Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

On using reinforcement learning for network slice admission control in 5G: offline vs. online

Sihem Bakri[1] | Bouziane Brik[1,2] | Adlen Ksentini[1]

[1]EURECOM, 06410 Biot, France
[2]Burgundy University, France

Correspondence
Email: {name.surname}@eurecom.fr
{bouziane.brik}@u-bourgogne.fr

Abstract

Achieving a fair usage of network resources is of vital importance in Slice-ready 5G network. The dilemma of which network slice to accept or to reject is very challenging for the Infrastructure Provider (InfProv). On one hand, InfProv aims to maximize the network resources usage by accepting as many network slices as possible; on the other hand, the network resources are limited, and the network slice requirements regarding Quality of Service (QoS) need to be fulfilled. In this paper, we devise three admission control mechanisms based on Reinforcement Learning, namely Q-Learning, Deep Q-Learning, and Regret Matching, which allow deriving admission control decisions (policy) to be applied by InfProv to admit or reject network slice requests. We evaluated the three algorithms using computer simulation, showing results on each mechanism's performance in terms of maximizing the InfProv revenue and their ability to learn offline or online.

KEYWORDS:
5G; Network slicing; Infrastructure Provider; Slice Admission Control; Reinforcement Learning.

1 | INTRODUCTION

The emerging 5G mobile networks are expected to provide a wide range of novel services that come with different needs and performances, such as ultra-low latency, very high data rates, reliable communications, etc.[1]. In this context, Network Slicing is envisioned to fulfill these requirements. It is based on new technologies, including Software Defined Network (SDN), Network Function Virtualization (NFV), etc.[2 3]. The basic idea of network slicing is to create several virtual instances (slices) of the same network infrastructure, where each instance can provide a specific service. So far, three main types of slice have been defined: eMBB (Enhanced Mobile Broadband), which needs both high data rates and low latency, Ultra-Reliable Low-Latency Communications (uRLLC) covering all services requiring ultra-low latency and high reliability, and Massive Machine-to-Machine Communication (mMTC), requiring wireless connectivity for mass deployment of devices.

Network Slicing enables the apparition of new players in the market: the Infrastructure, or slice, Provider (InfProv), which is the owner of the network infrastru (for example: operators), requesting for a network slice from InfProv to get a target service with specific needs[4 5]. However, as each provider has limited resources[6 7], it is challenging to have an optimal policy to decide which slice requests will be accepted (and/or rejected) by InfProv, and based on which criteria. Indeed, it is difficult to find the optimal policy that, on one hand, increases the revenue of the InfProv and allows an optimal usage of the infrastructure; on the other hand, guarantees the requirement of the admitted network slice in terms of QoS to avoid violating the Service Level Agreement

(SLA). Further, the optimal admission control policy has to consider the long term income of InfProv by accepting the slice that maximizes the revenue while considering their traffic dynamic. For instance, it is difficult to decide between accepting network slices that pay a higher price for a long duration or accepting more network slices for a short duration but pay less money.

In this work, we propose novel slice admission control (SAC) algorithms to be run at the InfProv level aiming at deriving an optimal policy to decide if an arrival network slice request has to be accepted or rejected. The proposed algorithms are based on Reinforcement Learning and seek the optimal policy to increase the InfProv revenue while reducing the penalty to pay due to SLA violation. Three algorithms are introduced: Q-Learning (QL), Deep Q-Learning (DQL), and Regret Matching (RM). Besides deriving the optimal policy, we shed light on the proposed algorithms ability to run offline or online, which is a crucial criterion. Indeed, offline solutions require a training phase before being used, which is sometimes costly; but they generally achieve the best results. While online solutions are trained on the fly using only observable information of the controlled system.

The rest of this paper is organized as follows: Section 2 presents different proposed solutions to address SAC's problem and the efficiency of using RL. Section 3 details the system model that will be analyzed later in section 4. Finally, section 5 presents and discusses the different simulation results.

2 | RELATED WORK

Different studies have addressed the problem of network slice admission control by exploring several techniques. In [16], the authors proposed an algorithm aims at maximizing the profit of InfProv, by admitting more slice requests than the overall capacity of the system, similarly to the concept of flight overbooking. They formulated the orchestration issue as a stochastic management problem that performs jointly resource allocation and admission control in all technological domains composing a mobile system. They then proposed solving the formulated problem using two algorithms: an optimal solution using Benders decomposition and a sub-optimal heuristic that accelerates the decision-making process. The authors of [17] proposed an admission control mechanism for network slicing that maximizes InfProv revenues while meeting services' latency requirements. They design a SAC policy using bid selection before studying the best strategy under different constraints (e.g., available resources, InfProv's strategy and requested traffic, etc.).

The authors of [5] proposed to maximize the revenues of InfProv when performing admission control by modeling the problem via Markov Decision Process (MDP). They proposed two methods to solve the MDP: value iteration and Q-learning. This work is based on offline solutions, whereas our work will present dynamic and online solutions. Moreover, the algorithms proposed in this work are not appropriate for a complex environment, while in our work we will test algorithms tailored to more complex environments, citing deep QL. Regarding modeling, this work aimed to maximize InfProv revenues by accepting the most expensive slices' requests first, which may lead to a lack of fairness between the network slices requests. Moreover, this work ignores the notion of priority between network slices, which, in contrast, will be considered in our contribution. Finally, it did not consider the needed physical resources in both UL and DL, which we will introduce in this work. In [18], the author proposed a new dynamic SAC model, using reinforcement learning. In this model, the InfProv generates revenues when accepting a slice request; and based on the requested slice priority, pays a penalty when rejecting it. The designed model aims at reducing the penalty fee by minimizing the rejection of expensive requests, hence maximizing InfProv revenues. This work also confirms the efficiency of using RL to maximize InfProv revenues. However, in our work, we consider more parameters to tackle the SAC issue, including the notion of hosting time needed by each slice and the requested physical resources by slice tenants at both UL and DL directions.

