# Admission Control for 5G Core Network Slicing based on Deep Reinforcement Learning

William F. Villota-Jacome*, Oscar Mauricio Caicedo Rendon†, and Nelson L. S. da Fonseca*
*Institute of Computing, University of Campinas, Brazil
†Telematics Engineering Group, Universidad del Cauca, Colombia

***Abstract*—Network Slicing is a promising technology for providing customized logical and virtualized networks for the 5G use cases (enhanced Mobile Broadband, Ultra-Reliable Low Latency Communications, and massive Machine Type Communications), which pose distinct Quality of Service requirements. Admission Control and Resource Allocation mechanisms are pivotal for realizing network slicing efficiently, but existing mechanisms focus on slicing the radio access network. This paper proposes an approach encompassing intelligent and efficient mechanisms for Admission Control and Resource Allocation for network slicing in the 5G core network. The Admission Control mechanism introduces two solutions, one based on Reinforcement Learning (called SARA) and the other based on Deep Reinforcement Learning (called DSARA). SARA and DSARA consider the QoS requirements of 5G use cases, differentiate network core nodes from edge nodes, and process slice requests in time windows to favor the service provider's profit and resource utilization. Results show that SARA and DSARA overcome existing mechanisms for managing admission control and resource allocation in 5G core network slicing.**

***Index Terms*—5G, Network Slicing, Admission Control, Resource Allocation, Reinforcement Learning, Deep Reinforcement Learning**

## I. Introduction

5G networks support a myriad of services accessible on-demand for numerous customers and devices. The 3rd Generation Partnership Project defined the 5G use cases called Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communication (URLLC), and massive Machine Type Communications (mMTC), with different Quality of Service (QoS) requirements [1]. Network Slicing is crucial to realize the 5G use cases since it provides flexibility, modularity, and programmability for managing customized and isolated logical networks, known as network slices (NSL) [2]. The employment of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) technologies allows the customization and compliance of NSLs with Service Level Agreements (SLAs).

Network Slice Providers (NSPs) receive NSL Requests (NSLRs) to implement distinct types of NSLs intended to support the 5G use cases. NSPs need to perform intelligent and efficient Admission Control (AC) and Resource Allocation (RA) for NSLs to increase the providers' profit and improve the utilization of 5G core network resources. Several papers have addressed AC by employing different frameworks such as Queuing Theory [3], Big Data [4], Heuristics [5]–[8], Reinforcement Learning (RL) [2] [9], and Deep RL (DRL) [10]. These papers propose making admission decisions considering individual NSLRs, although this can lead to sub-optimal decisions since more profitable requests arriving in the short term may be rejected due to the unavailability of recently allocated resources. Moreover, most of these proposals neither consider the QoS requirements of different 5G use cases nor the allocation of resources in the 5G core network. Various other solutions based on heuristics [11] [12], Queuing Theory [13], Complex Network Theory [14], Linear Programming [15], Alternating Direction Method of Multipliers [16], Artificial Neural Networks (NN) [17], and DRL [18]–[21] have considered only RA and neglected AC, although this prevents the achievement of NSP goals. Jointly performing AC and RA in an intelligent way can considerably improve the achievement of NSP's objectives.

This paper proposes a novel approach that increases service providers' profit and network resource utilization by furnishing intelligent and efficient AC and RA in 5G network slicing. The approach introduces two solutions: network **S**lice requests **A**dmission and **R**esource **A**llocation (SARA), based on RL, and Deep SARA (DSARA), based on DRL. Unlike other approaches, SARA and DSARA consider the QoS requirements of 5G use cases and the 5G core network (differentiate network core nodes from edge nodes) for making acceptance and allocation decisions. The proposed solutions employ Q-learning and Deep Q-Learning (DQN improved with two NNs, replay memory, and mini-batch) to explore, exploit, and learn to prioritize NSLRs arriving in a given time window to profit maximization. Furthermore, RL and DRL allow SARA and DSARA to consider the history of slice acceptance decisions rather than just accounting for the most recent values of descriptive parameters to favor resource utilization. The proposed approach was extensively validated using simulation with various network topologies and diverse loads of NSLRs. The results show that SARA and DSARA outperform the algorithms Always Admit Requests (AAR) and Node Ranking (NR) concerning profit and resource utilization.

The contributions of this paper are:

- An architecture that uses RL and DRL for achieving

This work was supported by CAPES, CNPq, and the Sao Paulo Research Foundation (FAPESP) under grants #19/03268-0 and 2015/24494-8, as well as the Universidad del Cauca. W. Villota-Jacome, and N. da Fonseca are with the Institute of Computing, University of Campinas, Brazil. e-mail: wfernando@lrc.ic.unicamp.br, and nfonseca@ic.unicamp.br. O. Caicedo is with the Department of Telematics, Universidad del Cauca, Popayán, Colombia. e-mail: omcaicedo@unicauca.edu.co.

TABLE I
Acronyms

| Acronym | Term | Acronym | Term | Acronym | Term |
|---|---|---|---|---|---|
| AAR | Always Admit Requests | NFV | Network Function Virtualization | RL | Reinforcement Learning |
| AC | Admission Control | NN | Artificial Neural Networks | SARA | **S**lice requests **A**dmission and **R**esource **A**llocation |
| ACM | Admission Control Module | NR | Node Ranking | SDN | Software-Defined Networking |
| AMF | Access and Mobility Function | NRF | Network Repository Function | SMF | Session Management Function |
| CP | Control Plane | NSL | Network Slice | UP | User Plane |
| DQN | Deep Q-learning | NSLR | Network Slice Request | UPF | User Plane Function |
| DRL | Deep Reinforcement Learning | NSP | Network Slice Provider | URLLC | Ultra Reliable Low Latency Communications |
| DSARA | Deep SARA | RA | Resource Allocation | VNF | Virtual Network Function |
| eMBB | Enhanced Mobile Broad Band | RAM | Resource Allocation Module | | |
| mMTC | massive Machine Type Communications | RAN | Radio Access Network | | |

efficient AC and RA for NSL in 5G core networks.
- RL-based and DRL-based algorithms that maximize profit and resource utilization by making optimal acceptance and allocation decisions by considering the QoS requirements of 5G use cases, differentiating network core nodes from edge nodes, and processing NSLRs in time windows. The DRL-based algorithm overcomes the computational barriers of the RL-based for efficient obtaining of solutions for large slicing environments.

The remainder of this paper is organized as follows. Section II presents related work. Section III introduces the architecture of the proposed approach. Sections IV and V describe RL-based AC and DRL-based AC, respectively. Section VI describes the proposed mechanism for RA. Section VII presents the evaluation of SARA and DSARA. Section VIII concludes the paper and makes suggestions for future work. For the sake of readability, Table I presents the acronyms used in this paper.

## II. Related Work

This section reviews related work on AC and RA in 5G.

### A. Admission Control in 5G Network Slicing

Several investigations have addressed the AC problem in 5G. Han et al. [3] propose a utility-driven and multi-service-based solution for network slicing based on Queuing Theory to maximize network utility. This solution considers different queues for two types of requests and accounts for impatient customers. Raza et al. [4] 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. [5] introduce a heuristic-based AC mechanism to maximize user Quality of Experience. This mechanism determines the acceptance of slice requests based on the minimum and maximum data rates and available resources in the Radio Access Network (RAN). Also, it dynamically changes the allocation of radio resources to NSLs according to the traffic load. Salvat et al. [7] introduce an end-to-end slicing heuristic solution that performs AC and RA of radio, edge, and cloud resources for increasing the provider profit; the solution does not learn to improve the profit nor consider the 5G use cases.

Sciancalepore et al. [6] introduce a 5G slice broker for radio resources, which includes three modules: forecasting, AC, and scheduling. The forecasting module predicts the network traffic to dimension NSLs. The AC module selects the NSLs by employing a heuristic algorithm based on a knapsack problem. 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 it is not used to make admission decisions on slice acceptance. Challa et al. [8] propose an AC model based on a knapsack problem formulation to optimize resource monetization. Bega et al. [9] introduce an analytical model based on Semi-Markov Decision Process and a Q-learning-based algorithm to perform AC for individual RAN slicing requests. Raza et al. [2] propose an AC mechanism that employs a Q-learning agent for maximizing provider profit; this solution considers a 5G flexible RAN and prioritizes requests with different latency requirements and expected revenues. Bega et al. [10] propose a DRL-based algorithm that performs AC for individual RAN slicing requests aimed at maximizing the monetization of the infrastructure provider.

