Approximate Q-Learning-based (AQL) Network Slicing in Mobile Edge-Cloud for Delay-sensitive Services

Mohsen Khani
Islamic Azad University
Shahram Jamali (jamali@uma.ac.ir)
University of Mohaghegh Ardabil
Mohammad Karim Sohrabi
Islamic Azad University

Research Article

Keywords: Slice acceptance control, 5G, Approximate reinforcement learning, Network slicing, Mobile Edge-Cloud

Posted Date: March 8th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2645843/v1

Abstract

Network slicing (NS) technology promises to provide a variety of services based on diverse latency-sensitive over shared infrastructure in Mobile Edge-Cloud (MEC) by creating customized slices for each application. However, to process users' dynamic slice requests, the infrastructure provider (InP) must be the online slice acceptance check and scaled if required. Based on a business model, network revenue is dependent on the acceptance of slices and the provision of infrastructure for them. If an InP does not provide more resources for an active slice, the network is penalized and its revenues are degraded. A proper solution to the aforementioned problem is to use reinforcement learning methods. But most of these methods have challenges in continuous spaces. This work presents a reinforcement learning-based method called approximate Q-learning (AQL) to intelligently slice acceptance control (SAC) to maximize utility in MEC for latency-sensitive services. The core idea of AQL is based on Q-learning, so we have developed some of its functions to adapt to a large area of spaces and actions. We have evaluated the performance of AQL in terms of coverage, cumulative rewards, resource utilization, and revenue. The results show the proposed approach has an acceptable performance.

1. Introduction

In recent years, network slicing (NS) has been introduced as a promising novelty to provide scalability and flexibility for diverse 5G applications that attach to manifold technical, service, and operation requirements [1]. This technology enabled infrastructure providers (InP) to create many isolated logic networks (slices) and share them among applications based on the type of service. The services of 5G are divided into three groups. (1) Ultra-Reliable and Low Latency Communication (URLLC) (i.e. Remote surgery, automation driving, etc), (2) Enhanced Mobile Brobroadband (MBB) such as high-definition video streaming, and (3) massive Machine Type Communication (mMTC) [2]. So, in order to serve these services in real-time constraints, Mobile Edge-Cloud (MEC) computing is raised that empowers devices with computational resources closer to the users [3, 4]. These edge devices, also known as fog nodes, are capable of hosting virtualized network functions (VNFs), providing the computing capability for small cells to bring cloud functionalities to the edge of the network [5, 6]. Since the fog nodes have limited capacity, they can handle slices that are more sensitive to delay. Also, these nodes are connected with the central office (CO) through a fronthaul link, and slices that can tolerate some delay are referred to there [7].

Slices are created by InP to gain revenue based on specific requirements (i.e., reliability, latency, capability, etc.)[8]. Also, according to the variety of requirements, the InP should dynamically scale up/down the provisioned slices. However, If the slices cannot scale up when they are needed, the service will be degraded and the InP must pay a penalty corresponding priority of the application [8]. Therefore, in order to maximize revenue, the InP faces two basic challenges. First, accept as many slice requests (S-REQ) as possible, and second, avoid the imposition of penalty. So, we need a slice acceptance strategy that can evaluate the acceptance performance of each slice and factor it into future decisions. So, it is inevitable to use reinforcement learning (RL) methods, more specifically Q-learning. It is a branch of ML that takes

steps to learn the environment without needing a model (model-free) and having prior knowledge and if it can get enough training makes the best decision in any state [9, 10]. Because the space of the 5G environment is large (continuous) and full of random parameters, the use of Q-learning poses a serious challenge. In this paper, we apply to modify some parameters of Q-learning algorithm to adapt to the continuous state space in order to make intelligent slice acceptance control (SAC) in mobile MEC and propose approximate Q-learning (AQL). The proposed method decides which S-REQ should be accepted by learning how to maximize revenue while minimizing the penalty (rejecting requests that degrade services). The scenario investigated in this work is a MEC, where applications from users are virtualized over the same resources. Two types of applications are considered: latency-sensitive (LS) and latency-tolerable (LT). LS has high revenue/penalty and LT is non-strict latency and low penalty/revenue. The performance of the proposed AQL is compared against two RL methods, and the results show the proposed approach has an acceptable performance. In short, the primary contributions of this work can be summarized as follows:

- We divide the applications into two classes according to delay sensitivity and their revenue.
- Moreover, we design an RL model to discover an optimal SAC policy to maximize InP revenue in the MEC environment.
- In addition, we seizethe complexity of the SAC in MEC by including time-varying resource situations, various service requirements, and frequency of requests from users' information in the state of the system. So, endorse model-free solutions for SAC.
- Also, we improve the performance of Q-learning by modifying some effective parameters.
- Finally, we propose the optimized approach to design the SAC system

The rest of this work is organized as follows. We consider the related works in Section 2. The system model is described in Section 3. In Section 4, we provide Q-learning, present how to overcome its weaknesses through AQL, and explain applying the proposed method to our model. The simulation results are summarized in Section 5. Finally, sec 6 concludes.