3 | SYSTEM MODEL

In this section, we describe the considered system model, which is illustrated in Figure 1. It comprises the following actors:

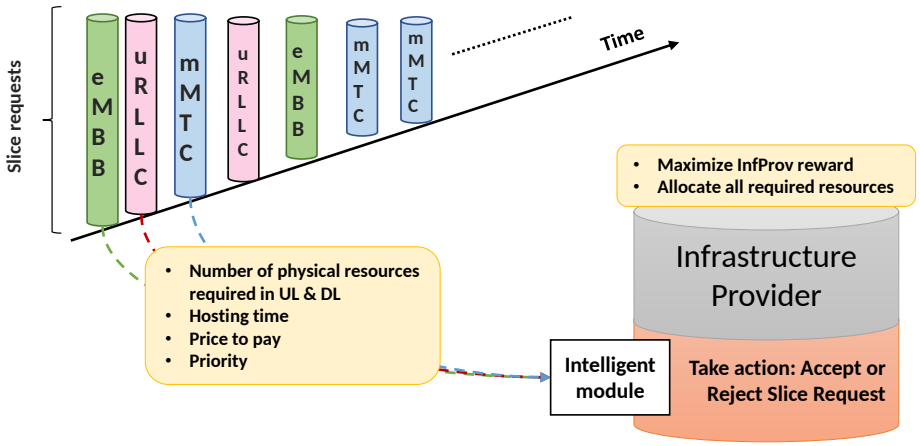


FIGURE 1 System model

Network players: In our system model, we consider two main players: (i) the Infrastructure Provider (InfProv), which is the owner of the network infrastructure, and in charge of instantiating network slices for tenants by providing the required resources; (ii) the tenants that request the instantiation of network slice from the infrastructure provider to offer services for their clients.

Network slice model: In our model, each network slice is characterized by five main criteria:

1. The physical resources needed to satisfy the requirement of a network slice, in UL and DL, noted $Nres^{i}_{req}(UL)$ and $Nres^{i}_{req}(DL)$, respectively. Where i is the slice type, which can be either eMBB, uRLLC, or mMTC. We define the needed resources by each slice type as follows:

   - $Nres^{eMBB}_{req}(DL) >> Nres^{eMBB}_{req}(UL)$
   - $Nres^{uRLLC}_{req}(DL) >> Nres^{uRLLC}_{req}(UL)$ or $Nres^{uRLLC}_{req}(DL) == Nres^{uRLLC}_{req}(UL)$ or $Nres^{uRLLC}_{req}(DL) << Nres^{uRLLC}_{req}(UL)$
   - $Nres^{mMTC}_{req}(DL) << Nres^{mMTC}_{req}(UL)$

   We justify these assumptions as most eMBB traffic has dominated DL traffic (ex. high definition video streaming), while mMTC traffic has dominated UL traffic (ex. IoT traffic). The case of uRLLC is different, as all the types of traffic may exist and depend on the service.

2. The hosting time of each network slice type $Htime$: Each slice, if admitted, will use the InfProv's resources for a given duration.

3. The $priority$ of the slice: It depends on the application running on the corresponding slice.

4. The price P^{i}_{req} that a slice tenant pays to InfProv for the used resources. The tenant has to pay the resources per time unit for the H_{time} duration. Here, req refers to a slice tenant, and i refers to the slice type.

Infrastructure provider model: The InfProv entity is characterized by its capacity in terms of available resources at time t (C_t). It represents the total amount of available resources that may be allocated to a new network slice. It is worth noting that C_t is

updated when a network slice is admitted or leaves. In other words, when a new network slice is accepted, the needed resources will be allocated and dedicated to it; when a network slice ends, its associated resources will be automatically released. Thus, at each time instant t, the available resources at InfProv is performed as follows: $C_t = C_{total} - C_{allocated} + C_{released}$. Note that, two formulas are used, one for UL and one for DL, as the resources are separated. C_{total} is the total number of resources available in the InfProv in UL or DL. $C_{allocated}$ and $C_{released}$ are respectively the number of allocated and released resources at time t in UL or DL.

The proposed SAC model is applied only for the RAN resources, composed of DL and UL. We argue this assumption by the fact that RAN is considered as the bottleneck of the system, while other network slice's required resources (such as computing) are always available, and no reservation is needed.

In summary, the InfProv is characterized by its resources capacity C_{total}, while a network slice is identified by: N^i_{res}, H_{time}, $Priority$, and P^i_{req}.

4 | SYSTEM ANALYSIS

As stated earlier, we seek an optimal admission control policy that aims at finding a trade-off between fulfilling the network slice resource request (UL and DL); while maximizing InfProv revenues. First, we propose to model the SAC using Markov Decision Process (MDP)[12]. Since exactly solving the MDP is very challenging due to the difficulties in modeling the traffic dynamics, we apply reinforcement learning to derive the optimal policy and to find the earlier-mentioned trade-off. For that, we will use different Reinforcement learning models, namely QL, DQL, and RM, to predict the optimal action to apply when a new demand of a network slice arrives at the system (i.e., accept or reject an arrival slice request).

4.1 | Markov Decision Process model

A Markov Decision Process is composed of 4-tuples $M = (S; A; T; R)$, where S is the set of states, A is the set of actions, T is the transition probability from state s at time t to state s' at time $t + 1$ when taking an action a, and R is the reward obtained by performing the action a, which leads to move from the state s to s'.

For our system, we assume that a state $s = (n, m, l, b)$ is composed of four information where:

- n is the number of accepted eMBB slices;
- m is the number of accepted uRLLC slices;
- l is the number of accepted mMTC slices;
- b is a value that can be equal to 1,2 or 3 to indicate the slice type, eMBB, uRLLC, or mMTC, respectively, of the last received request.

At receiving a new network slice request, InfProv, via an agent, observes the state of the system and takes action a: either to accept or reject the request. The action set is as follows:

$$a = \begin{cases} 1 & \text{if new arrival slice request is accepted} \\ 0 & \text{if new arrival slice request is rejected} \end{cases} \tag{1}$$

The different transitions of the system occur when a new network slice arrives, and a decision is needed, and when a network slice leaves the system. If the system is in a state $s = (n, m, l, b)$ and a new slice arrives, a decision needs to be taken (i.e., accept or reject), leading that the system transits to one of the following states:

- $(n + 1, m, l, 1)$ if a slice of eMBB is accepted;
- $(n, m + 1, l, 2)$ if a slice of uRLLC is accepted;

- $(n, m, l+1, 3)$ if a slice of mMTC is accepted;
- $(n, m, l, 1)$ if a slice of eMBB is rejected;
- $(n, m, l, 2)$ if a slice of uRLLC is rejected;
- $(n, m, l, 3)$ if a slice of mMTC is rejected.

