Admission Control for 5G 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 industry's vertical segments. Distinct use cases in 5G networks, such as enhanced Mobile Broadband, Ultra-Reliable Low Latency Communications, and Massive Internet of Things, have their Quality of Service requirements, which must be supported. In line with that, this paper proposes an approach encompassing mechanisms for Admission Control and Resource Allocation. The Admission Control mechanism introduces two solutions, one based on Reinforcement Learning (called SARA) and the other based on Deep Reinforcement Learning (named DSARA), to learn the admission policy that optimizes the profit of Network Slice Providers. SARA and DSARA consider the 5G use cases defined by the International Telecommunications Union. The Resource Allocation mechanism tries to balance the load on the network nodes as well as minimize the network resource utilization. Results show that both SARA and DSARA outperform existing mechanisms to manage network slicing in 5G networks.**

***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 [1] The International Telecommunications Union (ITU) defined several different use case categories having different Quality of Service (QoS) requirements [2]. Network Slicing is crucial for 5G to support diverse QoS requirements since it provides flexibility, modularity, and programmability to manage customized and isolated logical networks, known as slices (NSL) [3]. The employment of Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) technologies allows NSLs customization and compliance with Service Level Agreements (SLAs) [4].

Network Slice Providers (NSPs) receive NSL requests (NSLRs) to implement NSLs of distinct types such as Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communication (URLLC), and Massive IoT (MIoT), each with its own QoS requirements. NSPs need to implement an Admission Control (AC) mechanism to decide on the acceptance of NSLRs, considering the support of QoS requirements as well as the availability of physical resources. Moreover, a Resource Allocation (RA) mechanism should guarantee the resources needed by an NSL. In virtualized 5G networks, physical resources are allocated to Virtual Network Functions (VNFs) and virtual links. The acceptance of NSLRs depends on the criteria used by NSPs, such as maximization of profit as well as minimization of network utilization [5].

Several papers have addressed AC by employing different theoretical frameworks such as Queuing Theory [6], Big Data [7], Heuristics [8]–[11], Reinforcement Learning (RL) [3], [12], and Deep RL (DRL) [13]. These papers 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 recently allocated resources [14] Moreover, most of these proposals neither consider the QoS requirements of different service types (use cases) nor the allocation of resources in the 5G core network. In addition, various other solutions based on heuristics [15], [16], Queuing Theory [17], Complex Network Theory [18], Linear Programming [19], Alternating Direction Method of Multipliers [20], Artificial Neural Networks [21], and DRL [22]–[25] have considered only RA and neglected AC, although this prevents the achievement of NSP goals. On the other hand, jointly performing AC and RA in an intelligent way can considerably improve the achievement of target objectives.

This paper proposes a novel approach for AC and RA for network slicing in 5G core networks. The AC mechanism 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. These solutions employ Q-learning and Deep Q-Learning (DQN) algorithms to explore, exploit, and learn for prioritizing NSLRs collected in given time windows. Using RL and DRL, the history of slice acceptance decisions is considered rather than just accounting for the most recent parameter values. Unlike other solutions, SARA and DSARA consider the 5G use cases defined by ITU (eMBB, URLLC, and MIoT), process NSLRs in time windows to favor profit maximization, and differentiate network core nodes from edge nodes to support latency requirements of use cases. SARA and DSARA are compared with two other heuristics: the Always Admit Requests algorithm (AAR) and the Node Ranking algorithm (NR), which are overperformed by SARA and DSARA.

This work was supported by CAPES, CNPq, and the Sao Paulo Research Foundation (FAPESP) under grant #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, nfonseca@ic.unicamp.br. O. Caicedo is with the Department of Telematics, Universidad del Cauca, Popayán, Colombia. e-mail: omcaicedo@unicauca.edu.co.

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 an evaluation of SARA/DSARA. Section VIII concludes the paper and makes suggestions for future work.

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. [6] 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. [7] 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. [8] 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). Furthermore, it dynamically changes the allocation of radio resources to different NSLs according to the traffic load. Salvat et al. [10] introduce an end-to-end slicing heuristic solution that performs AC and RA of radio, edge, and cloud resources for increasing the provider profit. This approach does not learn how to improve the profit nor consider the 5G use cases defined by the ITU.