2. Related Works

In recent years, the topic of SAC in 5G has attracted a lot of attention. Some of these works focus on radio access network (RAN) slicing. Article [11] has addressed the opportunities and challenges of RAN slicing using ML-based techniques. An ensemble learning method-based SAC for adaptive RAN is proposed in [12]. Its objective is to support communication services. In [13] authors address the issue of SAC in the RAN by proposing the earliest deadline first (EDF) scheduling. Their reason for choosing RAN slicing is the limited resources compared to the network core. In the article [14], the authors believe that due to the stochastic nature of the wireless channels and complex resource coupling between slices, RAN slicing should be considered more than the fixed side of the network. Their objective is the reduction of resource usage while guaranteeing slice isolation and simultaneously accounting for each slice's average rate and delay requirements. Like the reported works, most of the other works [15–19] have addressed the

mentioned challenges in the field of radio slicing. In the new 5G network architectures such as Cloud-RAN and mobile MEC, the computing and storage functions are separated from the base stations and transferred to small cells, access points, microcells, CO, etc. [20]. For this reason, working on these sources is not as important as before and most of the new articles have dealt with the issue of SAC on cloud resources and edge devices. For example, to maximize long-term revenue a multi-agent deep reinforcement learning (DRL) for SAC in 5G C-RAN is proposed in [20]. The presented DRL method is able to learn the dynamics of S-REQ traffic and effectively address these joint issues. In [21], the authors proposed a two-timescale method for SAC in C-RAN. They formulated the utility maximization as a two-stage random scheduling problem and for solving this, first transformed the random scheduling problem into a deterministic optimization problem. They also apply semidefinite relaxation to transform the problem into a mixed integer nonconvex optimization problem, which is able to be solved via composition branch-and-bound and primal-relaxed dual techniques. To maximize the InP revenue by properly SAC in the C-RAN, tang et al [22] formulate it as mixed-integer nonlinear programming. In order to solve this problem, they used successive convex approximation and semidefinite relaxation approaches. However, the optimization algorithms used in [13, 14] have high computational costs making them impractical in the real world. The paper [23] presented an optimized multi-class classification resources management method based on the cooperative evolution of a support vector machine to assign the spectrum resources of macrocellular users in the C-RAN. The limitation of the presented method is the small number of provided services and considering only spectrum resources, while we consider computing and transmission resources. In the article [24], an innovative method was used to prevent service degradation for IoT applications in the fog environment. Their main goal is to maximize the number of users while we consider more services and the goals drawn are diverse. The reported works often only focus on the SAC problem or consider a small number of applications. Our scenario is such that, while dealing with the issue of SAC, also examine the resources required by the slices in operation. Two works close to our article are [25] and [8]. According to the authors [25] although an InP generates revenue by accepting an S-REQ, however, it might need to pay a penalty (proportional to the level of service degradation) if a slice cannot be scaled up when required. With this background, they have proposed a big data-based method for SAC in C-RAN. Their method requires knowledge and data that the algorithm must have and as well mentions in the article [8], it is almost impossible to achieve this knowledge in the real world of the network, so it is necessary to use methods that do not require this data. For the mentioned reason, in the next work [8] and the same scenario, they used the RL method that does not require prior knowledge. In the continuous environment where the number of actions and states are countless, RL has a fundamental weakness and in their proposed approach, how to overcome the existing weakness has not been addressed. Also, their scenario examined the core resources without considering the edge devices. While our proposed method covers most of the mentioned defects. Our contribution to this article is mentioned in the introduction section and will be described in a later part. A list of the indices, sets, parameters, and decision variables of the proposed AQL approach can be summarized in Table 1.

Table 1
Summary of Definition

RAN	Radio access network
AQL	Approximate Q-learning
5G	Five-generation mobile network
NS	Network slicing
SP	Service provider
InP	Infrastructure provider
F-RAN	Fog radio access network
FN	Fog nodes
AP	Access point
uRLLC	ultra-reliable and low-latency communications
eMBB	enhanced mobile broadband
mMTC	massive machine-type communication
DL	deep learning
DRL	deep reinforcement learning
RL	reinforcement learning
RM	Resource management
CO	CentAQL office
BPF	Baseband processing function
BS	Base station
SPP	special purpose processors
vPP	virtualized packet processor
VNFs	virtual network functions
HC	Health care
RS	Remote surgery
MEC	Mobile edge computing
LS	latency-sensitive
LT	Latency-tolerable
SAC	Slice admission control

3. System Model And Problem Formulation

3.1. System model

As shown in Fig. 1, we consider a three-layer model based on MEC that consists of, devices, fog nodes, and the CO. Each user device is connected to a fog node in the radio area which is connected to CO via a fronthaul link (FHL). We indicate the capacity of links with $\mathbb{W}$. The fog nodes have a limited computational function and handle services with the delay-sensitive application. Because some programs can tolerate some delay, require a slice of functions placed at either the CO or fog nodes. In Table. 2, we have included the sensitivity of the applications and the location of their required functions. The computing capacities of the nodes can be captured by the set of $C_n = \left(Q_1, \mathcal{A}_{max}^{(1)}\right), \ldots, \left(Q_n, \mathcal{A}_{max}^{(n)}\right) \ldots$,where Q_N and $\mathcal{A}_{max}^{(N)}$ denote the S-REQ queue and maximum ability of the Nth node. Also, a controller equipped with an AQL algorithm performs cross-domain management of radio, transport, and cloud resources. The number of accepted S-REQs must follow the rules: (1, 2, and 3).

$$\sum_{t=0}^{T} 1_{\{a_t \approx accept\}} = \mathcal{A}_{max}^{(n)}, (1)$$

Constraint (1) ensures that the number of accepted requests does not exceed the capacity of the servers. Where T is a time slot and a_t indicates accept S-REQ in time slot t.

$$\sum_{t=0}^{T} 1_{\{S-REQ_t \approx accept\}} \geq Q_n, \forall \mathbf{s} \geq \mathcal{T}_t \, (2)$$

Constraint (2) guarantees that when the node's capacity is full, the number of accepted S-REQ must be less/equal than $Q_n$and telorable time.

$$\sum_{\mathrm{l} \in \mathbb{W}} \sum_{n=1}^{|\mathrm{c}|} U_{ls} U_{nsc} \leq \mathbb{W} \forall \mathrm{n} \in \mathrm{c} | \mathcal{G}_n = 1 \, (3)$$

Constraint (3) guarantees that the bandwidth allocated does not exceed the capacity of the fronthaul link. Where $\mathcal{G}_n$ is the binary decision variable that shows whether a node has the capacity or not.

$$\mathcal{G}\mathrm{n} = \begin{cases} 1, if servernisactive \\ 0otherwise \end{cases} (4)$$