The mechanisms in the aforecited papers [2]–[10] make admission decisions for individual requests as they arrive, preventing the selection of a set of NSLRs that can potentially optimize a given NSP objective in a specific time window. Furthermore, these papers do not differentiate requests according to standardized 5G use cases, neglecting the diversity of QoS requirements of 5G services. Moreover, most of the proposed mechanisms focus on allocating radio resources [23] and disregard the allocation of 5G core network nodes (Table I). In contrast, SARA and DSARA focus on core and edge nodes and consider the typical standardized 5G use cases to perform AC and RA jointly.

### B. Resource Allocation in 5G Network Slicing

Several investigations have addressed the RA problem in 5G. Zhang et al. [11] introduce a heuristic for placing Virtual Network Functions (VNFs) in the core network for optimizing the acceptance ratio, the execution time, and throughput. When throughput degradation occurs, the heuristic changes the placement of the VNFs. Li et al. [12] 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 global resource capacities and the topological attributes of the physical nodes; the second stage performs link provisioning by using a k-shortest path algorithm. Agarwal et al. [13] propose MaxZ, a solution for

TABLE II
Related Work

| Paper | Technique | Focus |  | Time Window | Fifth Generation |  |  | Performance Metrics |
|---|---|---|---|---|---|---|---|---|
|  |  | AC | RA |  | Use Cases | Edge nodes | Core nodes |  |
| [2] | RL (Policy Network) | ✓ |  |  |  | ✓ |  | Provider profit |
| [3] | Queuing theory | ✓ |  |  |  | ✓ |  | Utility rate, admission rate, request waiting time |
| [4] | Big Data Analytics | ✓ |  |  |  | ✓ |  | Provider profit |
| [5] | Heuristic algorithm | ✓ | ✓ |  |  | ✓ |  | QoE, resource utilization |
| [6] | Heuristic algorithm | ✓ |  |  |  | ✓ |  | System resource utilization |
| [7] | Heuristic algorithm | ✓ | ✓ |  |  | ✓ | ✓ | Provider revenue |
| [8] | Knapsack problem | ✓ |  |  |  |  | ✓ | Monetization ratio |
| [9] | RL (Q-Learning) | ✓ |  |  |  | ✓ |  | Provider revenue |
| [10] | DRL (DQN) | ✓ |  |  |  | ✓ |  | Provider revenue |
| [11] | Heuristic algorithm |  | ✓ |  | ✓ |  | ✓ | Acceptance ratio and Execution time |
| [12] | Heuristic algorithm |  | ✓ |  |  |  | ✓ | Acceptance ratio, provider revenue |
| [13] | Queuing Theory |  | ✓ |  |  |  | ✓ | Running time |
| [14] | Complex Network Theory |  | ✓ |  | ✓ | ✓ | ✓ | Resource efficiency, acceptance ratio, execution time |
| [15] | Integer Programming |  | ✓ |  |  | ✓ |  | CPU Utilization |
| [16] | Alternating Direction Method of Multipliers |  | ✓ |  |  | ✓ |  | Latency |
| [17] | NNs |  | ✓ |  | ✓ | ✓ | ✓ | Latency |
| [18] | DRL (DQN) |  | ✓ |  |  | ✓ |  | Spectrum efficiency and QoE |
| [19] | DRL (DQN) |  | ✓ |  |  | ✓ | ✓ | Resource Utilization |
| [20] | DRL (DQN) |  | ✓ |  |  | ✓ |  | Resource Utilization and Satisfaction ratio |
| [21] | DRL (DDPG) |  | ✓ |  |  | ✓ | ✓ | Provider revenue |
| [22] | DRL (DQN) |  | ✓ |  |  | ✓ |  | Resource Utilization and Slice Satisfaction |
| SARA/DSARA | Q-learning and DQN | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Provider profit, resource utilization, acceptance ratio |

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).

Guan et al. [14] present a mapping algorithm for 5G NSL based on Complex Network Theory to analyze the topological characteristics of the network. The mapping process includes 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 a k-shortest path algorithm for the VNFs placed on the nodes. D'Oro et al. [15] propose a slicing framework to instantiate heterogeneous services by using Integer Linear Programming, aiming at optimizing CPU utilization at the network edge. Huang et al. [16] present an RA algorithm based on the Alternating Direction Method of Multipliers to reduce the latency experienced by User Equipment in networks composed of Base Stations and Fog nodes.

De Cola et al. [17] carry out RA for eMBB slices in 5G-satellite networks by using NNs to optimize QoS. Li et al. [18] propose an algorithm for NSL in 5G RAN to optimize the spectrum efficiency and QoE. The algorithm uses DRL to learn to allocate bandwidth to users by considering fluctuations in demand for services. Liu et al. [19] introduce a resource orchestration system that uses DRL to orchestrate networking and computing resources, aiming at optimizing utilization; this approach does not consider the 5G use cases. Sun et al. [20] propose an RA algorithm for vehicular networks based on DRL to provide QoS efficiently in which the DRL agent adjusts the allocated resources for a slice as a function of the demand. Zhang et al. [21] perform slicing in data center networks by using a DRL agent, which maximizes the revenue from provisioned slices while avoiding overutilized resources. Sun et al. [22] propose an RA framework that employs a DRL agent to adjust the resource allocated to a slice based on the average utility and resource utilization.

The aforecited papers [11]–[22] focus on mapping NSLs. They map the arriving NSLRs but without controlling admission. Performing AC and RA jointly in 5G core network slicing is critical for optimizing resource utilization and maximizing NSP profit. RL and DRL are efficient tools for solving decision-making problems modeled as Markov Decision Processes. Moreover, the Network Slicing process involves repetitive decisions and produces a large quantity of data useful to train RL and DRL algorithms [18], [24]. SARA and DSARA are based on model-free RL and DRL, respectively, meaning that they do not make assumptions about the environment (*i.e.,* substrate network) but rather learn while exploring it without prior knowledge. SARA and DSARA learn continuously about the environment. To the best of our knowledge, no other paper (see Table II) has proposed an approach based on RL and DRL that jointly performs AC and RA, differentiates core and edge nodes, and considers the 5G use cases to improve NSPs' profit and resource utilization.

## III. Architecture for Admission Control based on (Deep) Reinforcement Learning

This section describes the architecture of the proposed approach, which is illustrated in Figure 1.

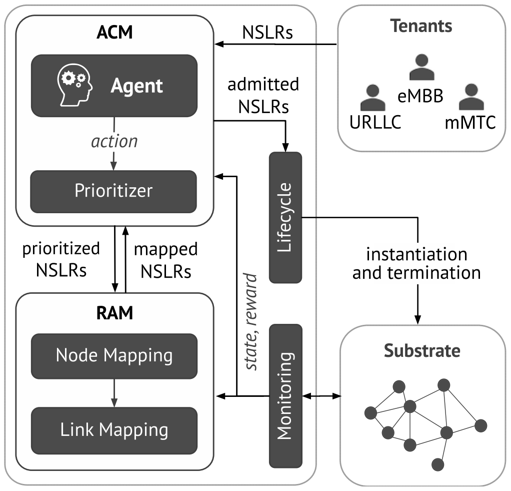


Fig. 1 Architecture for Admission Control based on RL and DRL

### A. 5G Core Network Substrate

The substrate considered in this paper follows the European Telecommunications Standards Institute recommendation in [25], which defines an NFV Infrastructure (NFVI) able to span several locations where Points of Presence (NFVI-PoPs) operate. The network between the locations is part of the NFV Infrastructure and should also be considered. NFVI-POPs are nodes that offer processing capacity; they can be either high-capacity data centers (core nodes) or small data centers close to end-users (edge nodes). Core nodes are appropriate for the 5G Control Plane since this plane involves VNFs demanding high processing and bandwidth demands. Edge nodes are adequate for the 5G User Plane, where VNFs should be located close to the end-users. VNFs composing an NSL can be placed on either edge or core nodes. The eMBB use case requires the activation of a wide range of VNFs and their distribution across core and edge nodes. URLLC instantiations require VNFs deployed at the edge to support latency requirements. The VNFs of mMTC slices, on the other hand, can be placed on core nodes since no strict latency requirements are involved.