Sciancalepore et al. [9] introduce a 5G slice broker for radio resources that 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 RL is not used to make admission decisions on slice acceptance. Challa et al. [11] propose an AC model based on a knapsack problem formulation to optimize resource monetization. Bega et al. [12] introduce an analytical model based on Semi-Markov Decision Process, and a Q-learning based algorithm to perform AC for individual slicing requests. Raza et al. [3] propose an AC mechanism based on RL for maximizing provider profit; this solution considers a 5G flexible RAN and prioritizes requests with different latency requirements and expected revenues. The proposed approach employs a Q-learning agent used to decide on slice acceptance in order to maximize revenue. Bega et al. [13] have proposed 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 [3], [6]–[13] make admission decisions for individual requests as they arrive, preventing the selection of a set of NSLRs that can potentially optimize a given 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 service types. Moreover, most of the proposed mechanisms focus on allocating radio resources [26], [27] and disregarding the allocation of 5G core network nodes (Table I). Differently, SARA and DSARA focus on core and edge nodes and consider the typical standardized 5G use cases to jointly perform AC and RA.

B. Resource Allocation in 5G Network Slicing

Several investigations have addressed the RA problem in 5G. Zhang et al. [15] have introduced a heuristic for placing 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. [16] have proposed 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 as well as the topological attributes of the physical nodes; the second stage performs link provisioning by using a k-shortest path algorithm. Agarwal et al. [17] propose MaxZ, a solution for 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. [18] present a mapping algorithm for 5G network slices based on Complex Network Theory 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 a k-shortest path algorithm for the VNFs placed on the nodes. D'Oro et al. [19] have proposed a slicing framework to instantiate heterogeneous services by using Integer Linear Programming; this framework aims at optimizing CPU utilization at the network edge. Huang et al. [20] present an RA algorithm based on the Alternating Direction Method of Multipliers to reduce the latency experienced by the User Equipment in networks composed of Base Stations and Fog nodes.

De Cola et al. [21] carries out RA for eMBB slices in 5G-satellite networks by using Artificial Neural Networks to optimize QoS. Li et al. [22] propose an algorithm for network slicing in 5G RAN to optimize the spectrum efficiency and QoE. The algorithm uses DRL to learn how to allocate bandwidth to users considering fluctuations in the demand for services. Liu et al. [23] introduce a resource orchestration system that uses DRL to orchestrate networking and computing resources, aiming at optimizing utilization. This approach does not follow the ITU use cases for 5G. Sun et al. [24] propose an RA algorithm for vehicular networks based on DRL to provide QoS inefficiently in which the DRL-agent adjusts the

TABLE I Related Work

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

allocated resources for a slice as a function of the demand. Zhang et al. [25] perform slicing in data center networks by using a DRL-agent. The DRL-agent maximizes the revenue from provisioned slices while avoiding overutilized resources.

The aforecited papers [15]–[25] 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 to optimize resource utilization and maximize the 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 that can be used to train RL/DRL-algorithms [22], [28]. 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 has proposed a solution based on RL and DRL that jointly performs AC and RA, differentiates core and edge nodes, and considers the typical standardized 5G use cases (Table I). Zhang et al. [25] perform slicing in data center networks by using a DRL-agent. The DRL-agent maximizes the revenue from provisioned slices while avoiding overutilized resources.

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.

A. 5G Core Network Substrate

The substrate considered in this paper follows the European Telecommunications Standards Institute recommendation in [29], which defines an NFV Infrastructure being able to span 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, they are either high capacity data centers (core nodes) or small data centers close to end users (edge nodes).

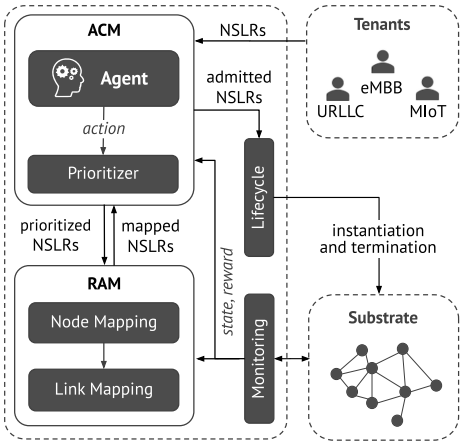


Fig. 1 RL/DRL-based Architecture for Admission Control

Core nodes are appropriate for the 5G Control Plane since this plane involves VNFs demanding high processing and bandwidth capacities [30] [31]. Edge nodes are adequate for the 5G User Plane since this involves VNFs which need to be located close to the end-users [32], [33]. Moreover, edge nodes in a Fog-RAN can involve remote radio heads and a centralized pool of virtual baseband units [34] [35].

VNFs composing an NSL can be placed on either edge nodes or core nodes. The eMBB use case requires the activation of a wide range of VNFs and their distribution across core

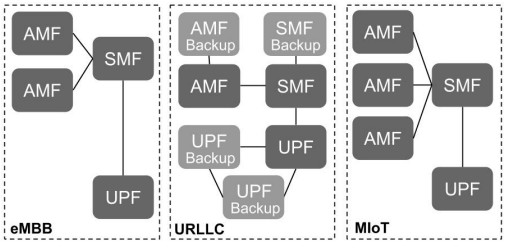


Fig. 2 NSL graphs

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 are no strict latency requirements for this use case [32].