3.2. Revenue model

We consider the time-slotted framework that has been formed of a long time slot (LTS) and a short time slot (STL). At the beginning of LTS, the controller has to decide whether to accept/reject the received S-REQ, and at the start of each STS, the controller generates the beamformers. In this paper, according to Table. 2, we consider two classes of S-REQ, LS) and LT. We show the LS S-REQ set as $s^{SL} = \{1, \ldots, s^{SL}\}$ and LT S-REQ set as $s^{LT} = \{s^{SL} + 1, \ldots, s^{SL} + s^{LT}\}$, respectively. Also, assumed that each LTS contains Z equivalent STS and denote the set of all of them in one LTS as $\mathrm{Z} \triangleq 1, \ldots, Z\}$. A $REQ_{i,j}$ to be served if the total service time is less than the maximum amount of resources it requires during operation and tolerable time$\mathcal{T}_{i,j}$. Otherwise, the slice is not served and fails, formally,

$$\mathfrak{f}_s = \begin{cases} 1, if s_d^r + s_{i,j}^T + s_d^Q + s_{i,j}^{\mathcal{C}} \leq \mathcal{T}_{i,j}, b_{o,t} \\ 0, otherwise \end{cases}$$

5

The whole service time for a slice$s_{i,j}$ contains the remaining time s_d^r of the slice running in the chosen node,$s_{i,j}^T$ the transmission time to offload the slice to the CO, s_n^Q the waiting time in the queue before service, $s_{i,j}^{\mathcal{C}}$is computing time and $b_{o,t}$is the resource required a slice in operation state. Also, the price of each unit of computing and transport resources in the layers is represented by $\mathbb{M}_Z = \{\mathbb{M}_{n1,t}, \mathbb{M}_{n2,t}, \ldots, \mathbb{M}_{N_n,t}, \mathbb{M}_{Ln,t}\}$, where $\mathbb{M}_{DN,t}$indicates the price of the Nth server at time slot t, and $L_{n,t}$ defined as the price of the transport link between N nodes at the time slot t. The amount of resource usage of each and the links available is shown by $U_T = \{u_{d1,t}, u_{d2,t}, \ldots, u_{D_n,t}, u_{Ln,t}\}$ which $u_{D_n,t}$ $u_{Ln,t}$, indicates the amount of use of the Nth node from the DC D and $u_{Ln,t}$indicates the used capacity of the FHL. Each node will earn an income for InP during the period and based on the duration of the activity. The revenue can be calculated as:

$$\mathbf{InP_{rev}} = \sum_{\mathbf{t}=0}^{\mathbf{Z}} \mathbb{M}_N + \sum_{\mathbf{t}=0}^{\mathbf{Z}} \mathbb{M}_{\mathbf{Ln}} - \left(\sum s_{\mathbf{n}}^r s_{\mathbf{i,j}}^{\mathbf{T}} s_{\mathbf{n}}^{\mathbf{Q}} s_{\mathbf{i,j}}^{\mathcal{C}} + \sum_{\mathbf{t}=0}^{\mathbf{Z}} \mathbf{U_T} \right) (6)$$

As mentioned earlier, a penalty occurs when a slice fails to scale up during operation. The amount of the penalty is calculated according to the following formula for each service:

$$\mathrm{InP_{pen}} = \mathrm{InP_{rev}} \times \mathfrak{f}_s \left(-\mathrm{STS} \sum s_{\mathrm{n}}^r s_{\mathrm{i,j}}^{\mathrm{T}} s_{\mathrm{n}}^{\mathrm{Q}} s_{\mathrm{i,j}}^{\mathcal{C}} + \sum_{\mathrm{t}=0}^{\mathrm{T}} \mathrm{U_Z} \right) (7)$$

Table 2 Classification of services

Applications	Bandwidth	Latency	Executor nodes	Example
Latency-sensitive (LS)	≤ 5 Mbps	≤ 5 MS	Fog nodes	Remote surgery
	3–7 Mbps	≤ 10 MS	Fog nodes	Remote driving
Latency-telorable (LT)	≤ 10 Mbps	50 ms-1s	Fog nodes & CO	Augment reality
	15 ~ Mbps	≤ 5 s	Fog nodes & CO	Smart city
	≤ 100 Mbps	≤ 5 s	Fog nodes & CO	3D video

3.3. Problem formulation

In the incoming MEC computing field, the InP makes an effort to accept as many S-REQs as possible to maximize long-term revenue. However, a reason to limited resources it, is not possible to accept incoming S-REQ. For this purpose, the InP has to properly select and accept S-REQ. Besides the analysis from the last subsection, the LTS revenue from accepted S-REQ is:

$$\max_{a_0,\ldots,a_{Z-1}} \sum_{t=0}^{Z} 1_{\{a_t=accept} and \min_{a_0,\ldots,a_{Z-1}} \sum_{t=0}^{Z} 1_{\{a_t=reject\}}$$

$$\max \sum_{n=1}^{N} InP_{rev} \text{and Min } \sum_{n=1}^{N} \text{InP}_{\text{pen}} \, (8)$$

s.t.f$_s = 1, (1), (2), and\,(3)$

Problem (8) is NP-hard as it has non-linear and conditional constrain. We use the AQL method to solve this problem, which will be explained later.

4. Proposed Method

In this part, we explain the Q-learning method and will improve some of its capabilities including the train-validation phase for adopting SAC.