If a network slice leaves, then the system moves to one of the following states without taking any action:

- $(n-1, m, l, 1)$ if a slice of eMBB has left;
- $(n, m-1, l, 2)$ if a slice of uRLLC has left;
- $(n, m, l-1, 3)$ if a slice of mMTC has left.

As mentioned above, each network slice is described by $<Nres^i_{req}(UL), Nres^i_{req}(DL), H_{time}, priority$ and $P^i_{req}>$. We assume that the price P^i_{req} to pay by each slice tenant, by time unit, is proportional to the slice $priority$. Hence, we propose to model the estimated reward that InfProv expects to receive from each accepted network slice as follows:

$$R_{inf} = sign[C_t - Nres^i_{req}] \times P^i_{req} \times Htime^i_{req} \tag{2}$$

With:

$$sign(C_t - Nres^i_{req}) = \begin{cases} 1 & C_t \geq Nres^i_{req} \\ -1 & C_t < Nres^i_{req} \end{cases} \tag{3}$$

In equation 2, we multiply the P^i_{req} by H_{time}, since each accepted slice tenant pays a P^i_{req} according to the slice priority by a time unit. Hence, the total price that a tenant of an accepted slice will pay depends on his priority and the requested hosting time. Besides, we have added the sign in this equation to ensure that there are enough resources in InfProv to support the number of required slice resources.

It is worth noting that in this work, for each accepted network slice, the InfProv should be able to provide the needed resources for both UL and DL. Otherwise, the InfProv will pay a penalty, if it accepts a slice request without having enough resources to cover the slice resource requirements. To this end, C_t should be always higher than $Nres^i_{req}$ in UL and DL. Therefore, the reward defined in 2 for both UL and DL will be calculated as follows:

$$R_{inf} = [sign(C_{InfDL} - Nres^i_{reqDL}) \wedge sign(C_{InfUL} - Nres^i_{reqUL})] \times P^i_{req} \times Htime^i_{req} \tag{4}$$

We also note that the R_{inf} is null if the slice request is rejected, as neither penalty nor reward can be applied. Having defined the MDP model, we need to find the optimal policy that maximizes the long term total reward for InfProv. The optimal policy corresponds to the action to take for each state s aiming at maximizing the long-term total reward. Since the MDP is hard to solve using techniques like Value iteration or Policy iteration as the traffic model is hard to model, we describe in the next section how to find this policy using Reinforcement Learning, through three models QL, DQN, and RM.

4.2 | Admission control using QL

Q-learning is an offline reinforcement learning algorithm that generates an optimal policy to maximize the expected total reward for any finite MDP, i.e., the state and action spaces may be finite, which is our model's case. This policy is based on the Q-learning function, which is designed to seek the best action in each state to maximize the long-term total reward.

The QL method consists first of calculating, for each possible action in each state, a value named Q-value. Then the QL method stores these Q-values in a table, namely the Q-table. This step is called the exploration of the unknown environment. It is worth noting that the Q-table is initiated to zero and updated with the new Q-values obtained after each episode.

The agent performs in a state s_t, one of the two actions: accept or reject a new slice request for the epoch t, and it observes the state transitions s_{t+1}, and rewards r. Hence, it updates the Q-value using the weighted average of the previous and the new Q-value, as shown in the following equation:

$$Q_{new}\left(s_t, a_t\right) \leftarrow (1-\alpha)\, Q\left(s_t, a_t\right) + \alpha\left(r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a)\right) \tag{5}$$

with:

- $Q(s_t, a_t)$ is the old Q value;
- $Q_{new}(s_t, a_t)$ the new value obtained after updating the old one;
- α is the learning rate that controls how fast the new estimation adapts to the random changes imposed by the environment.
- γ is the discount factor that notifies the importance of future rewards;
- r_t is a reward received from action a_t;
- $maxQ(s_{t+1}; a)$ is the estimation of the optimal future action.

After several episodes, the Q-table converges and becomes the reference table for the agent (.i.e., the entity that takes decisions) to select the best action based on the Q-value. However, one of the QL method's weaknesses is the convergence time, i.e., the time needed by the agent to explore all the states to learn the best action to take in the future. Indeed, it depends on the state space; if the latter is large, the time to converge is high, which may be problematic if QL is used without offline training.

4.3 | Admission control using DQL

Q learning is based on a Q-table to store the learned results for each state and action. Consequently, if the state space is large, the table size explodes, leading to an increase in the training time as the agent has to take more time to explore all the states. To this end, DQL uses deep learning to represent the Q-values, where each state passes through several hidden layers of a neural network to get the Q-values. Then, DQL calculates: (i) the loss function that represents the mean squared error (MSE) as shown equation 6 of the predicted Q-value (Q_{pred}), and (ii) the target Q-value (Q_{target}) (see equation 7) that represents the maximum possible value for the next state.

$$MSE(\theta_i) = [Q_{target} - Q(s_t, a, \theta_i]^2 \tag{6}$$

$$Q_{target} = E[r + \gamma maxQ(s_t, a', \theta_{i-1})] \tag{7}$$

With θ is the weight.

Using the same θ weights in (6), the values Q_{target} and Q_{pred} move at the same time. For this purpose, DQL uses two neural networks, one for Q_{pred} and the other one for Q_{target}. Algorithm 1 presents the different steps of the DQL algorithm.

Note that we have considered the states and actions as defined in the MDP.

4.4 | Admission control using Regret Matching

Regret Matching (RM) is an online learning algorithm similar to Reinforcement Learning. Its agent (player or user) looks for the right action based on the regrets of the previous actions. The main principle consists of minimizing the regrets of its decisions at each step of the system. To do so, the agent relies on past behavior of taken actions to guide its future decisions by favoring the actions that it regrets not to have chosen before.