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 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 $BW(n_i, n_j)$.

B. 5G Network Slice Requests

An NSLR is described by $nslr = \{s\_type, T_o, G\}$. The s_type defines, according to the ITU [2], the 5G use case eMBB, URLLC, or MIoT. T_o defines the requested operational time (duration of a slice). $G = \{F, V\}$ 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 a VNF requests by $type(vnf_i)$. Similarly, $bw(vnf_i, vnf_j)$ is the bandwidth demanded by the virtual link (vnf_i, vnf_j).

Figure 2 depicts graphs for the three types of use cases most common in 5G. The NSL graphs follow the 5G Control Plane (CP) and User Plane (UP) Separation (CUPS) concept for flexible deployment, independent evolution, and scalability [36]. Each graph includes essential CP VNFs (Access and Mobility Management Function, AMF) and Session Management Function, SMF) and UP VNFs (User Plane Function, UPF) [36]. 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 [32]. 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 types of device [37]. The eMBB G has fewer AMFs than the MIoT graph since the former attends fewer devices than the latter.

C. Modules

The architecture of the proposed approach includes four modules: the *Monitoring Module*, the *Admission Control Module (ACM)*, the *Resource Allocation Module (RAM)*, and the *Lifecycle Module*. The approach 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 this information to the ACM and RAM. Information is structured as states and rewards (see Section IV).

The ACM performs the admission of NSLRs. It employs an Agent and a Prioritizer. The Agent determines a normalized weight value to each type of 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 will 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 gives the profit generated by taking that action. The approach includes two agents for carrying out ACM. The first, based on RL, is detailed in Section IV; the second, based on DRL, is presented in-depth in Section V.

The Prioritizer enqueues NSLRs in batches of a 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 use cases are employed to determine the number of NSLRs in subsequent batches. In the example described, 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 MIoT, followed by 5 batches composed by one NSLR of URLLC, and one of MIoT.

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

The RAM allocates resource on nodes for the VNFs composing an NSLR (node mapping), and then, allocates bandwidth in selected links (link mapping) for connecting the allocated nodes. Decisions on node mapping consider not only whether a node has resources available to support the demand but also whether the latency and reliability requirements of the NSLR type can be supported. RAM maps CP VNFs onto core nodes. UP VNFs of URLLC are mapped onto edge nodes to satisfy strict latency requirements, while UP VNFs of the other use cases are mapped, preferably, onto core nodes [37]. In order to meet reliability requirements, a primary VNF and its backup are never placed on the same node [32].

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 successfully concluded, 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, resources are deallocated.

IV. Admission Control based on Reinforcement Learning

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 [38]. A Q-learning agent selects an action (a_t) when on a state (s_t), performs an action, receives a reward (R_t), and goes 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, α is the learning rate with values between 0 and 1, it defines the Q-values kept 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). γ 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 gives value to the expected future reward.

In the following subsections, a definition is provided for the state space, the action space, the exploration and exploitation method, and the reward function, thus specifying the Q-learning-based AC algorithm.

A. State Space

A state represents the available resources in the substrate of the 5G core network. The State Space S is the set of all states 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 in 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 is available in E and C, respectively, while 60% of the total bandwidth is available in L.

B. Action Space

For each step, the RL-agent selects the weights for each type of use case. The set of actions the RL-agent can take is denoted by A. An action a is denoted by $a = \{pct_{embb}, pct_{urllc}, pct_{miot}\}$, where pct_{embb}, pct_{urllc}, and pct_{miot} are the weights for each type of use case. The RL-agent chooses the action a which returns the maximum profit.

C. Reward

The reward value expresses the monetary profit obtained by taking an action on 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 by the acceptance of an NSL, $p(nsl)$ is the amount of money earned by the NSP for selling the NSL minus the operational cost of processing and bandwidth resources; $p(nsl)$, and 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 [39]. 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

Algorithm 1 presents the Q-learning based solution employed by SARA to perform AC. The aim is to admit the most profitable set of NSLRs. Time is discretized. The algorithm receives as input the NSLRs arriving in a given time window, $RS = \{nslr_1, ...nslr_n\}$, and produces as output the weight values for each type of use case.

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), comprised of a sequence of m-steps with each step consisting of a state, an action, and a reward received.

Algorithm 1: RL-based AC Algorithm

Data : Learning parameters (Table II)
Input : Sets of NSLRs (RS) collected during the time window
Result: Admitted 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 ε-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 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**

Algorithm 1 creates a Q-table (Line 1) with dimension $|S| \cdot |A|$. 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 that goes through m-steps (Line 4). In the inner loop, the RL-agent uses the ε-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 type of service. NSLRs are dequeued and a request is sent to 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.