We model the 5G core network substrate 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 $L$ 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 $BW(n_i, n_j)$.

### B. 5G Network Slice Requests

An NSLR is described by $nslr = \{s\_type, T_o, G\}$. The $s\_type$ defines the 5G use case with eMBB, URLLC, and mMTC. $T_o$ refers to the requested operational time (slice duration). $G = \{F, V\}$ 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. The weight on each edge gives the bandwidth requested by the virtual link. The processing capacity required is denoted by $cpu(vnf_i)$ and the node type requested by a VNF as $type(vnf_i)$. Similarly, $bw(vnf_i, vnf_j)$ is the bandwidth demanded by the virtual link $(vnf_i, vnf_j)$.

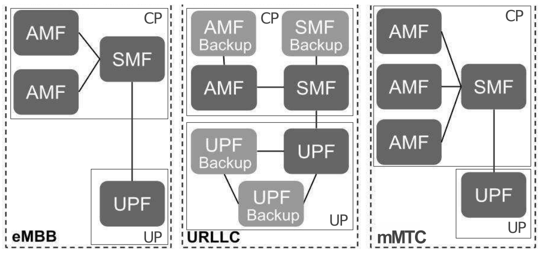


Fig. 2 NSL graphs

Figure 2 presents graphs for NSLRs of the three most common 5G use cases. The NSL graphs follow the 5G Control Plane (CP) and User Plane (UP) Separation concept for flexible deployment, independent evolution, and scalability. Each graph includes essential CP VNFs (Access and Mobility Management Function, AMF, and Session Management Function, SMF) and UP VNFs (User Plane Function, UPF). The AMF handles connections and mobility tasks, the SMF establishes, modifies, and releases data unit sessions, and the UPF performs packet routing and forwarding. The URLLC graph considers backups for AMF, SMF, and UPF that should be instantiated at different nodes close to the end-user since URLLC must offer high reliability and low latency. No backup for SMF or UPF is included for the eMBB and mMTC graphs since these service types have no ultra-high reliability requirements. The mMTC graph has more AMFs than do the other types since access to several types of devices must be provided. The eMBB graph has fewer AMFs than does the mMTC graph since it attends fewer devices than the latter.

### C. Modules

The proposed architecture includes the *Monitoring Module*, the *Admission Control Module (ACM)*, the *Resource Allocation Module (RAM)*, and the *Lifecycle Module*. Arriving NSLRs are processed in batches collected in time windows. The Monitoring Module collects information about resource availability in the network substrate (nodes and links) and, periodically, delivers this information to ACM and RAM. Information is structured as states and rewards (see Section IV).

The ACM performs the admission of NSLRs by employing an Agent and a Prioritizer. The Agent determines a normalized weight value for each 5G use case. The Prioritizer uses these values to sort the NSLRs to establish the order in which resources should be allocated. These weight values lead to the maximum profit, i.e., the Agent selects an action that, if taken, maximizes the profit. The Agent learns to select actions that increase profit by considering the information on states and rewards from interaction with the environment. A state represents the available resources after the Agent executes an action, and a reward provides the profit generated by taking that action. Two agents are used for carrying out ACM; the first, based on RL, is introduced in Section IV; and the second, based on DRL, is presented in Section V.

The Prioritizer enqueues NSLRs in batches of a minimum size, obeying the proportion given by their weight values and arrival times. If, for instance, weight values are 10, 5, and 5 for the eMBB, URLLC and mMTC, respectively, then batches with 2, 1, and 1 NSLRs of type eMBB, URLLC and mMTC 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 use cases are employed to determine the number of NSLRs in subsequent batches. In the example described here, if there are 10 NSLRs per use case in the queue, there will be a sequence of 5 batches with 2, 1, and 1 NSLRs of type eMBB, URLLC, and mMTC, followed by 5 batches composed of one NSLR of URLLC, and one of mMTC. After assembling the priority queue, each NSLR is dequeued, the ACM sends a resource allocation request to the RAM. If resources are successfully allocated, then the NSLR is accepted. Otherwise, it is rejected. Dequeueing an NSLR and attempting to allocate resources to it is repeated until the priority queue is empty.

The RAM allocates resources on nodes for the VNFs composing an NSLR (node mapping); those VNFs must be available in the Network Repository Function (NRF). Afterward, the RAM allocates bandwidth in selected links for connecting the allocated nodes (link mapping). For decision-making, the RAM considers whether a node has resources available to support the demand and whether the latency and reliability requirements of the NSLR type can be supported. The time spent interacting with the NRF is considered negligible compared to that spent in node and link mapping. The RAM maps CP VNFs onto core nodes (a.k.a. cloud domain). It maps UP VNFs of URLLC onto edge nodes (a.k.a. edge domain) to satisfy strict latency requirements while mapping UP VNFs of the other use cases, preferably onto core nodes [26]. In order to meet reliability requirements, a primary VNF and its backup are never placed on the same node.

The Link Mapping procedure maps virtual links on the shortest paths in the substrate that satisfy the required bandwidth for the virtual link. If the Node Mapping and Link Mapping procedures are successful, the RAM sends a notification of successful allocation (mapped notification) to ACM. Otherwise, a non-mapped notification is sent. Upon accepting an NSLR, the Lifecycle module instantiates its VNFs and virtual links, thus, realizing an NSL. When the lifetime of the NSL expires, the resources are deallocated.

## IV. Admission Control based on RL

The approach proposed here is called SARA when the ACM employs an Agent that runs a Q-learning-based AC algorithm. Q-learning is an off-policy, model-free, and temporal-difference RL algorithm. A Q-learning agent selects an action ($a_t$) when on a state ($s_t$), performs an action, receives a reward ($R_t$), and moves to the next state ($s_{t+1}$). Q-learning uses a lookup table (Q-table) to store the quality value (Q-value) of an action in a state, as follows:

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

In Equation 1, $\alpha$ is the learning rate with values between 0 and 1; it defines the Q-values retained by the RL agent. If $\alpha = 1$, the RL agent discards the old Q-values; if $\alpha = 0$, the most recently learned Q-value is discarded (*i.e.,* the RL agent learns nothing new). $\gamma$ is the discount factor used to balance the immediate reward and the maximum expected future reward. Its value is in the interval $[0, 1]$. If the discount factor is 1, the RL agent values the expected future reward.

The following subsections define the state space, the action space, the exploration and exploitation method, the reward function, and the Q-learning-based AC algorithm to specify SARA.

### A. State Space

A state represents the available resources in the 5G core network substrate. The State Space $S$ is the set of all states that the RL agent can experience. 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 of the set of edge ($E$) and core nodes ($C$), respectively, and $bw(L)$ is the available bandwidth in the set of links ($L$). For instance, the state $\{80, 50, 60\}$ indicates that $80\%$ and $50\%$ of the total capacity for processing are available in $E$ and $C$, respectively, while $60\%$ of the total bandwidth is available in $L$.

### B. Action Space

The Action Space $A$ is the set of all actions an agent can take. In every step, the agent chooses the action $a$ which provides the maximum profit return. An action $a \in A$ is denoted by $a = \{w_{embb}, w_{urllc}, w_{mmtc}\}$, where $w_{embb}$, $w_{urllc}$, and $w_{mmtc}$ are the weights for eMBB, URLLC, and mMTC. These weights are restricted to take integer values in the interval $[0, 10]$ to reduce the action space size. As an example, let us consider the following action: $a_i = \{10, 5, 5\}$. This action indicates that SARA should prioritize NSLRs of eMBB type. Specifically, it should admit one NSLR of URLLC and one of mMTC for every two NSLRs of eMBB.