The strategy of this method is to adjust the agent's policy by distributing probabilities on all actions proportionally to the regrets of not having played other actions.The regret is defined as follows: if a is the action chosen by the agent at time T, thus for any other action $a \neq$a*, the regret of choosing the action a but not another action a^* up to time T is obtained as shown equation 8 [19].

$$Reg_T(a, a^*) = \frac{1}{T}\sum_{t=1}^{T} r_t(a) - \frac{1}{T}\sum_{t=1}^{T} r_t^i(a^*) \tag{8}$$

Algorithm 1 Deep Q Learning algorithm

Ensure: $s \leftarrow s_{t+1}$
Initialize: replay memory D to capacity N
Initialize: $Q_0(s, a)$ with random weights,

for $episode \leftarrow$ 1: M **do**

Initialize state s_t

for $t \leftarrow$ 1: T **do**

- With probability ϵ select:
  - a random action a_t
  - $a_t = max_a$ Q(s_t, a, θ) (otherwise)
- Execute action a_t and observe reward r_t and state s_{t+1}
- Store transition (s_t,a_t,r_t,s_{t+1}) in D
- Set $s_{t+1} = s_t$
- Sample random minibatch of transitions (s_t,a_t,r_t,s_{t+1}) from D
- Set $Q_{target\,j} = \begin{cases} \mathrm{r}_j \text{ for terminal } s_{t+1} \\ \mathrm{r}_j + \gamma * max_{a'} * Q(s_{t+1}, \mathrm{a}', \theta_{i-1} \text{ for non terminal } s_{t+1} \end{cases}$
- Perform a gradient descent step:
- $MSE(\theta_i) = [Q_{target} - Q(s_t, a_j, \theta_i]^2$

end for
end for

Algorithm 2 Regret matching algorithm

Ensure: $s \leftarrow s_{t+1}$
Initialize: action a_1,

for $t \leftarrow$ 1: T **do**

- Take action a_t
- Receive reward $r_t(a)$ and $r_t(a^*)$
- Update the regret of choosing action a_t and not a_t^* is: $Reg_T(a, a^*) = \frac{1}{T} \sum_{t=1}^{T} r_t(a)$ - $\frac{1}{T} \sum_{t=1}^{T} r_t^i(a^*)$
- The probability of choosing action a_t in the next step is: $P_{T+1}(a) = \begin{cases} \frac{[Reg_T(a,a^*)]^+}{\sum_{a=0,1}[Reg_T(a,a^*)]^+} & \text{if a} \neq \mathrm{a}_T \\ 1 - \sum_{a' \neq a_T} P_{T+1}(a') & \text{if a} = a_T \end{cases}$
- Select action corresponding to maximum probability
- Update action a_{t+1}

end for

with: $r_t(a)$ is the reward obtained at time T, by choosing the action a. At each step, the agent chooses an action a_T between two actions (accept or reject) considered in this study (see equation 1). The probability P_{T+1} that the agent will choose action a in the next step defined by next time T+1, is defined as follows:

$$P_{T+1}(a) = \begin{cases} \frac{[Reg_T(a,a^*)]^+}{\sum_{a=0,1}[Reg_T(a,a^*)]^+} & \text{if a} \neq \text{a}_T \\ 1 - \sum_{a' \neq a_T} P_{T+1}(a') & \text{if a= } a_T \end{cases} \tag{9}$$

With: $Reg_T(a, a^*)]^+ = \max [Reg_T(a, a^*),0]$ presents the non negative part of the regret $Reg_T(a, a^*)$.
The RM algorithm is illustrated in Algorithm 2.
It is worth noting that the RM is a fully online solution; the policy should be initiated before that the algorithm starts adapting itself. Therefore, we consider RM with two initial policies: accept and reject. The first one starts by accepting the network slice requests, while the second one starts by rejecting the requests.

5 | PERFORMANCE EVALUATION

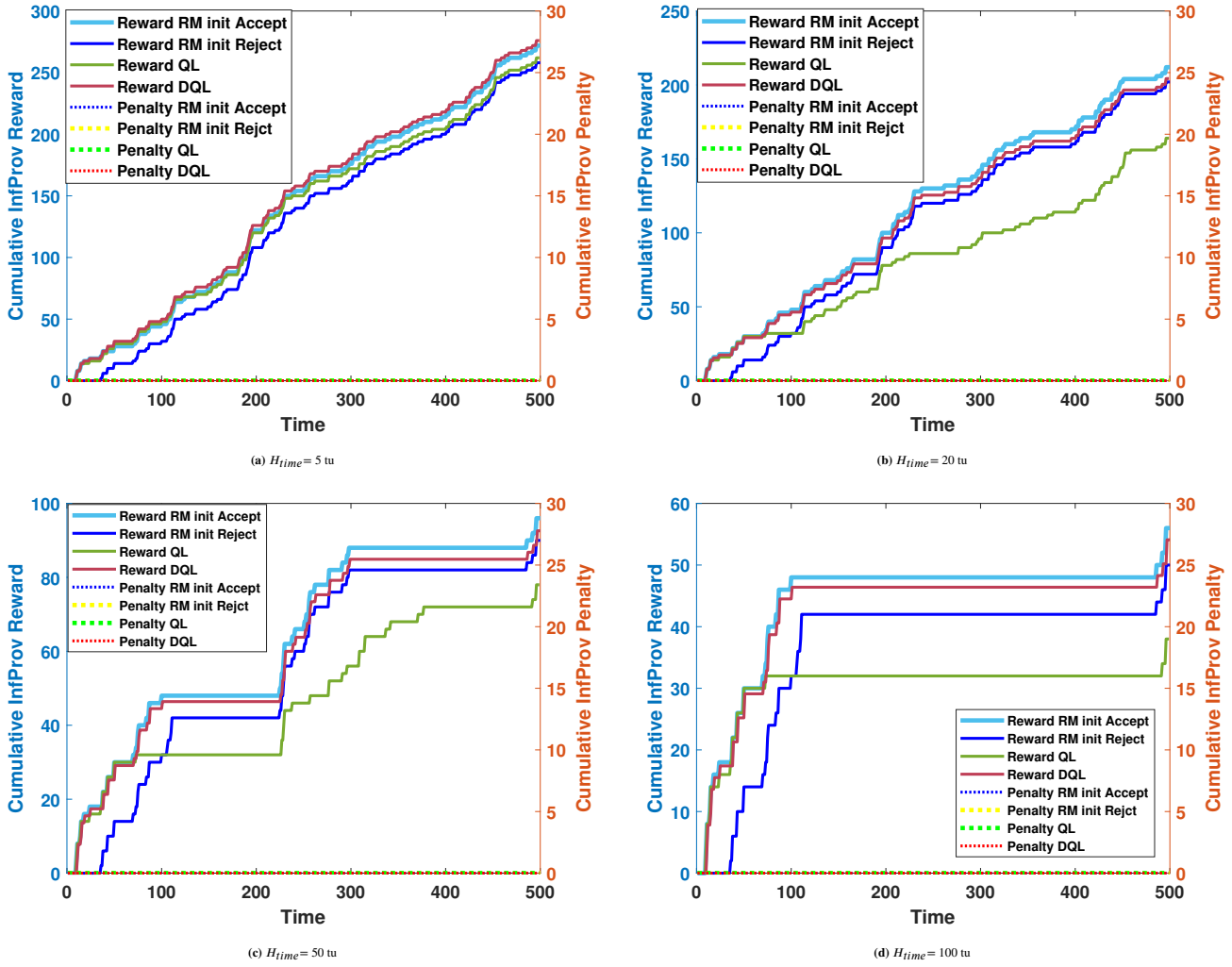
In this section, we present the simulation results of the slice admission control problem by comparing the three methods' performances. It is worth noting that the RM considered here is initialized once by accepting the first received requests (noted as RM with accept policy), and once by rejecting the first received requests (noted as RM with reject policy).