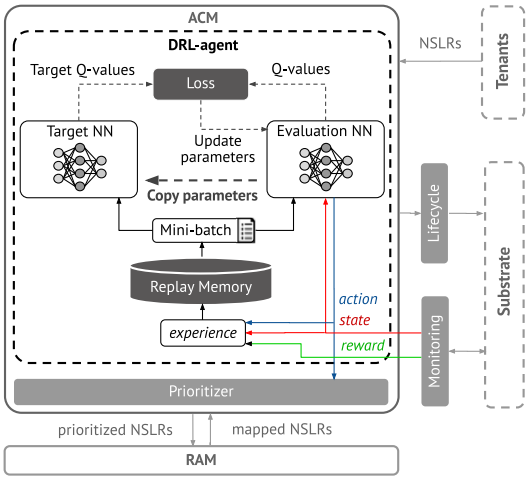


Fig. 3 DRL-based ACM

V. Admission Control based on Deep Reinforcement Learning

The approach proposed here is called DSARA when the ACM module employs an Agent which runs a DRL-based algorithm. DRL involves Deep Learning and RL [22]. Deep Learning uses Artificial Neural Networks (NNs) to allow RL to manage problems with large state and action spaces by generalizing knowledge [40]–[42], while RL allows DRL to maximize long-term performance by interacting with the environment.

DQN is a DRL algorithm that uses an NN to estimate the Q-value of each action in a particular state. The agent takes the 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} [38]. As Figure 3 shows, DSARA runs an AC algorithm based on an enhanced DQN [43] 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 [43]. DSARA stores the experiences (s_t, a_t, s_{t+1}, R_t) into 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 7). The target Q-values serve as labels for the calculation of the loss when training the Evaluation NN. DSARA updates the Target NN periodically with the Evaluation NN weights since the latter is iteratively trained [41].

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

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 $bw(L)$ represent, 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 MIoT, respectively.

B. Exploration and Exploitation Method

DSARA uses an epsilon-greedy method with decaying ε to move from a more explorative policy to a more exploitative one. The DRL-agent initially explores with decays ε by using Equation 6; where, ε_{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 ε 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.

$$\varepsilon_t = \varepsilon_{max} - (nsteps_t \times dr) \tag{6}$$

C. Neural Networks

$$Q^+(s_t, a_t) = R_t + \gamma \cdot maxQ(s_{t+1}, a) \tag{7}$$

The Evaluation NN and Target NN have an input neuron per variable in the state tuple s, a single hidden layer since it is sufficient to approximate a target function, as shown in [42], and an output neuron per action in the Action Space A. Each neuron in the output layer estimates the Q-value, Q_i, associated with an 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 [41] [43].

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

D. DRL-based Admission Control Algorithm

Algorithm 2 presents the DQN-based solution employed by DSARA to perform AC. The aim is to admit the most profitable NSLRs. Time is discretized. The algorithm input is a set of NSLRs (RS) arriving in a given time window. The output are the admitted NSLRs that maximize the profit. The data employed by this algorithm include the number of neurons in the hidden layer, the discount factor (γ), the decay rate (dr), maximum exploration (ε_{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 ε by using Equation 6 (Line 5). Then, the DRL-agent uses the exploration method (Equation 5) to select either a random action or an optimal action a_t for the current state s_t (Line 6). a_t contains the weight of each use case that the Prioritizer uses to sort the arrived NSLRs into a priority queue (Line 7). In Lines 8 to 16, NSLRs are dequeued and a request is sent to 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 into the Replay Memory D (Line 18). Lines 19 to 21 are executed only after storing k experiences. The DRL-agent 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 7) the target $Q^+(s_t, a_t)$ (Line 20). The DRL-agent uses gradient descent and backpropagation 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 only every C episodes the DRL-agent replaces them with the parameters of the Evaluation NN.

Algorithm 2: DRL-based AC Algorithm

```
Data  : Learning parameters (Table II)
Input : Sets of NSLRs (RS) collected during time windows
Result: Admitted NSLRs that generates the maximum profit
1  Initialize: Evaluation NN Q with weights θ, Target NN Q̂ with weights θ̂,
     and Replay Memory D
2  for episode ← 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 ε by using Equation 6;
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 ∈ 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 + γ · 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 θ of the Evaluation NN Q;
22         Update the current state s_t ← s_{t+1};
23     end
24     Every C episodes, the agent copies weights and biases from Evaluation
         NN to Target NN (Q̂ = Q)
25 end
```

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 all be met. Algorithm 3 describes the RA used by SARA and DSARA, which 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 all simple paths from src to dst;
19 | Sort the simple paths into the list of $CandidatePaths$ 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.