4.1. Q-learning algorithm

The Q-learning algorithm is one of the most powerful RL algorithms which seeks to find the optimal policy π^*in each state without the need for prior knowledge or data. Similar to all machine learning algorithms, the Q-learning follows the features of the Markov decision process (MDP) model, which is defined by a tuple (S, A, p, r). S, A are possible states and actions respectively, $p$is a probability of transferring from state S to s' by and, $r$is a reward function. In this article, the AQL agent is embedded in the cloud controller which is a charge for deciding whether to accept or reject S-REQs. On the other hand, the agent needs a policy that to able maximizing overall rewards as follows:

$$\pi^{*}=\underset{\pi}{\mathbf{argmax}}\mathbb{E}\left[\sum_{t\geq 0}\gamma^{\mathrm{t}}\mathcal{R}_{\mathbf{t}}\pi\right]\text{ s.t. }s_0\sim p\left(s_0\right)a_0\sim\pi\left(.\left|s_1\right.\right)a_t\sim\pi\left(.\left|s_t\right.\right)a_{t+1}\sim\mathbf{p}\left(.\left|s_t,a_t\right.\right)$$
(9)

Where γ is the discount factor and its amount (0 < γ > 1) and$\mathbb{E}$ is the expected value. We define the value function $V^{\pi}\left(s\right)$to represent the expected value that is able to be achieved through a policy πbeginning from state s as below.

$$V^{\pi}\left(s\right)=\mathbb{E}\left[\sum_{t\geq 0}\gamma^{\mathrm{t}}\mathcal{R}_{\mathrm{t}}\ \mathrm{s}_0=s,\pi\right](10)$$

The Q-learning effort to learn the value of taking a specific a for a given S, ie. Q-value, through a sequence updating the Q in a temporal difference (TD) manner as.

$$\mathrm{Q}\left(s,a\right)\leftarrow Q\left(s,a\right)\alpha\left[\mathcal{R}+\gamma\underset{a'}{\mathrm{max}}\mathrm{Q}(\mathrm{s}',a\prime)-Q\left(s,a\right)\right](11)$$

Whereα is the learning rate. Therefore, the Q-learning algorithm is able to obtain the optimal policy when action-state spaces are discrete. Since the 5G environment is massive and full of stochastic parameters, using traditional Q-learning leads the Q-value table to explode and lack convergence in the exploration and exploitation phases. Due to this, we improve this algorithm and introduce the AQL algorithm. The core idea of the AQL is $\mathrm{Q}\left(s,a\right)$ obtained from a linear combination of features:

$$\mathrm{Q}\left(\mathrm{s},a;\mathrm{w}\right)=\sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{f_i}\left(\mathrm{s},a\right)\mathrm{w_i}\left(12\right)$$

Here, we have added $\mathrm{f}\mathrm{and}\mathrm{w}$ parameters which represent a feature and its weight to the algorithm.Also, we have generated features according to the important characteristics of the system and given them a default value. The agent has to learn and update the weights for each feature from the model states. A feature $f\left(s,a\right)$ is defined over the state and action pairs, which yields a vector $\left(f_0\left(s,a\right).f_1\left(s,a\right)....f_i\left(s,a\right)....f_n\left(s,a\right)\right)$. The weights are updated according to the following rule:

$$Newsample:\left(s,a,s',r\right)$$

$$difference=\left[\mathrm{r}+\gamma\underset{\mathrm{a}'}{\mathrm{max}}\mathrm{Q}(\mathrm{s}',a\prime)=\right]-\mathrm{Q}\left(\mathrm{s},a\right)$$

$$\mathrm{w_i}\leftarrow\mathrm{w_i}+\alpha.\left[\mathrm{difference}\right]\times\mathrm{f_i}(\mathrm{s},\mathrm{a})$$

$$\mathrm{w_i}\leftarrow\mathrm{w_i}+\alpha.\left(\mathrm{r}+\gamma\underset{\mathrm{a}'}{\mathrm{max}}\mathrm{Q}\left(\mathrm{s}',a'\right)-\mathrm{Q}\left(\mathrm{s},a\right)\right)\times\mathrm{f_i}\left(\mathrm{s},a\right)$$

13

Algorithm

, shows the learning process of the proposed method by pseudo-code. To use ARL to SAC, firstly, we must extract and define the related features and weights. In order to, the capacities of fog nodes, CO and FHL transfer to a container, and, their values are numbered according to the standardization made. Also, one counter-deployment into each container represents the resource status. Based on these criteria, we were able to define 5 proper to improve the performance of the algorithm:

(f_1) Accepting S-REQ whenever they arrive

(f_2) Reaching the counter to the container roof

(f_3) Accepting S-REQ with high profit at any time

(f_4) The proximity of the counter to the container floor

(f_5) Fairness in accepting different types of class

Therefore, Initial weights are given for the 5 features that are:

$$\mathbf{Q}(s,a) = -w_1 f_1(s,a) + w_2 f_2(s,a) + w_3 f_3(s,a) - w_4 f_4(s,a) + w_5 f_5(s,a) \quad (14)$$

Algorithm 1, AQL

1- Set features from the model states

2- Initialize random weights

3- Reward function $R : S \times A \longrightarrow R$

4- The agent is chosen a with probability epsilon value ϵ;

5- Learning rate: $\alpha \in [0,1]$

6- Discounting factor: $\gamma \in [0,1]$

7- Procedure ARL $(S, A, T, \alpha, \gamma)$

8- Initialize $Q(s,a), \forall s \in S, a \in A(s), \text{arbitrarily, and} Q(\text{terminal} - \text{state}, .) = 0$

9- While weights are not converged do

10- $\mathrm{w_i} \leftarrow \mathrm{w_i} + \alpha . \left(\mathrm{r} + \gamma \underset{\mathrm{a'}}{\mathrm{max}} \mathrm{Q}(\mathrm{s'}, a') - \mathrm{Q}(\mathrm{s}, a) \right) \times \mathrm{f_i}(\mathrm{s}, a)$

11- While weights are not converged and terminal

12- Repeat (for each step of the episode):