### C. Reward

The reward value expresses the monetary profit obtained by taking an action in a state. The action taken implies the acceptance of NSLRs of different types and consumes resources having distinct costs. Typically, resources at core nodes are abundant and cheaper than those at edge nodes. The RL agent maximizes the NSP profit while optimizing resource utilization. A reward value considers the profit generated after the RL agent takes an action. The profit obtained $p(nsl)$ by the acceptance of an NSL is the amount of money earned by the NSP for selling the NSL minus the operational cost of processing and bandwidth resources; it is given by:

$$p(nsl_i) = (rev_i - cst_i) \times T_o, \quad (2)$$

$$cst_i = \sum_{j=0}^{m} cpu(vnf_j) \times f_{cpuj} + \sum_{j=0}^{n} bw(v_j) \times f_{bw} \times h, \quad (3)$$

where:

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

$cst$ - 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 virtual link $v_j$ in $nsl_i$
$f_{cpuj}$ is the processing cost of $vnf_j$, 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, $R$, is given by:

$$R = \frac{\sum_{i=0}^{k} p(nsl_i)}{maxP(SN,T)}, \quad (4)$$

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

## D. Exploration and Exploitation Method

The RL agent uses the epsilon-greedy method (Equation 5) to explore or exploit an action in each step. To choose an action, 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 (*i.e.,* exploit a past optimal action). Otherwise, it chooses a random action (*i.e.,* explore a new action).

$$a = \begin{cases} \max_{a} Q_t(s_t,a), & if \quad rn > \varepsilon \\ random\,action, & otherwise. \end{cases} \quad (5)$$

## E. RL-based Admission Control Algorithm

We use the Q-learning algorithm to realize the RL agent of SARA. Algorithm 1 aims at admitting the most profitable set of NSLRs. Time is discretized. The algorithm receives as input the NSLRs arriving in a time window, $RS = \{nslr_1, ....nslr_n\}$. Data employed by the algorithm include the Action Space ($A$) and the State Space ($S$) as well as the parameters learning rate ($\alpha$), discount factor ($\gamma$), exploration-exploitation factor ($\varepsilon$), number of steps ($m$), and number of learning episodes ($n$). A step is the completion of one interaction of the agent with the environment (*i.e.,* state, action, and reward), and $m$ steps comprise a learning episode. After $n$ episodes, Algorithm 1 produces as output the weight values for each 5G use case.

Algorithm 1 creates a Q-table (Line 1) with dimensions $|S| \cdot |A|$. 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 $n$-episodes and an inner loop through $m$-steps (Line 4). In the inner loop, the RL agent uses the $\varepsilon$-greedy method to select either a random action or an optimal action $a_t$. The optimal action allows the RL agent to exploit learned knowledge. Since Q-values depend on past rewards (*i.e.,* 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 use case. NSLRs are dequeued, and then a request is sent to the RAM. If resources are allocated, the NSLR is mapped (Line 8) onto the substrate, *i.e.* it 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 $a_t$ and observes the new state $s_{t+1}$ (*i.e.,* new processing and bandwidth capacities available in the substrate after executing $a_t$). The RL agent updates the Q-table. The state $s_{t+1}$ becomes the current state $s_t$, and the RL agent starts a new iteration.

**Algorithm 1:** RL-based AC Algorithm

```
Data   : Learning parameters
Input  : Sets of NSLRs ($RS$) collected during the time window
Result: Admission of NSLRs that generate the maximum profit
1  Initialize $Q : Q(S,A) = 0, \forall s \in S, \forall a \in A$
2  for $episode \leftarrow 1$ to $n$ do
3      The agent observes the initial state $s_i$ // when 100% of
         substrate resources are available;
4      while next state $s_{t+1}$ is not the final state do
5          The agent chooses $a_t$ using $\varepsilon$-greedy exploration method (Equation
             5) // Selection of a random or an optimal
             action that increases the profit;
6          The agent invokes the Prioritizer to sort the NSLRs $PR$ into a
             priority queue;
7          for each $nslr \in PR$ do
8              The agent invokes the RAM that runs Algorithm 3 to map
                 $nslr$;
9              if $nslr$ is mapped then
10                 The agent admits $nslr$ and sends information to
                     Lifecycle;
11             end
12             else
13                 The agent rejects $nslr$;
14             end
15         end
16         The agent observes the reward $R_t$ that is calculated by using
             Equation 4;
17         The agent observes the new state $s_{t+1}$ // The current
             resource availability;
18         The Q-table is updated according to Equation 1;
19         The current state is updated $s_t \leftarrow s_{t+1}$;
20     end
21 end
```

The complexity of Algorithm 1 depends on the number of state-action pairs $s$, the finite number of visits to each pair the agent needs to learn $r$, and the complexity of Algorithm 3 with a complexity of $\mathcal{O}(n+n\log n+fn+v(n+l))$ (see Section VI), where $v$, $n$, $l$, and $f$ are the number of virtual links, substrate nodes, substrate links, and VNFs, respectively. Algorithm 1 executes Algorithm 3 $sr$ times for RA. Thus, the complexity of Algorithm 1 is $\mathcal{O}(sr(n+n\log n+fn+v(n+l)))$.

## V. Admission Control based on DRL

Q-learning algorithms face two limitations in large environments: long convergence times since the agent has to experience more state-action pairs several times before learning and high demand for memory capacity to store Q-values. DQN, a DRL algorithm, deals with such limitations by using a function approximator (usually, NN) for generalizing the knowledge learned from some already visited states to other similar states. An NN allows the DQN agent to learn from a few interactions with the environment. To extend our proposal to cope with large environments, we have replaced the Q-learning agent with a DQN agent. This new approach is named DSARA.

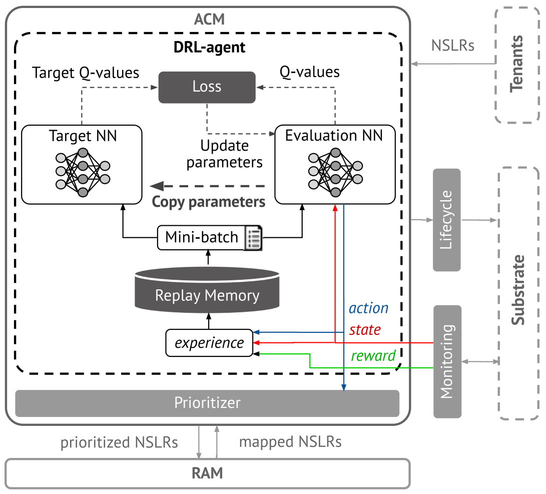


Fig. 3 DRL-based ACM

The DQN agent takes action $a_t$ with the best estimated Q-value, receives a reward $R_t$ for the action taken, and observes the new state $s_{t+1}$. As Figure 3 shows, DSARA runs an AC algorithm based on an enhanced DQN [27] composed of Replay Memory, Evaluation NN, and Target NN to learn the optimal admission action for each learning step, as well as achieve learning stability. Classic DQN uses a single NN to estimate Q-values and Target Q-values, which leads to a high correlation in their estimations [27]. DSARA stores the experiences $(s_t, a_t, s_{t+1}, R_t)$ in the Replay Memory to later use a mini-batch to train the Evaluation NN. The Evaluation NN estimates the Q-values ($Q(s_t, a_t)$), while the Target NN estimates the Target Q-values (Equation 6). The target Q-values serve as labels for calculating the loss when training the Evaluation NN. DSARA updates the Target NN periodically with the Evaluation NN weights since the latter is iteratively trained [28].

$$Q^{+}(s_t, a_t) = R_t + \gamma \cdot maxQ(s_{t+1}, a). \tag{6}$$

Next, we define the state space, action space, reward, exploration and exploitation method, NNs, and DQN-based AC algorithm to specify DSARA.

## A. State Space, Action Space, and Reward

DSARA employs the same reward function and structure of actions used by SARA. The DSARA State Space $S$ is the set of all states the DRL agent can experience. A state $s \in S$ is defined by: $\{cpu(E), cpu(C), bw(L), n_e, n_u, n_m\}$. $cpu(E)$, $cpu(C)$, and $bw(L)$ represents, as in SARA, the available resources in the substrate of the 5G core network. $n_e$, $n_u$, and $n_m$ indicate the number of NSLs running eMBB, URLLC, and mMTC, respectively.

## B. Exploration and Exploitation Method

DSARA uses an epsilon-greedy method with decaying $\varepsilon$ to move from a more exploratory policy to a more exploitative one. The DRL agent initially explores with decays by using Equation 7; where $\varepsilon_{max}$ is the maximum probability of exploration, used at the start of the training, $nsteps_t$ is the number of steps the agent has experienced, and $dr$ is the decay rate, a constant to reduce $\varepsilon$ linearly over time. The DRL agent then generates a random number $rn \in [0, 1]$. Finally, the DRL agent uses Equation 5 to select the action to be executed.

$$\varepsilon_t = \varepsilon_{max} - (nsteps_t \times dr). \tag{7}$$

## C. Neural Networks

The structure of Evaluation NN and Target NN is as follows. An input layer of 6 neurons (*i.e.,* one neuron per variable in the state tuple $s$), a single hidden layer of 150 neurons with a single hidden layer being sufficient to approximate a target function, as shown in [29], and an output layer of 30 neurons (one neuron per action in the Action Space $A$). Each neuron in the output layer estimates the Q-value, $Q_i$, associated with the action $a_i \in A$.