VNFs are mapped according to their $type(vnf)$ (CP or UP) and the use case (s_type) to which it belongs. A VNF belonging to the CP is always mapped onto a core node. VNFs belonging to the UP are mapped as a function of the use case as follows. For URLLC, VNFs are mapped onto edge nodes to satisfy the strict latency requirements. For MIoT, 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.

To perform node mapping, substrate nodes, $n \in SN$, are ordered according to their embedding potential value, EP, (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 $BW(l_j)$ [44] [16]. The RAM sorts the substrate nodes in decreasing order of EP value.

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

In Lines 5 to 15, an attempt is made to map the VNF $vnf \in F$ onto the substrate nodes, 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 higher 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, 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 a *non-mapped* notification for $nslr$ to the ACM and deallocates all the nodes that have been allocated to the NSLR. In case all the VNFs are successfully mapped, 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 required and with the fewest hops are sought to minimize the network bandwidth utilization.

In Lines 16 to 29, 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 module uses the depth-first search, considering the source src and destination dst nodes (Line 18), to compute all the simple paths (loop-free paths) between src and dst. In Line 19, the RAM sorts the simple paths, in descending order of their number of hops and put them in the $CandidatePaths$ queue. Subsequently, the 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 for all the 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 finish successfully, and the RAM sends a mapped notification to ACM.

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 are presented, and finally, the results obtained in the experiments are discussed.

A. Metrics

The metrics employed in the evaluation are the 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 NSLRs (req_a) and the number of NSLRs (req_t) requested.

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

B. Experiment 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 Intel Core i5-4570 CPU and 15.5 GB RAM. The simulator uses the NetworkX library [45] to create and manipulate the graphs of NSLRs and the substrate network topology. For the RL-agent, α was set to 0.9, γ to 0.9, ε to 0.1, and the number of state-action pairs in 10^4 (10 actions and 10^3 states). For the DRL-agent, we set γ to 0.9, ε_{max} to 1, dr to $1/1000$, training start to 300 steps, mini-batch size to 15 samples, the Target NN to update every 150 steps, and the number of state-action pairs to 3×10^7 (30 actions and 10^6 states). Both the Evaluation NN and Target NN were set with 6 neurons in the input layer (*i.e.,* one input neuron per variable in the state tuple s), a single hidden layer, and 30 neurons in the output layer (*i.e.,* one neuron per action in the DSARA Action Space A). Each neuron in the output layer estimates the Q-value Q_i associated with action $a_i \in A$.

Experiments were conducted with different topologies (16, 32, and 64-nodes) generated by using the Barabasi-Alberth algorithm [46]. In this paper, results are given for a 16-node network topology composed of 4 core nodes and 12 edge nodes, with a 300 and 100 processing units capacities, respectively. All substrate links had a capacity of 100 bandwidth units. The results for this topology complied with 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 MIoT graphs require 3, 2, and 1 bandwidth units, respectively. The operational time of NSLRs followed an exponential distribution with a mean of twelve time units. The arrivals processes for the three types of NSLRs respected the Poisson process and had the same arrival rate value.

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 95% confidence level. Table II summarizes the setup of the experiments.

Simulation	
Parameter	**Value**
Substrate	16, 32, 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)	from 1 to 100
Time window (time units)	2
SARA	
Learning rate (α)	0.9
Discount factor (γ)	0.9
Exploration factor (ε)	0.1
DSARA	
NNs	6, 1 hidden layer of 150 neurons, 30
Discount factor (γ)	0.9
Maximum Exploration (ε_{max})	1
Decay rate (dr)	$1/1000$
Training start (k)	300 steps
Mini-batch size (sz)	15 samples
Target NN update (C)	every 150 steps (10 episodes)