Update weights

13- $\mathrm{Update} \mathrm{Q}(s,a)$

14- Return Q

15- Update the policy

5. Scenario Description And Simulation Results

5.1. Scenario description

This part explains the evaluation of the proposed AQL. It tests by using custom-built python-based PyTorch - an open-source ML framework- that accelerates the route from research prototyping to

production deployment. In addition, PyTorch used several RL-specific libraries such as TensorFlow, Cras, etc. to handle RL methods. To evaluate the AQL, a stochastic topology with 10 Fog nodes and 2 CO is designed. We dedicated 2 types of resources named CPU and bandwidth to physical nodes with 20 units to each fog node and 50 units to each CO, according to the article [1, 2]. We assume communication resource is enough. We consider 5 SPs with an average of 100 users related to them and different sensitivities and a variety of requirements. These requirements include delay tolerance and needed resources at different levels. The needed resource for each slice follows a uniform distribution between 1 and 15 resource units during each time step of the service. Also, in the proposed scenario, the operation time of a slice is from 1 to 24 hours with an interval of 1 hour. 2 types of SP are service class 1 and serve this application in edge device and class 2 that required function provision both edge and CO nodes. The AQL is compared against three methods including Deep reinforcement learning (DRL) also known as Deep Q-network (DQN), which is state-of-the-art, and this method has been used in some recent works [26–28], Q-learning, and random. We have used the Q-learning algorithm to better show the speed of convergence and the cumulative rewards. Additionally, random uses a first-come, first-served approach, and any S-REQ sent to the controller first is served. Gamma value, learning rate, and other hyperparameters are considered the same in all intelligent algorithms, and by changing the parameters in the proposed algorithm, the same value is set for other methods under test. In addition, to evaluate the efficiency of the algorithm, the methods have been evaluated in terms of convergence speed and cumulative reward, and to check the performance of the methods in the network environment, resource usage and gaining revenue, and have been tested. Also, in this article, in order to better show the performance of the approaches, we converted all the tested values into percentages by the normalization method. To achieve more information about DRL, it is recommended to study articles [29–31].

5.2. Simulation results

In this section, the convergence speed of the algorithms is first examined. The convergence of RL algorithms depends on the repetition of operations (iteration) in the exploration phase. On the other hand, several tests (episodes) are needed in the training phase so that the algorithm can enter the exploitation phase. We claim that our algorithm has the capability of fast convergence even in large environments. This claim is based on extracting features and assigning default weights to them so that instead of training the actions in each state, the algorithm only learns the weight of the features and performs the corresponding action based on the weights. Figure 2. shows the performance of the methods in terms of convergence. In this experiment, we have set the learning rate of the algorithms in three different numbers 0.25, 0.5, and 0.75. The closer the number is to 1, the more weight gives to newer experiences. The test output under all three numbers set for the learning rate is averaged and shown as a graph in Fig. 2. Also, 5000 episodes were performed during the simulation. For a better understanding of the performances, the convergence on line 0 is set. As can be seen, the AQL has converged with the adjusted line in the same initial episodes, and this is because the algorithm has avoided unnecessary actions in each state and only updated the weights according to the features. Needless to say, the DQN algorithm also has a good performance due to the use of a target network and mini-batching from that network, but despite the power of DQN, AQL has a better performance than this method. Because the Q-learning stores all the

actions/states in the Q-value table and performs the optimal policy after obtaining all the information, it performs poorly in larger dimensions.

Another important parameter for evaluating RL algorithms is cumulative received rewards. In each action, if resources are allocated to a slice, a score is added to the rewards. If we cannot add the resource to the active slices when required, a punishment will be imposed on the agent. Optimal politics are the maximization of the total rewards. We have tested cumulative rewards to evaluate the efficiency of the algorithm and graphed the result in Fig. 3. As shown by the green line, the proposed method seeks to increase this parameter by receiving rewards greedily, and with the first punishment, the greedy movement stops and collects the rewards intelligently. Also, leaving the CO resource capacity empty while the edge resources are fully consumed leads to punishment. For simplicity, we set the maximum reward received on line 1 and compare the methods with each other. However, the AQL reaches the mentioned line faster than other methods.

The goal of the InP is to make money by renting out resources. This revenue is priced per unit and the resulting profit is estimated in this way. Figure 4, shows the average earned income according to rule (6). We have mapped the total price of resources in percentage and we show the amount of acquisition with it. Because the request method is distributed, at the beginning of the slice acceptance process, the random method due to the lack of a specific strategy, has been able to earn 40%. Since the use of the CO resource depends on the existence of empty capacity in the Fog-node, by accepting each request, the capacities of the fog devices are completed and it leads to no significant increase in income in the random method. For the total price of the resources, the proposed method has been able to earn about 93% of the income. Compared to the DRL method, our method performs 14% better this semester.

With the optimal performance of the AQL in the field of revenue, the proposed method should work better in resource usage and prevent them from being idle. The amount of resource consumption is checked at the edge locations, connections, and CO. The usage of these resources has been tested and the average consumption of all resources is shown in the form of a linear diagram in Fig. 5. As can be seen, with the increase in the number of users, the usage of the random method increases to about 40%, and the majority of this growth is estimated to be due to the consumption of resources at the edge nodes. With the increase of S-REQ, the number of consumed resources has not grown significantly and this shows that if smart methods are not used, network resources will be wasted. The proposed method has an acceptable resource usage rate compared to other smart methods, so with the increase of S-REQ, this rate has reached about 85%. The DQN method has a high usage rate compared to the proposed method but considering that the AQL revenue is about 18% better than DQN, we conclude that the proposed method has a smarter consumption. Also, AQL kept about 15% of the resources for possible requests of slices, which shows that training the agent based on features 2 and 3 is effective and that the agent has the necessary intelligence.

6. Conclusion

In this work, we have proposed an approximate Q-learning-based network slicing that can be used in a mobile edge-cloud (MEC) where multiple applications ask for resource slices to accommodate service with different requirements constraints. Based on NFV-enabled resource function virtualization, the resources of the MEC are sliced among various applications to improve resource utilization and serve more customers. Our main finding is that the traditional machine learning method cannot perform well in MEC environments with continuous space and infinitive actions. Additionally, we receive that an intelligent slice acceptance policy can to maximize the overall net profit. For this reason, we enhance reinforcement learning performance by an added extra parameters for the controller agent rapidly converge between the exploration and exploitation phase and find the optimal policy for the slice acceptance control. The experiment results have shown that the proposed method outperforms the Q-learning, deep reinforcement learning, and random approaches and can better behave to maximize resource utilization, convergence speed, cumulative rewards. In future work, researchers can investigate the resource allocation for delay-sensitive applications in the environment of cloud-RAN 5G networks via reinforcement learning methods.

