng Li, Zhifeng Zhao, Qi Sun, I Chih-Lin, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[41] Xin Li, Chengcheng Guo, Jun Xu, Lav Gupta, and Raj Jain. Towards efficiently provisioning 5g core network slice based on resource and topology attributes. *Applied Sciences*, 9(20):4361, 2019.

[42] M. Mamdouh, M. A. I. Elrukhsi, and A. Khattab. Securing the internet of things and wireless sensor networks via machine learning: A survey. In *2018 International Conference on Computer and Applications (ICCA)*, pages 215–218, Aug 2018.

[43] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, Atlanta, GA, USA, 2016. ACM.

[44] Francisco S Melo and M Isabel Ribeiro. Q-learning with linear function approximation. In *International Conference on Computational Learning Theory*, pages 308–322. Springer, 2007.

[45] 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.

[46] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, 55(5):80–87, May 2017.

[47] M. Peng, S. Yan, K. Zhang, and C. Wang. Fog-computing-based radio access networks: issues and challenges. *IEEE Network*, 30(4):46–53, 2016.

[48] Ioannis Priggouris, Evangelos Zervas, and Stathes Hadjiefthymiades. Location-based network resource management. In *Handbook of Research on Mobile Multimedia*, pages 139–164. IGI Global, 2006.

[49] Muhammad Rehan Raza, Carlos Natalino, Peter Öhlen, Lena Wosinska, and Paolo Monti. Reinforcement learning for slicing in a 5g flexible ran. *Journal of Lightwave Technology*, 2019.

[50] Muhammad Rehan Raza, Ahmad Rostami, Lena Wosinska, and Paolo Monti. A slice admission policy based on big data analytics for multi-tenant 5g networks. *Journal of Lightwave Technology*, 37(7):1690–1697, 2019.

[51] Esteban Rodríguez, Gustavo Prado Alkmim, Nelson L. S. da Fonseca, and Daniel Macêdo Batista. Energy-aware mapping and live migration of virtual networks. *IEEE Systems Journal*, 11(2):637–648, 2017.

[52] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[53] Maria P. Salamanca, Néstor M. Peña, and Nelson L. S. da Fonseca. Impact of the routing protocol choice on the envelope-based admission control scheme for ad hoc networks. *Ad Hoc Networks*, 31:20–33, 2015.

[54] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs. Mobile traffic forecasting for maximizing 5g network slicing resource utilization. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, Atlanta, GA, USA, May 2017.

[55] Vincenzo Sciancalepore, Xavier Costa-Perez, and Albert Banchs. Rl-nsb: Reinforcement learning-based 5g network slice broker. *IEEE/ACM Transactions on Networking*, 27(4):1543–1557, 2019.

[56] Bart Spinnewyn, Pedro Heleno Isolani, Carlos Donato, Juan Felipe Botero, and Steven Latré. Coordinated service composition and embedding of 5g location-constrained network functions. *IEEE TNSM*, 15(4):1488–1502, 2018.

[57] R. Su, D. Zhang, R. Venkatesan, Z. Gong, C. Li, F. Ding, F. Jiang, and Z. Zhu. Resource allocation for network slicing in 5g telecommunication networks: A survey of principles and models. *IEEE Network*, 33(6):172–179, Nov 2019.

[58] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[59] Arryon D Tijsma, Madalina M Drugan, and Marco A Wiering. Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Athens, Greece, 2016. IEEE.

[60] Arryon D Tijsma, Madalina M Drugan, and Marco A Wiering. Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

[61] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta. Joint optimization of service function chaining and resource allocation in network function virtualization. *IEEE Access*, 4:8084–8094, 2016.

[62] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, March 2018.

[63] Ronald W Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2):223–231, 1982.

[64] Q. Zhang, F. Liu, and C. Zeng. Adaptive interference-aware vnf placement for service-customized 5g network slices. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2449–2457, 2019.