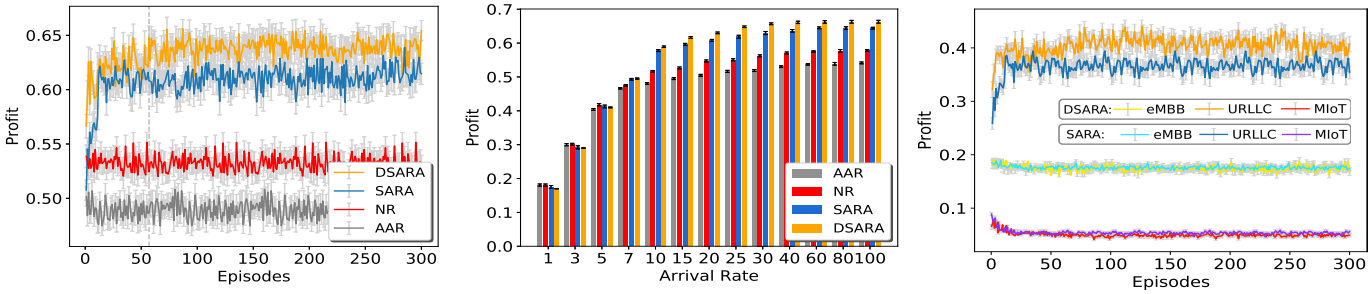
TABLE II Experiment Setup

The performance of SARA and DSARA were compared to those of two heuristics; AAR and NR. NR ranked the substrate nodes according to the embedding potential defined in [44] [16], while the ARR did not differentiate nodes for allocation. These heuristics admitted NSLRs as they arrive if there was available capacity to do so. Unlike SARA, these heuristics did not use any strategy to achieved target goals.

C. Results and Analysis

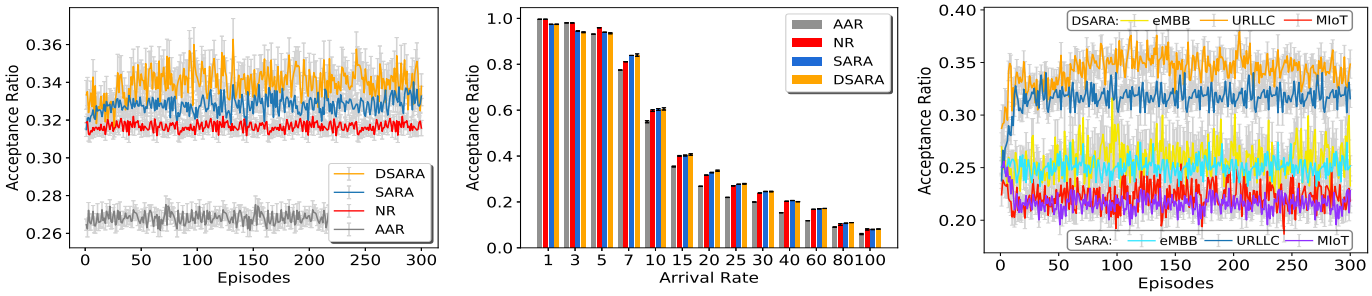
Figure 4a depicts the profit as a function of time for an arrival rate of 20 requests per time unit. The profit obtained by SARA and DSARA increased from episodes 1 to 12 and from episodes 1 to 55, respectively, when they converged. Overall, DSARA profit is 3%, 10%, and 14.3% greater than those obtained by SARA, the NR, and the AAR, respectively. These profit values from DSARA are due to the employment of a DQN-algorithm that quickly learns the most profitable combination of NSLRs. DSARA obtained higher profit values than SARA because the RL-agent deals with 10^4 state-action pairs while the DQN-agent only copes with 3×10^7 state-action pairs; more states and actions allow DSARA to select actions that lead to the highest profit.

Figure 4b shows the profit as a function of the total arrival rate. Under low loads (*e.g.,* 1 request per time unit), SARA and DSARA obtained lower profit values than those resulting from the other two heuristics. If the arrival rate is too low, the number of arrivals of NSLRs is not sufficient to provide useful information to the RL-agent and to the DRL-agent. On the other hand, for arrival rates equal to or greater than 7 requests per time unit, DSARA obtained the highest profit. For example, for 25 requests per time unit, DSARA profit was 3.2%, 9.7%, and 13.2% greater than those obtained by SARA,



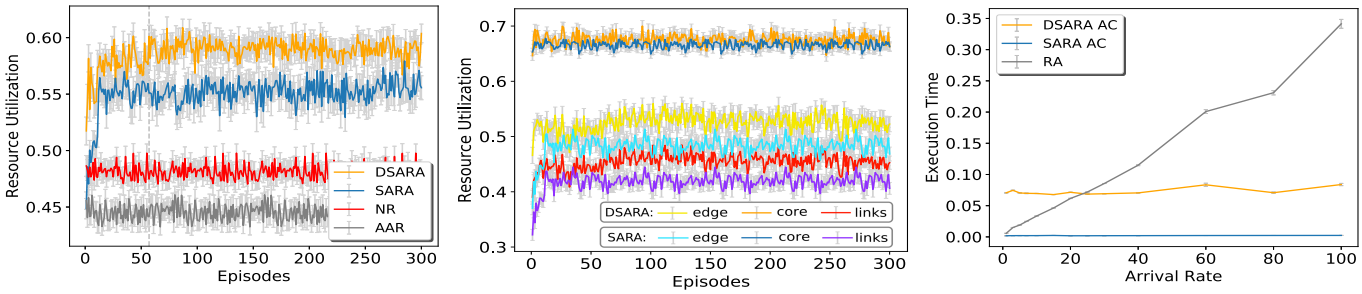
(a) Profit (b) Profit vs Arrival Rate (c) Profit per NSLR Type

Fig. 4 Profit Results



(a) Acceptance Ratio (b) Acceptance Ratio vs Arrival Rate (c) Acceptance per NSLR Type

Fig. 5 Acceptance Ratio Results



(a) Resource Utilization (b) Utilization per Node Type (c) Execution Time vs Arrival Rate

Fig. 6 Resource Utilization and Execution Time

NR, and AAR, respectively. DSARA outperforms SARA, NR, and AAR for medium to high loads because the DRL-agent learns to admit the proper proportion of NSLR type to increase the profit.