DSARA uses the gradient descent and backpropagation approach to reduce the loss in Q-value estimations by adjusting weights and biases in each layer of the Evaluation NN. The loss (also called error or cost, Equation 8) is the difference between the Q-value estimated by the Evaluation NN and the Target Q-value obtained by the Target NN. DSARA fixes the Target NN temporarily and updates it periodically with the trained parameters of the Evaluation NN for learning stability reasons [28] [27].

$$Loss = (Q^{+}(s_t, a_t) - Q(s_t, a_t))^2. \tag{8}$$

## D. DRL-based Admission Control Algorithm

We use the DQN algorithm to realize the DRL agent of DSARA. Algorithm 2 aims at admitting the most profitable NSLRs. Time is discretized. The algorithm input is a set of NSLRs ($RS$) arriving in a given time window. The output is the admitted NSLRs which maximize the profit. The data employed by this algorithm include the number of neurons in the hidden layer, the discount factor ($\gamma$), the decay rate ($dr$), maximum exploration ($\varepsilon_{max}$), training start ($k$), mini-batch size ($sz$), Target NN update ($C$), Action Space ($A$), State Space ($S$), and the number of episodes ($n$).

Algorithm 2 starts by initializing the Evaluation NN, Target NN, and Replay Memory (Line 1). The algorithm has an outer loop (Line 2) that goes through $n$ episodes and an inner loop (Line 4) that goes through $m$ steps. In the inner loop (Lines 4 to 23), the DRL agent updates its exploration probability $\varepsilon$ with decaying (Line 5, Equation 7). Then, the DRL agent uses the exploration method to select either a random action or an optimal action $a_t$ for the current state $s_t$ (Line 6, Equation 5). $a_t$ contains the weight of each use case that the Prioritizer uses to sort the NSLRs which have arrived in a priority queue (Line 7). In Lines 8 to 16, NSLRs are dequeued, and a request is sent to the RAM. If an NSLR is successfully mapped, it is admitted, and a notification is sent to the Lifecycle module (Line 11).

In Line 17, the DRL agent receives the reward $R_t$ from the environment and observes a new state $s_{t+1}$. The experience $e_t = (s_t, a_t, s_{t+1}, R_t)$ is stored in the Replay Memory $D$ (Line 18). Lines 19 to 21 are executed after $k$ experiences are stored. The DRL agent then randomly takes a set of experiences (mini-batch) from $D$ to train the Evaluation NN (Line 19); this enables the DRL agent to learn from a reduced number of interactions with the environment. By taking the fields $s_t$, $R_t$ and $s_{t+1}$ from the experiences, the DRL agent estimates $Q(s_t, a_t)$ by using the Evaluation NN and calculates with the Target NN (see Equation 6) the target $Q^+(s_t, a_t)$ (Line 20). The DRL agent uses the gradient descent and backpropagation approach for adjusting the weights and biases of the Evaluation NN in order to reduce the loss (Line 21, Equation 8); a low loss means that the estimations of the DRL agent are accurate. The new state becomes the current state, and the DRL agent begins a new iteration (Line 22). Note that for stability in learning, the Target NN parameters (*i.e.,* weights and biases) stay fixed and are replaced with the parameters of the Evaluation NN after every number of episodes $C$.

**Algorithm 2:** DRL-based AC Algorithm

```
Data   : Learning parameters
Input  : Sets of NSLRs ($RS$) collected during time windows
Result: Admitted NSLRs that generates the maximum profit
1  Initialize: Evaluation NN $Q$ with weights $\theta$, Target NN $\hat{Q}$ with weights $\hat{\theta}$,
     and Replay Memory $D$
2  for $episode \leftarrow 1$ to $n$ do
3      The agent observes the initial state $s_i$;
4      while next state $s_{t+1}$ is not final state do
5          Update exploration probability $\varepsilon$ by using Equation 7;
6          The agent selects action $a_t$ according to Equation 5;
7          The agent invokes the Prioritizer to sort the NSLRs into a priority
             queue $PR$;
8          for each $nslr \in PR$ do
9              The agent invokes RAM that runs Algorithm 3 to map $nslr$;
10             if $nslr$ is mapped then
11                 The agent admits $nslr$ and sends it to Lifecycle;
12             end
13             else
14                 The agent rejects $nslr$;
15             end
16         end
17         The agent receives reward $R_t$ (Equation 4) and observes the new
             state $s_{t+1}$;
18         Store experience $e_t = (s_t, a_t, s_{t+1}, R_t)$ into $D$;
19         Sample random mini-batch of experiences from $D$;
20         Calculate $Q^+(s_t, a_t) = R_t + \gamma \cdot maxQ^+(s_{t+1}, a)$, if state is
             final state, $Q^+(s_t, a_t) = R_t$ ;
21         Minimize loss function (Equation 8) by gradient descent and
             updates the weights $\theta$ of the Evaluation NN $Q$;
22         Update the current state $s_t \leftarrow s_{t+1}$;
23     end
24     Every C episodes, the agent copies weights and biases from Evaluation
         NN to Target NN ($\hat{Q} = Q$)
25 end
```

Algorithm 2 has a structure similar to that of Algorithm 1, with a complexity of $\mathcal{O}(sr(n+n\log n+fn+v(n+l)))$, since it depends on the times $r$ the agent needs to visit the state-action pairs $s$ for training and the complexity of Algorithm 3 (see Section VI). Algorithm 2 employs Algorithm 3 for RA.

## VI. Resource Allocation

The RAM aims at allocating resources to NSLRs by mapping (embedding) $M : G = \{F, V\} \rightarrow SN' = \{N', L'\}$, where $N' \in N$ and $L' \in L$. The latency, bandwidth, processing, and reliability requirements must be met. Algorithm 3 describes the RA used by SARA and DSARA; it is carried out in two steps: node mapping and link mapping. The input of Algorithm 3 is a $nslr = \{s\_type, T_o, G\}$ provided by the ACM. The output is a notification on either successful or unsuccessful allocation of resources. Information about the allocation of resources for an accepted $nslr$ is passed on to the Lifecycle module.

**Algorithm 3:** RA Algorithm

```
Input  : An NSLR, substrate resource availability
Result: Mapped NSLR
1  for each substrate node $n \in N$ do
2      Calculate embedding potential (Equation 9);
3  end
4  Rank the substrate nodes $N$ according to the embedding potential value in
     descending order
5  for each $vnf \in F$ do
6      for each node $n$ in the ranked list do
7          if $match(type(n), type(vnf), s\_type) == True$ and
             $isAllowed(n, vnf) == True$ and $\frac{CPU(n)}{cpu(vnf)} \geq 1$ then
8              Map $vnf$ onto $n$;
9              Break;
10         end
11     end
12     if $vnf$ is not mapped then
13         Return non-mapped notification;
14     end
15 end
16 for each virtual link $v \in V$ do
17     Obtain source $src$ and destination $dst$ from $v$;
18     Compute candidate paths from $src$ to $dst$;
19     Sort the paths into the list $CandidatePaths$ regarding the number of
         hops // The first path in the list has the lowest
         number of hops;
20     for each path $CP$ in $CandidatePaths$ do
21         if each link $l \in CP$ satisfies $\frac{BW(l)}{bw(v)} \geq 1$ then
22             Map $v$ onto $CP$;
23             Break;
24         end
25     end
26     if $v$ is not mapped then
27         Return non-mapped notification;
28     end
29 end
30 Return mapped $NSLR$;
```

### A. Node Mapping

The Node Mapping step, Lines 1 to 15, aims at mapping the VNFs of an NSLR onto nodes in the substrate network ($F \rightarrow N'$) while respecting the processing, latency, and reliability requirements. Each node in $F$ represents a VNF, and its label gives the processing requirement ($cpu(vnf)$). The use case type ($s\_type$ ) is used to choose the type of node in $SN$.