Declarations

Funding There are no sponsors or funding sources for this article

Competing interests

The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Author names:

Mohsen khani

Shahram jamali

Mohammad karim sohrabil

Availability of data and materials

Because the proposed method does not require a model and previous knowledge, therefore, no data or datasets have been used.

Ethical Approval

All scientific, academic, and policy ethics of the journal are approved.

Authors' contributions

This article is the result of several months of study and research, which is extracted from the doctoral thesis. Also, author A is responsible for presenting the research idea, designing the study, collecting data, and analyzing and interpreting the data. Author B collaborated in writing the initial version of the manuscript or revising and critically reviewing it in such a way that the text changes and evolves in terms of scientific content. Author C is responsible for reading and approval of the final version of the manuscript before submission

References

1. Rost, P., et al., *Network slicing to enable scalability and flexibility in 5G mobile networks.* IEEE Communications magazine, 2017. **55**(5): p. 72–79.
2. Salhab, N., R. Langar, and R. Rahim, *5G network slices resource orchestration using Machine Learning techniques.* Computer Networks, 2021. **188**: p. 107829.
3. Liu, H., et al., *Mobile edge cloud system: Architectures, challenges, and approaches.* IEEE Systems Journal, 2017. **12**(3): p. 2495–2508.
4. Shakarami, A., et al., *A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective.* Journal of Grid Computing, 2020. **18**(4): p. 639–671.
5. Chantre, H.D. and N.L. da Fonseca, *Multi-objective optimization for edge device placement and reliable broadcasting in 5G NFV-based small cell networks.* IEEE Journal on Selected Areas in Communications, 2018. **36**(10): p. 2304–2317.
6. Mohammadi, F., S. Jamali, and M. Bekravi, *Survey on job scheduling algorithms in cloud computing.* International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), 2014. **3**(2): p. 151–154.
7. Xiang, H., et al., *Network slicing in fog radio access networks: Issues and challenges.* IEEE Communications Magazine, 2017. **55**(12): p. 110–116.
8. Raza, M.R., et al., *Reinforcement learning for slicing in a 5G flexible RAN.* Journal of Lightwave Technology, 2019. **37**(20): p. 5161–5169.
9. Watkins, C.J. and P. Dayan, *Q-learning.* Machine learning, 1992. **8**(3): p. 279–292.
10. Clifton, J. and E. Laber, *Q-learning: theory and applications.* Annual Review of Statistics and Its Application, 2020. **7**: p. 279–301.
11. Mazied, E.A., L. Liu, and S.F. Midkiff, *Towards Intelligent RAN Slicing for B5G: Opportunities and Challenges.* arXiv preprint arXiv:2103.00227, 2021.
12. Moon, S.I., et al. *Ensemble learning method-based slice admission control for adaptive RAN*. in *2020 IEEE Globecom Workshops (GC Wkshps*. 2020. IEEE.

13. Guo, T. and A. Suárez, *Enabling 5G RAN slicing with EDF slice scheduling.* IEEE Transactions on Vehicular Technology, 2019. **68**(3): p. 2865–2877.
14. Papa, A., et al. *Optimizing dynamic RAN slicing in programmable 5G networks*. in *ICC 2019–2019 IEEE International Conference on Communications (ICC)*. 2019. IEEE.
15. Bakri, S., B. Brik, and A. Ksentini, *On using reinforcement learning for network slice admission control in 5G: Offline vs. online.* International Journal of Communication Systems, 2021. **34**(7): p. e4757.
16. Vila, I., et al. *Performance measurements-based estimation of radio resource requirements for slice admission control*. in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. 2019. IEEE.
17. Pérez-Romero, J. and O. Sallent, *Optimization of multitenant radio admission control through a semi-Markov decision process.* IEEE Transactions on Vehicular Technology, 2019. **69**(1): p. 862–875.
18. Yang, J., et al., *Joint admission control and routing via approximate dynamic programming for streaming video over software-defined networking.* IEEE Transactions on Multimedia, 2016. **19**(3): p. 619–631.
19. Kulmar, M., I. Müürsepp, and M.M. Alam. *The Impact of RAN Slice Bandwidth Subpartitioning on Slice Performance*. in *2022 International Wireless Communications and Mobile Computing (IWCMC)*. 2022. IEEE.
20. Sulaiman, M., et al. *Multi-Agent Deep Reinforcement Learning for Slicing and Admission Control in 5G C-RAN*. in *NOMS 2022–2022 IEEE/IFIP Network Operations and Management Symposium*. 2022. IEEE.
21. Zhang, H. and V.W. Wong, *A two-timescale approach for network slicing in C-RAN.* IEEE Transactions on Vehicular Technology, 2020. **69**(6): p. 6656–6669.
22. Tang, J., B. Shim, and T.Q. Quek, *Service multiplexing and revenue maximization in sliced C-RAN incorporated with URLLC and multicast eMBB.* IEEE Journal on Selected Areas in Communications, 2019. **37**(4): p. 881–895.
23. Kumar, N. and A. Ahmad, *Cooperative evolution of support vector machine empowered knowledge-based radio resource management for 5G C-RAN.* Ad Hoc Networks, 2022. **136**: p. 102960.
24. Ai, Y., et al., *Joint resource allocation and admission control in sliced fog radio access networks.* China Communications, 2020. **17**(8): p. 14–30.
25. Raza, M.R., et al., *A slice admission policy based on big data analytics for multi-tenant 5G networks.* Journal of Lightwave Technology, 2019. **37**(7): p. 1690–1697.
26. Li, R., et al., *Deep reinforcement learning for resource management in network slicing.* IEEE Access, 2018. **6**: p. 74429–74441.
27. Wang, H., et al., *Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach.* Information Sciences, 2019. **498**: p. 106–116.
28. Qi, C., et al., *Deep reinforcement learning with discrete normalized advantage functions for resource management in network slicing.* IEEE Communications Letters, 2019. **23**(8): p. 1337–1341.
29. Li, Y., *Deep reinforcement learning: An overview.* arXiv preprint arXiv:1701.07274, 2017.