Figure 4c presents the contribution of each type of admitted NSLR to the total profit obtained by SARA and DSARA for a load of 20 requests per time unit. The contribution of URLLC NSLRs 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 MIoT decreased slightly from episodes 1 to 12 and from 1 to 55, when SARA and DSARA converged. SARA and DSARA learned to identify the NSLRs that generated the highest profit value.

Figure 5a shows the acceptance ratio as a function of time. Overall, DSARA reached the highest acceptance ratio. Before converging, DSARA produced acceptance ratios higher 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% higher 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 in the substrate. DSARA obtained values of acceptance ratio similar to those achieved by SARA. These values are slightly higher than those given by NR and AAR for 7 requests per time unit. The reason for such a small difference is that the primary goal of SARA and DSARA is to increase the profit and not the acceptance ratio. Several factors influence the profit: the quantity of NSLRs accepted, its operational time, and the cost of the resources consumed. If SARA and DSARA accept requests requiring many resources and with long operational times, the substrate will be busy for a long

time, which prevents the acceptance of new NSLRs.

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

Figure 6a presents the 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 these heuristics make 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 and 55, when they converged, respectively. After converging, DSARA obtained the maximum resource utilization, which was 12%, 9%, and 5% higher than that achieved by AAR, NR, and SARA, respectively.

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

Figure 6c depicts the execution time of the algorithms AC and RA for different arrival rates. The execution time of RL-based and DRL-based AC algorithms were not significantly affected by the load increment. DSARA-AC execution time was higher than that obtained by SARA-AC because the DQN-based algorithm must train two NNs. Figure 6c shows only one line for RA, since SARA and DSARA use the same allocation algorithm. The results show that the RA execution time increased with the load; since a higher load means a greater number of NSLRs arriving for processing in the RA phase.

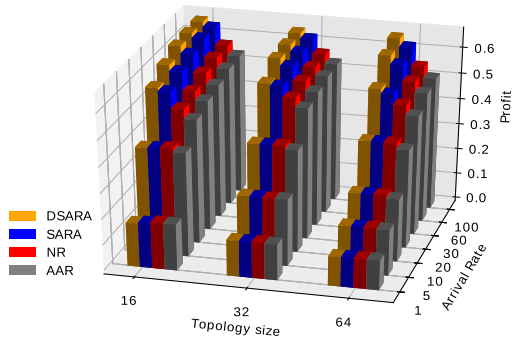


Fig. 7 Profit for different loads and topologies

Figure 7 depicts the profits obtained by DSARA, SARA, NR, and AAR for three different topologies as a function of arrival rates. Overall, the profits obtained by SARA and DSARA were greater than those achieved by NR and AR. The profit from SARA was greater than that achieved by AAR and NR from 7, 15, and 25 requests for the 16, 32, and 64-node topologies, respectively. These results are due to the fact that in large substrate networks, the RL-agent and DRL-agent needed to process many requests to achieve rewards to learn. For instance, in the 16-node topology, a load of 7 requests per time unit was sufficient for the agent to increase the profit, while in the 64-node topology, a load of 25 requests per time unit was needed. The profit obtained by DSARA was greater than that achieved by SARA under 10, 20, and 30 requests per time unit for 16, 32, and 64-node topologies, respectively. DSARA profits were a maximum of 3.2%, 3%, and 3.6% higher than those for SARA on the 16, 32, and 64-node topologies. These gains may seem to be low; however, they can represent a significant monetary difference.

VIII. Conclusions and Future Work

This paper has proposed SARA and DSARA for the performance of admission control and resource allocation for NSLRs of eMBB, URLLC, and MIoT in the 5G core network. SARA introduced a Q-learning based algorithm and DSARA a DQN-based algorithm to select the most profitable NSLRs from a set of NSLRs that arrived in given time windows. These algorithms are model-free, meaning they do not make assumptions about the substrate network as do optimization-based approaches. DSARA achieved up to 3.6%, 7.9%, and 11.7% greater profit than that resulting from the use of SARA, NR, and AAR, respectively. These results corroborate the fact that SARA and DSARA are worth adopting by NSPs for managing networking slicing.