Node Mapping considers the NSL graphs defined in Figure 2, especially, the VNFs, which are mapped according to their $type(vnf)$ (CP or UP) and $s\_type$ (eMBB, mMTC, or URLLC). A VNF belonging to the CP is always mapped onto a core node. VNFs of type UP are mapped according to the use case. For URLLC, UP VNFs are mapped onto edge nodes to satisfy the strict latency requirements. For mMTC, UP VNFs are always mapped onto core nodes since this use case is not latency-sensitive. For eMBB, UP VNFs are preferentially mapped onto core nodes, although they can be mapped onto edge nodes when core nodes are unavailable. Overall, the backup VNFs are mapped onto different nodes to satisfy reliability requirements.

To perform node mapping, substrate nodes, $n \in SN$, are ordered according to their potential embedding value, $EP$, according to Equation 9 (Lines 1 to 3). The $EP$ metric reflects the capacity of a substrate node to embed a VNF on the basis of its available processing capacity $CPU(n_i)$ and available bandwidth $BW(l_j)$ [30] [12]. The RAM sorts the substrate nodes in decreasing order of $EP$ value.

$$EP(n_i) = cpu(n_i) \times \sum_{l \in adj(n_i)} bw(l_j). \tag{9}$$

Lines 5 to 15 attempt to map each VNF $vnf \in F$ onto a substrate node, $n$. Three conditions need to be satisfied for a successful mapping (Line 7): i) the node type ($type(n)$) needs to match with the use case of the NSLR ($s\_type$) and the VNF type ($type(vnf)$), ii) a backup of a VNF should not be placed on the same node as the primary VNF, and iii) the available processing capacity is greater than the processing requirement ($\frac{CPU(n)}{cpu(vnf)} \geq 1$). If all the conditions are satisfied, the $vnf$ is mapped onto $n$, and the node resources are allocated. Otherwise, the RAM visits the next node in the ranked list to verify if the mapping conditions hold. If the RAM cannot map at least one VNF, it returns a *non-mapped* notification for $nslr$ to the ACM and deallocates all the nodes that have been allocated to the NSLR. If all the VNFs are successfully mapped, the RAM continues, and the Link Mapping step is carried out.

### B. Link Mapping

The Link Mapping procedure attempts to map virtual links onto substrate links, $V \rightarrow L'$, consuming the least possible amount of bandwidth. Paths in the substrate with available bandwidth satisfying the bandwidth requirements and with the fewest hops are sought to minimize network bandwidth utilization. In Lines 16 to 29, the RAM carries out the Link Mapping procedure. For each virtual link, $v \in V$, an attempt is made to map it onto a substrate path. The RAM computes the candidate paths between source $src$ and destination $dst$ nodes (Line 18). In Line 19, the RAM sorts the paths in descending order by the number of hops and puts them in the $CandidatePaths$ queue. Subsequently, the RAM takes from $CandidatePaths$ the first path $CP$ (*i.e.,* the shortest path in terms of the number of hops). The RAM then verifies if the bandwidth availability for all the links along $CP$ satisfies the NSLR bandwidth requirement $\frac{BW(l)}{bw(v)} \geq 1$. If all links meet this condition, the RAM maps $v$ onto the links of $CP$; otherwise, the RAM considers the next shortest path. If the RAM cannot map at least one virtual link, it returns a *non-mapped* notification to the ACM, and the nodes are deallocated. Otherwise, Node Mapping and Link Mapping are finished successfully, and the RAM sends a mapped notification to the ACM.

### C. Algorithm Complexity

The complexity of Algorithm 3 depends on the following elements. The first loop (Line 1) has a complexity of $\mathcal{O}(n)$, where $n$ is the number of substrate nodes. The node ranking in Line 4 uses the $timsort$ algorithm with worst-case complexity $\mathcal{O}(n \log n)$. The second loop (Line 5) has a complexity of $\mathcal{O}(fn)$, where $f$ is the number of VNFs. The third loop (Line 16) has a complexity of $\mathcal{O}(v(n + l))$. This loop computes the candidate paths $v$ times (the number of virtual links) by using a breadth-first search from source to destination with a complexity of $\mathcal{O}(n + l)$, where $l$ is the number of substrate links. Thus, the complexity of Algorithm 3 is $\mathcal{O}(n+n \log n+ fn + v(n + l))$.

## VII. Performance Evaluation

This section describes the evaluation of SARA and DSARA. First, the metrics used for assessing their performance are introduced. Then, the detailing of the setup for the experiments is presented, and finally, the results obtained are discussed.

### A. Metrics

The metrics employed in the evaluation are profit, resource utilization, and acceptance ratio. The profit $P$ is calculated according to Equation 2. The resource utilization is given by:

$$U = \frac{\frac{\sum_j cpu(nsl_j)}{CPU(SN)} + \frac{\sum_j bw(nsl_j)}{BW(SN)}}{2}, \tag{10}$$

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 NSLs ($req_a$) and the number of NSLRs ($req_t$).

$$AR = \frac{req_a}{req_t}. \tag{11}$$

### B. Experimental Setup

The architectural modules were developed using Python 3. We also developed a Python-based discrete event simulator to test SARA and DSARA, which was executed on an Ubuntu 16.04 LTS desktop with an Intel Core i5-4570 CPU and 15.5 GB RAM. The simulator uses the NetworkX library to create and manipulate the graphs of NSLRs and the substrate network topology. For the RL agent, the value of $\alpha$, $\gamma$, $\varepsilon$, and the number of state-action pairs were set to 0.9, 0.9, 0.1, $10^4$, respectively. For the DRL agent, the value of $\gamma$, $\varepsilon_{max}$, $dr$, training start, mini-batch size, and number of state-action pairs were set to 0.9, 1, 1/1000, 300 steps, 15 samples, and $3 \times 10^7$, respectively. Moreover, the Target NN update was set for every 10 episodes ($C = 150$ steps).

Experiments were conducted with different topologies (16, 32, and 64-nodes) generated by using the Barabasi-Alberth algorithm. This paper gives results for a 16-node network topology composed of 4 core nodes and 12 edge nodes, with capacities of 300 and 100 processing units, respectively. All substrate links had a capacity of 100 bandwidth units. The results for this topology coincided with the results for the other topologies employed in the evaluation. The computational demand of the VNFs was 5 processing units. The virtual links in the eMBB, URLLC, and mMTC graphs required 3, 2, and 1 bandwidth units, respectively. The operational time of the NSLRs followed an exponential distribution with a mean of twelve units of time. The arrivals for the three types of NSLR respected a Poisson process and had the same arrival rate. The total arrival rate was varied from 1 to 100 requests per time unit to assess the performance of SARA and DSARA under different load conditions. The duration of the window was 2 time units. Thirty-three repetitions were conducted to obtain results with a 95% confidence level.

We compared SARA and DSARA to the algorithms AAR and NR. The AAR algorithm obeys two criteria: (*i*) it processes NSLRs in a FIFO manner; and (*ii*) it admits an NSLR if sufficient resources are available to instantiate its VNFs and virtual links. In addition to the criteria mentioned above, the NR algorithm ranks the substrate nodes according to their embedding potential, defined in [30] [12] to enhance admission and resource allocation. Unlike SARA and DSARA, the NR and AAR algorithms did not use RL (or any machine learning model) to learn the target admission policy. We did not compare SARA and DSARA to any RL-based work focused on admission control for the RAN since its results could not be extensible to the network core in which our solutions operate.

## C. *Results and Analysis*

Figure 4a depicts the profit as a function of time (episodes) for an arrival rate of 20 requests per time unit. The profit obtained by SARA and DSARA increased from episodes 1 to 12 and from episodes 1 to 55, respectively, at which time they converged. Overall, DSARA's profit is 3%, 10%, and 14.3% greater than that obtained by SARA, NR, and AAR. The profit values of the DSARA are due to the employment of the improved DQN-algorithm that quickly learns the most profitable combination of NSLRs. The DSARA approach resulted in higher profit values than that of SARA because the DRL-agent of the former can deal with $\approx 10^4$ state-action pairs, while the RL-agent can only cope with $\approx 3 \times 10^7$ state-action pairs. The additional states and actions give DSARA more options for selecting actions that will lead to the greatest profit.