TABLE 1 Number of resources: (i) available in InfProv, (ii) requested by each slice in UL and DL

[UL, DL] InfProv	[100, 100]
[uRLLC, mMTC, mMBB] UL slices	[5,9,5]
[uRLLC, mMTC, mMBB] DL slices	[5,5,10]

We assume that InfProv receives requests to create slices following a Poisson process with two different arrival rates as follows: (i) rate=2 per time unit (tu) corresponding to a low arrival rate (i.e., the slice requests arrive rarely), and (ii) rate=10 per tu corresponding to a frequent slice request arrival. Besides, we assume that each slice request stays hosted in InfProv for a H_{time} period. To show the impact of H_{time}, we use four values as follows: (i) short period where H_{time}= 5 *tu*, (ii) medium period where $H_{time} = 20$ *tu*, (iii) large period where $H_{time} = 50$ *tu*, and (vi) very large period where $H_{time} = 100$ *tu*. We consider that the number of resources requested by each slice in UL and DL, and the number of resources available in the InfProv are different, and their values are presented in tab Table 1. Note that the three algorithms considered in our work (i.e., RM, QL, and DQL) apply the same reward formula, which is based on the price and hosting time of each accepted slice request.
Regarding slices' priority, we assume obviously that uRLLC slices have the highest priority since it hosts application requiring critical latency and reliability, while eMBB and mMTC slices have the same priority. The price of running the uRLLC slice type is four times higher than the price to pay for running the mMTC and eMBB slice types. The latter slices (i.e., mMMTC and eMBB) have the same price. In other words, we use the price to pay as a way to enforce priority among the slice types.
Regarding the training session, it is worth noting that the offline version of both QL and DQL algorithms requires a training phase, in which the agent explores the environment to learn how to achieve the optimal and most rewarding actions. However, our results are generated on the test phase, i.e., we aim to evaluate our learning models on new data.

Figure 2 illustrates the cumulative reward as well as the cumulative penalty obtained using the proposed algorithms (RM initialized with accept policy, RM initialized with reject policy, QL, and DQL) when the arrival rate is 2 per *tu*, and for four