In future work, we will enrich the AC mechanism with the DRL latest enhancements and the RA mechanism with more sophisticated strategies like the proposed in [47]. We also plan to provide AC and adaptive RA for end-to-end slices involving both the 5G radio access and core network.

References

[1] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Comm. Surveys and Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[2] K. Husenovic, I. Bedi, S. Maddens, I. Bozsoki, D. Daryabwite, N. Sundberg, M. Maniewicz *et al.*, "Setting the scene for 5g: Opportunities & challenges," *International Telecommunications Union*, vol. 56, 2018.

[3] 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.

[4] 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 COMMAG*, vol. 55, no. 5, pp. 80–87, May 2017.

[5] J. F. Borin and N. L. S. da Fonseca, "Admission control for wimax networks," *Wireless Comm. and Mobile Computing*, vol. 14, no. 14, pp. 1409–1419, 2014.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

[14] N. L. S. D. Fonseca and R. D. A. Façanha, "The look-ahead-maximize-batch batching policy," *IEEE Trans. Multimedia*, vol. 4, no. 1, pp. 114–120, 2002.

[15] 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.

[16] 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.

[17] 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 Trans. on Networking*, vol. 27, no. 1, pp. 433–446, 2019.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] A. Aijaz, "$hap - slicer$: A radio resource slicing framework for 5g networks with haptic communications," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2285–2296, 2017.

[27] 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.

[28] 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.

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

[30] F. Giust, G. Verin, K. Antevski, J. Chou, Y. Fang, W. Featherstone, F. Fontes, D. Frydman, A. Li, A. Manzalini *et al.*, "Mec deployments in 4g and evolution towards 5g," *ETSI White Paper*, vol. 24, pp. 1–24, 2018.

[31] B. Görkemli, S. Tatlıcıoğlu, A. M. Tekalp, S. Civanlar, and E. Lokman, "Dynamic control plane for sdn at scale," *IEEE JSAC*, vol. 36, no. 12, pp. 2688–2701, 2018.

[32] B. Chatras, U. S. Tsang Kwong, and N. Bihannic, "Nfv enabling network slicing for 5g," in *ICIN*, Paris, France, March 2017, pp. 219–225.

[33] A. Abouaomar, S. Cherkaoui, A. Kobbane, and O. A. Dambri, "A resources representation for resource allocation in fog computing networks," in *IEEE GLOBECOM*, 2019, pp. 1–6.

[34] M. Peng, S. Yan, K. Zhang, and C. Wang, "Fog-computing-based radio access networks: issues and challenges," *IEEE Network*, vol. 30, no. 4, pp. 46–53, 2016.

[35] M. A. Habibi, M. Nasimi, B. Han, and H. D. Schotten, "A comprehensive survey of ran architectures toward 5g mobile communication system," *IEEE Access*, vol. 7, pp. 70 371–70 421, 2019.

[36] G. Brown, "Service-based architecture for 5g core networks," *A Heavy Reading white paper produced for Huawei Technologies Co. Ltd. Online: https://www.huawei.com/en/press-events/news/2017/11/HeavyReading-WhitePaper-5G-Core-Network*, vol. 1, pp. 1–12, 2017.

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

[38] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[39] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *IEEE SSCI*. Athens, Greece: IEEE, 2016, pp. 1–8.

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

[41] 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.

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

[43] 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.

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

[45] A. Hagberg, P. Swart, and D. S Chult, "Exploriponerng network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[46] R. Albert and A.-L. Barabási, "Topology of evolving networks: local events and universality," *Physical rev. letters*, vol. 85, no. 24, p. 5234, 2000.

[47] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. on networking*, vol. 20, no. 1, pp. 206–219, 2011.

William F. Villota Jácome is a Ph.D. student in computer science at the State University of Campinas, Brazil. His research interests include network and service management, 5G, network virtualization, network softwarization, and machine learning.

Oscar M. Caicedo (GS'11–M'15–SM'20) is a Full Professor at the Departamento de Telematica, Universidad del Cauca. His research interests include network management, network virtualization, software-defined networking, machine learning for networking, 5G and beyond networks, and network slicing.

Nelson L. S. da Fonseca (M'88–SM'01) is a Full Professor with the Institute of Computing, State University of Campinas, Brazil. He has authored or coauthored more than 400+ papers and supervised 70+ graduate students. Currently, he serves as Senior Editor for the IEEE Systems Journal, IEEE Communications Magazine and IEEE Communications Surveys and Tutorials. Dr. Fonseca served as Vice President of Technical and Educational Activities, Vice President of Publications and Vice President Member Relations for the IEEE Communications Society (ComSoc).