Figure 4b shows the profit as a function of the total arrival rate. Under low loads, SARA and DSARA obtained lower profit values than those resulting from AAR and NR. If the arrival rate is too low, the number of arrivals of NSLRs is insufficient to provide helpful information to the agents of RL and DRL. On the other hand, for arrival rates equal to or greater than 7 requests per time unit, DSARA obtained the most significant profit. For example, for 25 requests per time unit, DSARA's profits were 3.2%, 9.7%, and 13.2% greater than those obtained by SARA, NR, and AAR, respectively. DSARA outperforms SARA, NR, and AAR for medium to high loads because its DQN agent learned to admit the proper proportion of NSLR types.

Figure 4c presents the contribution of each type of admitted NSL to the total profit obtained by SARA and DSARA for a load of 20 requests per time unit. The contribution of URLLC NSLs to the total profit increased from episodes 1 to 12 and from episodes 1 to 55 when SARA and DSARA obtained the maximum profit. Conversely, the profit of NSLRs of eMBB and mMTC decreased slightly from episodes 1 to 12 and from 1 to 55 when the results of SARA and DSARA converged. The Q-learning and DQN agents had learned to identify the NSLRs that generated the highest profit value.

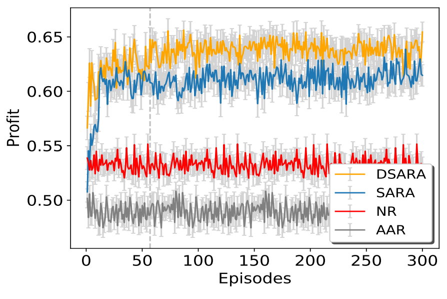
Figure 5a shows the acceptance ratio as a function of time. Overall, DSARA attained the highest acceptance ratio. Before converging, DSARA produced acceptance ratios greater than those achieved by AAR and NR and slightly higher than those obtained by SARA. After converging, DSARA produced acceptance ratios 2%, 8%, and 12% greater than those achieved by SARA, NR, and AAR, respectively. Despite the low acceptance ratio values, Figure 4a shows that they did not impact negatively on the profit.

Figure 5b depicts the acceptance ratio as a function of the arrival rate. The four algorithms produced low acceptance ratios when the arrival rate was high due to the limited resources available in the substrate. SARA and DSARA obtained values of acceptance ratios similar. These values are slightly higher than those given by NR and AAR for 7 requests per time unit. This small difference resulted from the primary goal of SARA and DSARA to increase the profit rather than the acceptance ratio. Various other factors influence the profit, including the quantity of NSLRs accepted, the operational time, and the cost of the resources consumed. If SARA and DSARA accept requests requiring many resources and long operational times, the substrate will be busy for a long time, preventing the acceptance of new NSLRs.

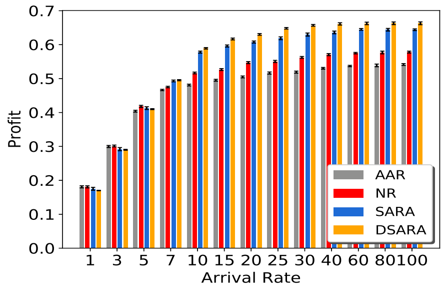
Figure 5c presents the acceptance ratio per use case. For the URLLC NSLR, the acceptance ratio of SARA and DSARA increased from the first episode to the convergence point, at which they achieved the maximum value. After that, the number of admitted eMBB and mMTC NSLRs decreased over time because the RL and DRL agents had learned that these types of requests were less profitable (especially the mMTC requests). Figures 5c and 4c corroborate the fact that SARA and DSARA accept a higher proportion of NSLRs of the URLLC type than do the other use cases.

Figure 6a presents network resource utilization as a function of time for an arrival rate of 20 requests per time unit. The values produced by NR and AAR did not evolve because they made decisions without learning from the environment. Conversely, the utilization produced by SARA and DSARA increased rapidly from episodes 1 to 12 and from episodes 1 to 55 when they converged, respectively. After convergence, DSARA obtained the maximum resource utilization value, which was 12%, 9%, and 5% greater than that achieved by AAR, NR, and SARA, respectively.

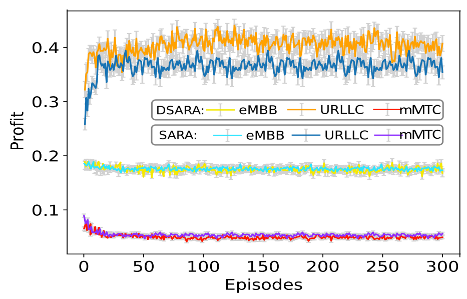
Figure 6b depicts the resource utilization resulting from the use of the SARA and DSARA algorithms in core nodes, edge nodes, and links. The utilization of edge nodes by the two algorithms also increased from episodes 1 to 12 and from episodes 1 to 55 when they converged, respectively. The RL and DRL agents had learned to prioritize the URLLC NSLRs, which used more edge resources than core resources, thus resulting in an increased profit, as shown in Figure 4a.



(a) Profit vs Episode

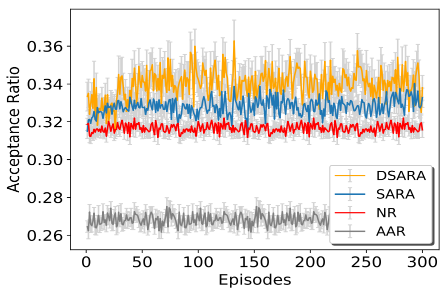


(b) Profit vs Arrival Rate

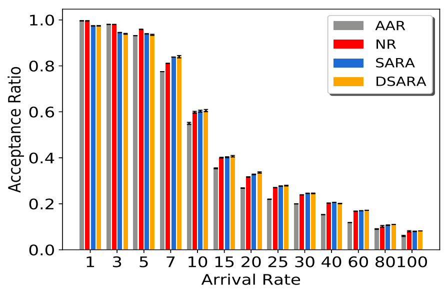


(c) Profit per NSLR Type vs Episode

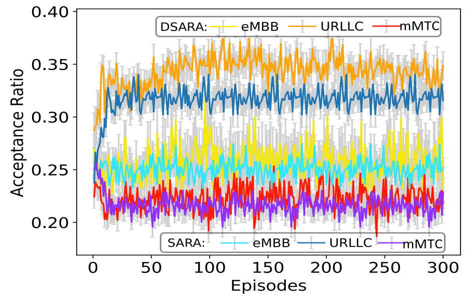
Fig. 4  Profit Results



(a) Acceptance Ratio vs Episode

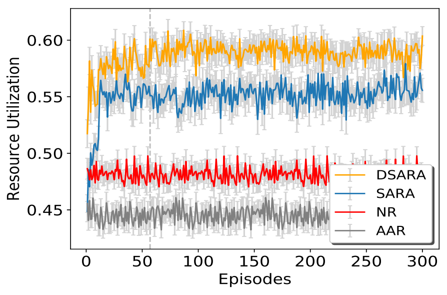


(b) Acceptance Ratio vs Arrival Rate

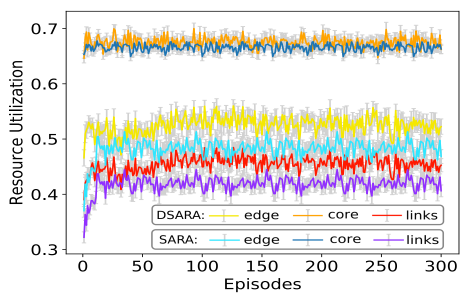


(c) Acceptance per NSLR Type vs Episode

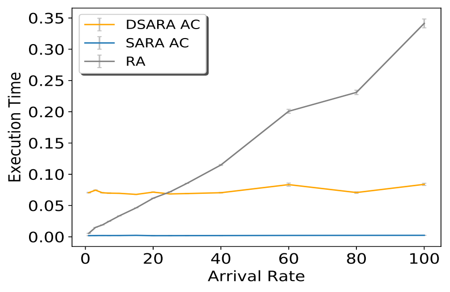
Fig. 5  Acceptance Ratio Results



(a) Resource Utilization vs Episode



(b) Utilization per Node Type vs Episode



(c) Execution Time vs Arrival Rate

Fig. 6  Resource Utilization and Execution Time

Figure 6c depicts the execution time of the AC and RA algorithms for different arrival rates. The execution time of Algorithm 1 (based on Q-learning ) and Algorithm 2 (based on DQN) was not significantly affected by the load increase. DSARA-AC execution time was slightly longer (about $0.07s$) than that obtained by SARA-AC because the DQN-based algorithm uses NNs (Deep Learning) to learn the control admission policy. Figure 6c shows only one line for RA since SARA and DSARA use the same allocation process (Algorithm 3). The results show that the execution time of the RA algorithm increased with the load; since a higher load meant a more significant number of NSLRs arriving for performing node and link mapping.