(a) H_{time}= 5 tu

(b) H_{time}= 20 tu

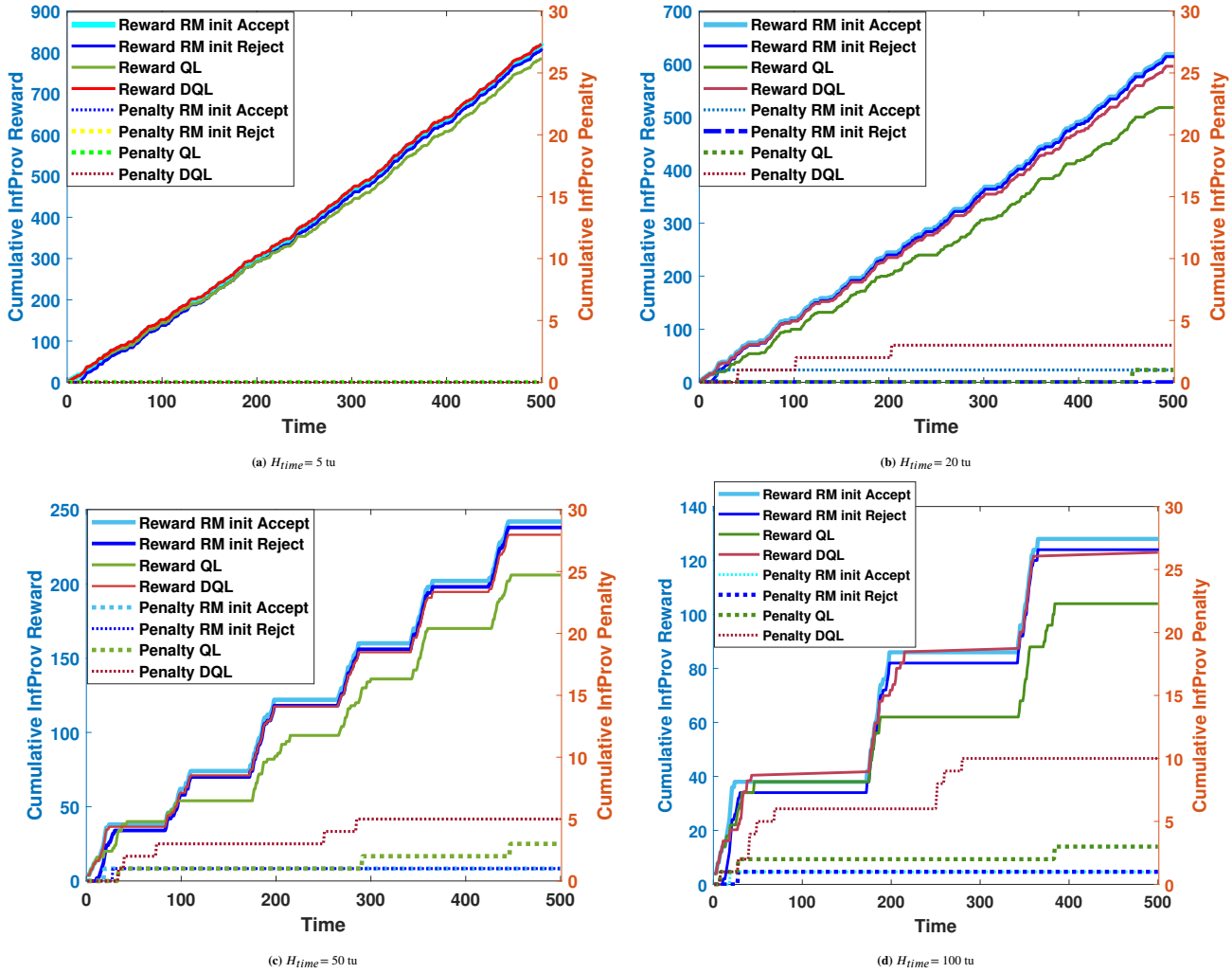
(c) H_{time}= 50 tu

(d) H_{time}= 100 tu

FIGURE 2 $Inf\,Prov$ reward and penalty vs. $Time$ for slice arrival request rate= 2

values of the Holding time (H_{time}). The same metrics are shown in Figure 3, but for an arrival rate of 10 per *tu*. We recall that cumulative reward is obtained by the infrastructure provider when accepting slices, and (ii) the penalty is incurred when a slice is accepted but InfProv has not sufficient resources either in DL or UL or in both directions to satisfy its requirements. For the cumulative reward, we notice that, for both arrival rates and in the four proposed solutions, when the H_{time} increases, the cumulative reward decreases. We argue this by the fact that the resources are not released quickly when H_{time} is high; hence the InfProv rejects new requests during this H_{time} period.

Besides, penalties (accepting a slice without having a resource) occur when the InfProv resources are saturated, and the SAC keeps accepting arrival slices. We remark that penalties are higher when both the arrival rate and the holding time are high, which is evident as the resources are quickly saturated since accepted slices stay longer in the system. Further, we note that most of the algorithms require some time to detect that the resources are saturated and keep accepting requests until the reward starts to be negative. Consequently, they change the policy to reject. The time to detect that the resources are saturated is a criterion to understand which algorithm performs well, and hence learned the system's behavior. In this case, we remark that RM obtains the best performances with accept policy, followed by DQL. QL achieves the worst performance. We argue this by the fact that RM, thanks to its regret formula, detects quickly that the reward starts to be negative, and adapts the policy accordingly. Further, DQL with the neuronal network can learn and predict better when to change the policy, compared to QL, where the Q-tables cannot predict when to update.



(a) H_{time}= 5 tu

(b) H_{time}= 20 tu

(c) H_{time}= 50 tu

(d) H_{time}= 100 tu

FIGURE 3 *Inf Prov* reward and penalty vs. *Time* for slice arrival request rate=10

On the other hand, when the arrival rate and the holding time are low, the probability of having penalties is very small or even null as there are always available resources to accept new slices Figure 2.
The results of Figure 2 and Figure 3 also show that in terms of rewards, the QL algorithm is the worst algorithm of all tested algorithms regardless of the arrival rate of each type of slices, by achieving the lowest cumulative reward. This means that it does not learn well when the policy should change from accepting to rejecting, or the contrary. Indeed, QL derived policies that favor rejecting slice requests.

Figure 4 presents the percentage of rejected requests according to H_{time} when using the proposed algorithms: RM with accept policy, QL and DQL, and request arrivals rate= 10 corresponding to a frequent slice request arrival. We notice that increasing H_{time} leads to an increase in the percentage of rejection for all the algorithms. This is obvious as high values of H_{time} mean that admitted slices will stay longer in the network, and hence low resources are available to accept new slices (i.e., increase the reject rate). Moreover, we noticed that QL rejects more requests than RM and DQL for all the H_{time}, which confirms the low cumulated penalty and reward of QL shown in the two precedent figures.

In Figure 5 we present the percentage of accepted slices according to their type and for different H_{time}, when using RM algorithms with accept policy, QL and DQL. Here our objective is to verify whether the proposed algorithms satisfy the slice priority