Figure 7 depicts the profits obtained by the four evaluated algorithms for three different topologies as a function of arrival rates. Overall, the profits obtained by SARA and DSARA were more significant than those achieved by NR and AR. The profit produced by SARA was higher than that achieved by AAR and

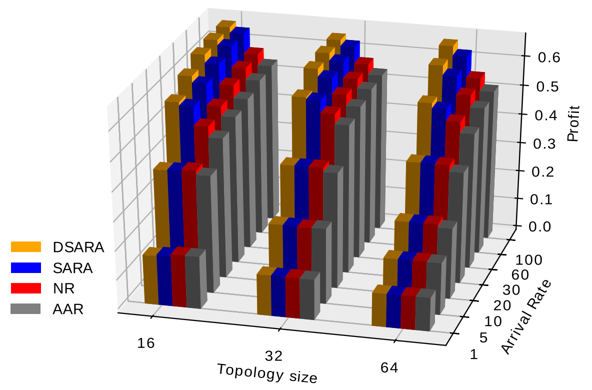


Fig. 7  Profit for different arrival rates and topologies

NR from 7, 15, and 25 requests for the 16, 32, and 64-node topologies, respectively. For instance, in the 16-node topology, a load of 7 requests per time unit was sufficient for the agent to increase the profit, although, in the 64-node topology, a load of 25 requests per time unit was needed. The profit obtained by DSARA was higher than that achieved by SARA for 10, 20, and 30 requests per time unit for 16, 32, and 64-node topologies, respectively. These profits are because the agents of Q-learning and DQN need to process abounding requests to achieve rewards from learning in large substrate networks. DSARA profits were a maximum of 3.2%, 3%, and 3.6% greater than those obtained by SARA on the 16, 32, and 64-node topologies. The gains of DSARA concerning SARA may seem low; however, they can represent a significant monetary difference.

## VIII. Conclusions and Future Work

This paper has proposed SARA and DSARA for realizing admission control and resource allocation for NSLRs for eMBB, URLLC, and mMTC in the 5G core network efficiently. SARA introduced an algorithm based on Q-learning and DSARA another based on DQN to select the most profitable NSLRs from a set of NSLRs that have arrived in a given time window. These algorithms are model-free, meaning they do not make assumptions about the substrate network as optimization-based approaches do. DSARA achieved up to 3.6%, 7.9%, and 11.7% greater profits than those obtained using SARA, NR, and AAR. Furthermore, DSARA accomplished up to 12%, 9%, and 5% higher resource utilization values than AAR, NR and SARA obtained. These results corroborate that SARA and DSARA are worth adopting by NSPs for managing network slicing.

In future work, we will enrich the AC mechanism with the DRL's latest enhancements and the RA mechanism with sophisticated strategies predicting and correlating the number of arrival requests per service with the expected network utilization for performing the resource assignment to VNFs chains. We also plan to provide admission control and adaptive resource allocation for end-to-end slices involving the RAN and core network of 5G.

## References

[1] 3GPP, "Digital cellular telecommunications system (Phase 2+) (GSM);Universal Mobile Telecommunications System (UMTS); LTE; 5G; ," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 21.915, 10 2019, version 15.0.0.

[2] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5g flexible ran," *Journal of Lightwave Technology*, 2019.

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

[4] M. R. Raza, A. Rostami, L. Wosinska, and P. Monti, "A slice admission policy based on big data analytics for multi-tenant 5g networks," *Journal of Lightwave Technology*, vol. 37, no. 7, pp. 1690–1697, 2019.

[5] M. Jiang, M. Condoluci, and T. Mahmoodi, "Network slicing management & prioritization in 5g mobile systems," in *EW Conference*. Oulu, Finland: VDE, 2016, pp. 1–6.

[6] V. Sciancalepore, X. Costa-Perez, and A. Banchs, "Rl-nsb: Reinforcement learning-based 5g network slice broker," *IEEE/ACM Trans. on Netw.*, vol. 27, no. 4, pp. 1543–1557, 2019.

[7] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration," in *ACM CoNEXT*, 2018, pp. 353–365.

[8] R. Challa, V. V. Zalyubovskiy, S. M. Raza, H. Choo, and A. De, "Network slice admission model: Tradeoff between monetization and rejections," *IEEE Systems Journal*, vol. 14, no. 1, pp. 657–660, 2019.

[9] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5g infrastructure markets: The business of network slicing," in *IEEE INFOCOM*, Atlanta, GA, USA, 2017, pp. 1–9.

[10] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, and X. Costa-Pérez, "A machine learning approach to 5g infrastructure market optimization," *IEEE TMC*, vol. 19, no. 3, pp. 498–512, 2020.

[11] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware vnf placement for service-customized 5g network slices," in *IEEE INFOCOM*, 2019, pp. 2449–2457.

[12] X. Li, C. Guo, J. Xu, L. Gupta, and R. Jain, "Towards efficiently provisioning 5g core network slice based on resource and topology attributes," *Applied Sciences*, vol. 9, no. 20, p. 4361, 2019.

[13] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "Vnf placement and resource allocation for the support of vertical services in 5g networks," *IEEE/ACM TON*, vol. 27, no. 1, pp. 433–446, 2019.

[14] W. Guan, X. Wen, L. Wang, Z. Lu, and Y. Shen, "A service-oriented deployment policy of end-to-end network slicing based on complex network theory," *IEEE Access*, vol. 6, pp. 19 691–19 701, 2018.

[15] S. D'Oro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi, and T. Melodia, "Sl-edge: Network slicing at the edge," in *ACM Mobihoc*, 2020, pp. 1–10.

[16] A. Huang, Y. Li, Y. Xiao, X. Ge, S. Sun, and H.-C. Chao, "Distributed resource allocation for network slicing of bandwidth and computational resource," in *IEEE ICC*, 2020, pp. 1–6.

[17] T. De Cola and I. Bisio, "Qos optimisation of embb services in converged 5g-satellite networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12 098–12 110, 2020.

[18] R. Li, Z. Zhao, Q. Sun, I. Chih-Lin, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74 429–74 441, 2018.

[19] Q. Liu, T. Han, and E. Moges, "Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning," *arXiv preprint arXiv:2003.12911*, 2020.

[20] G. Sun, G. O. Boateng, D. Ayepah-Mensah, G. Liu, and J. Wei, "Autonomous resource slicing for virtualized vehicular networks with d2d communications based on deep reinforcement learning," *IEEE Systems Journal*, vol. 14, no. 4, pp. 4694–4705, 2020.

[21] X. Zhang, B. Li, J. Peng, X. Pan, and Z. Zhu, "You calculate and i provision: A drl-assisted service framework to realize distributed and tenant-driven virtual network slicing," *Journal of Lightwave Technology*, vol. 39, no. 1, pp. 4–16, 2021.

[22] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang, "Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks," *IEEE Access*, vol. 7, pp. 45 758–45 772, 2019.

[23] G. Sun, K. Xiong, G. O. Boateng, D. Ayepah-Mensah, G. Liu, and W. Jiang, "Autonomous resource provisioning and resource customization for mixed traffics in virtualized radio access network," *IEEE Systems Journal*, vol. 13, no. 3, pp. 2454–2465, 2019.

[24] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *ACM Workshop on Hot Topics in Networks*, Atlanta, GA, USA, 2016, pp. 50–56.

[25] N. Etsi, "Etsi gs nfv 002 v1. 1.1 network functions virtualization (nfv)," *Architecture and Framework: ONF*, 2013.

[26] R. El Hattachi and J. Erfanian, "Ngmn 5g white paper," *NGMN Alliance, February*, 2015.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[28] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

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

[30] J. G. Herrera, "Resource allocation in an nfv/sdn-based network architecture," Ph.D. dissertation, Universidad de Antioquia, august 2018.