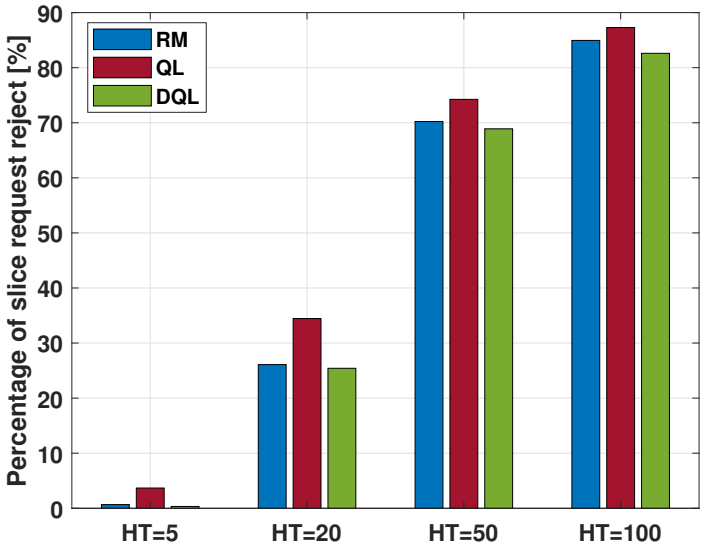
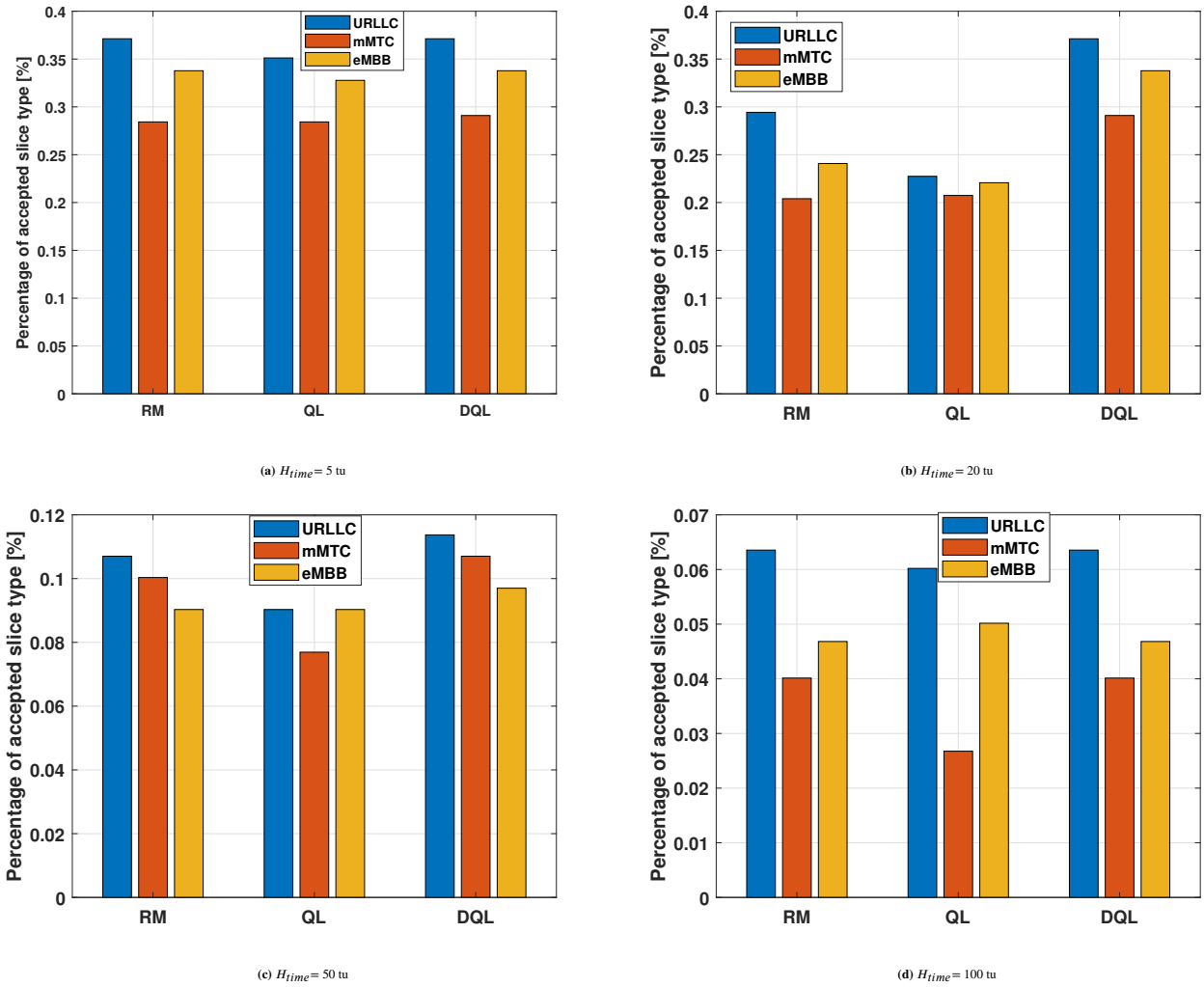


FIGURE 4 Percentage of slice request reject vs H_{time} for slice arrival request rate= 10

condition, i.e., the ability to give priority to uRLLC slices compared to the other slice types. We can see that all algorithms favor the uRLLC slice over the other slice types. Further, we see that DQL and RM show the highest percentage of accepted uRLLC slices compared to QL. In other words, these results validate our reward function and the fact of using the price to pay as a way to enforce priority among the three types of slices.

One of the biggest challenges when using Reinforcement Learning to solve SAC's problem is whether online or offline learning is better? And what is the time of convergence? So far, we have seen that RM, a fully online algorithm, works well for SAC's problem, while DQL and QL need to be trained before being used. Regarding DQL we wanted to understand if an online version could make sense to address the SAC problem efficiently. To this aim, we draw in Figure 6 a comparison between the offline DQL (when we first perform the training phase and then the tests) and the online DQL in terms of average reward. The online training phase of DQL varies between 0 and 1000 episodes. Each episode represents one training epoch during which the system receives 100 arrival slice requests. The maximum number of episodes depends on the convergence of the online learning model. We have increased the number of episodes and calculated the corresponding average reward. We have stopped when the learning model starts to give a constant reward (between 500 and 1000). Note that the average reward is an average value of the reward obtained for the four considered H_{time}. We clearly observe that the online DQL needs time i.e., more episodes to converge (improve the learning) and start achieving the same performance as offline DQL (around 400 episodes). This means that during this period, i.e., before converging, the DQL performances are awful and can seriously affect the business of InfProv.

All the results confirm that one of SAC's best policy is to accept whenever the resources are available to maximize the InfProv profile. In this context, RM with accept policy achieves the best performances by reducing the penalty and increasing the reward. DQL and QL could be a good candidate, but there is a need to well tune the learning steps in order to anticipate when the policy has to change. Indeed, RM uses a simple and efficient formula to understand the need to change policy, while DQL and QL need to learn this. However, in a more complex system, where a high number of actions are available, things may change as RM can hardly, by using a simple formula, capture the behavior of the system. In contrast, DQL can be a powerful solution. But, in the case of a SAC with only two available actions, RM with the accept policy is the best alternative for InfProv.



(a) H_{time}= 5 tu

(b) H_{time}= 20 tu

(c) H_{time}= 50 tu

(d) H_{time}= 100 tu

FIGURE 5 Percentage of accepted slice type for slice arrival request rate=10

6 | CONCLUSION

In this paper, we have studied the challenge of network slice admission control for future 5G networks. We have proposed algorithms based on Reinforcement Learning algorithms (QL, DQL), and an online algorithm, which is the Regret Matching to solve this problem. We first modeled the problem as an MDP and described how the proposed algorithms could derive a policy to be used by the agent to maximize the InfProv revenue while avoiding violating network slice requirements. We have modeled the reward as a function of the network slice price to pay, and the penalty to pay if the slice requirement is not satisfied. Through extensive simulation, we showed that all algorithms could derive a good policy, but RM achieves the best performances, as the SAC problem is not complex and the actions state is limited.

ACKNOWLEDGEMENT

This work has been partially supported by the European Union's H2020 MonB5G (grant no. 871780) project.

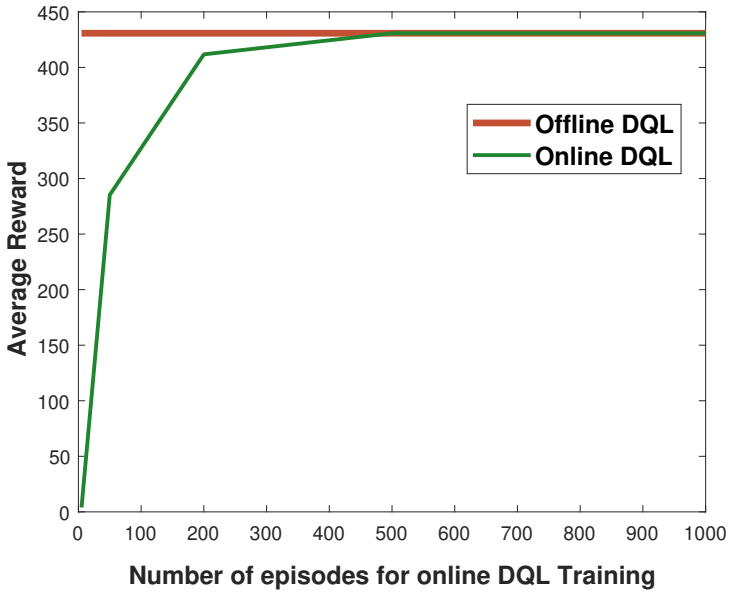


FIGURE 6 Offline and Online DQL Average Reward of H_{time}= (5, 20, 50, 100) for slice arrival request rate= 10

DATA AVAILABILITY STATEMENT

Research data are not shared.

References

1. M. Agiwal, A. Roy and N. Saxena, "Next Generation 5G Wireless Networks: A Comprehensive Survey," in IEEE Communications Surveys & Tutorials, vol. 18, no. 3, pp. 1617-1655, third quarter 2016.

2. A. Ksentini and N. Nikaein, “Towards enforcing Network Slicing on RAN: Flexibility and Resources abstraction,” in IEEE Communications Magazine, Jun. 2017.

3. A. Ksentini, P. A. Frangoudis, A. PC and N. Nikaein, “Providing low latency guarantees for slicing-ready 5G systems via two-level MAC scheduling,” in IEEE Network, Nov.2018.

4. Study on Management and Orchestration of Network Slicing for Next Generation Network, document TR 28.801, Release 15, 3GPP, Jan. 2018.

5. D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing" in Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM), Atlanta, GA, USA, May 2017, pp.1–9.

6. S. Bakri, P. A. Frangoudis and A. Ksentini, “Dynamic slicing of RAN resources for heterogeneous coexisting 5G services,” IEEE GLOBECOM, 2019

7. A. Anand, G. De Veciana and S. Shakkottai, "Joint Scheduling of uRLLC and eMBB Traffic in 5G Wireless Networks", in *Proc. IEEE INFOCOM*, 2018.

8. W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), Jilin, 2011

9. A.Nassar and Y.Yilmaz, "Reinforcement Learning for Adaptive Resource Allocation in Fog RAN for IoT With Heterogeneous Latency Requirements", IEEE Access (Volume: 7), Pages: 128014- 128025, September 2019.

10. Ranran Xi: "Real time resource slicing for 5G RAN via deep reinforcement learning", 25th International Conference on Parallel and Distributed Systems (ICPADS), 2019.

11. M. Yan, G. Feng, J. Zhou, Y. Sun and Y. Liang, "Intelligent Resource Scheduling for 5G Radio Access Network Slicing"; IEEE Transactions on Vehicular Technology, Pages: 7691 - 7703, year 2019.

12. Y. Chen, Y. Gao, C. Jiang and K. J. R. Liu, "Game theoretic Markov decision processes for optimal decision making in social systems," 2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Atlanta, GA, 2014.

13. D. Pandey and P. Pandey, "Approximate Q-Learning: An Introduction," 2010 Second International Conference on Machine Learning and Computing, Bangalore, 2010

14. A. Jeerige, D. Bein and A. Verma, "Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing," 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2019.

15. Q. Yu, J. Chen, Y. Sun, Y. Fan and X. Shen, "Regret Matching Based Channel Assignment for Wireless Sensor Networks," 2010 IEEE International Conference on Communications, Cape Town, 2010.

16. J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, X. Costa-Perez, "Overbooking Network Slices through Yield-driven End-to-End Orchestration", in *Proc. ACM CoNEXT*, 2018.

17. M. Vincenzi, E. Lopez-Aguilera and E. Garcia-Villegas, "Maximizing Infrastructure Providers' Revenue Through Network Slicing in 5G", IEEE Access (Volume: 7), P: 128283- 128297, Sept 2019.

18. M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska and P. Monti, "Reinforcement Learning for Slicing in a 5G Flexible RAN", IEE Journal of Lightwave Technology, Volume: 37, Issue: 20, Oct 2019

19. Nguyen DD, Rajagopalan A, Lim CC. Online versus offline reinforcement learning for false target control against known threat. In: Chen Z, Mendes A, Yan Y, Chen S, editors. Intelligent Robotics and Applications. ICIRA 2018. Lecture Notes in Computer Science, vol 10985. Springer.

How to cite this article: S. Bakri, B. Brik, and A. Ksentini, (2020), On using reinforcement learning for network slice admission control in 5G: offline vs. online, *International journal of communication systems*, *2020;00:1–14*.