30. Arulkumaran, K., et al., *Deep reinforcement learning: A brief survey.* IEEE Signal Processing Magazine, 2017. **34**(6): p. 26–38.
31. Mousavi, S.S., M. Schukat, and E. Howley. *Deep reinforcement learning: an overview*. in *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 2*. 2018. Springer.

Figures

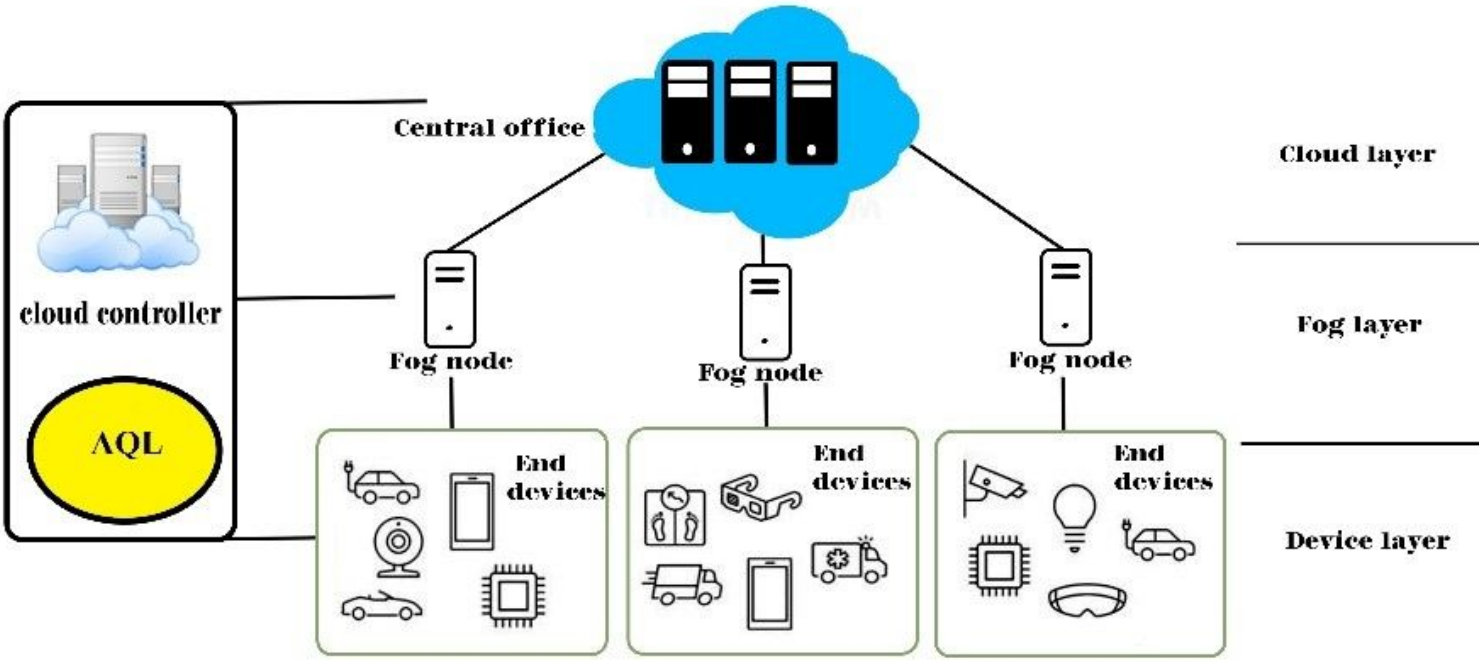

Figure 1

Edge computing model

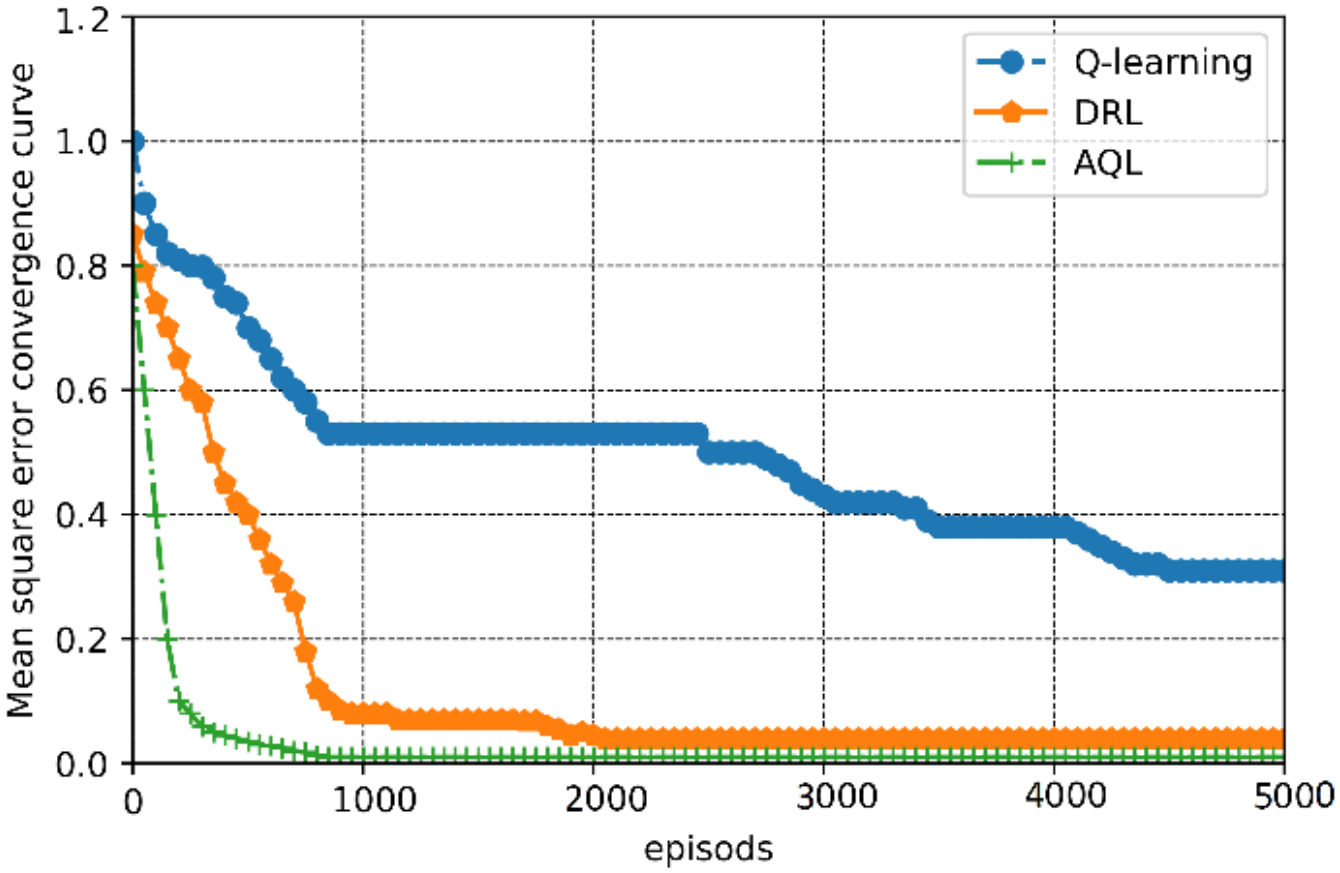

Figure 2

Convergence speed between exploration and exploitation

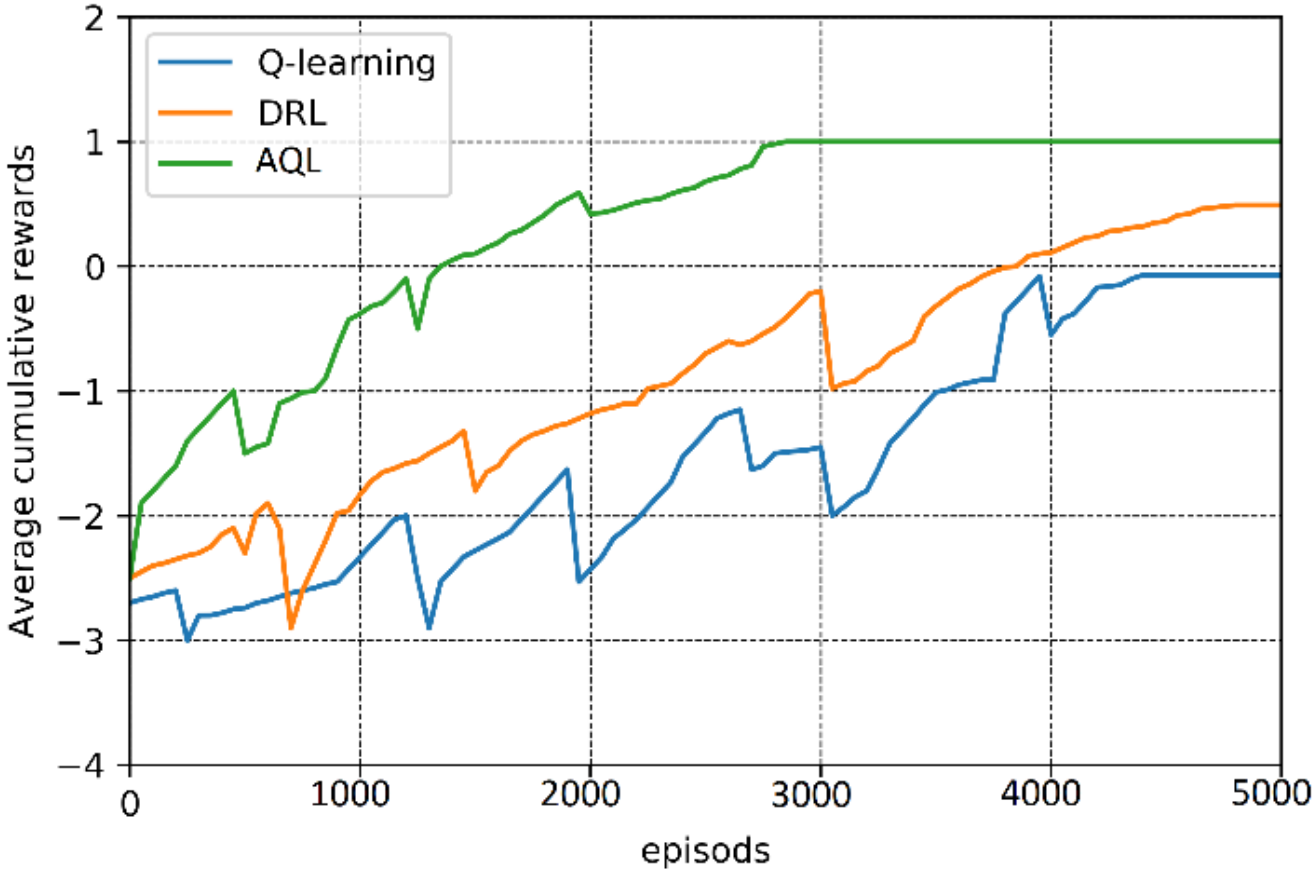

Figure 3

Average cumulative rewards

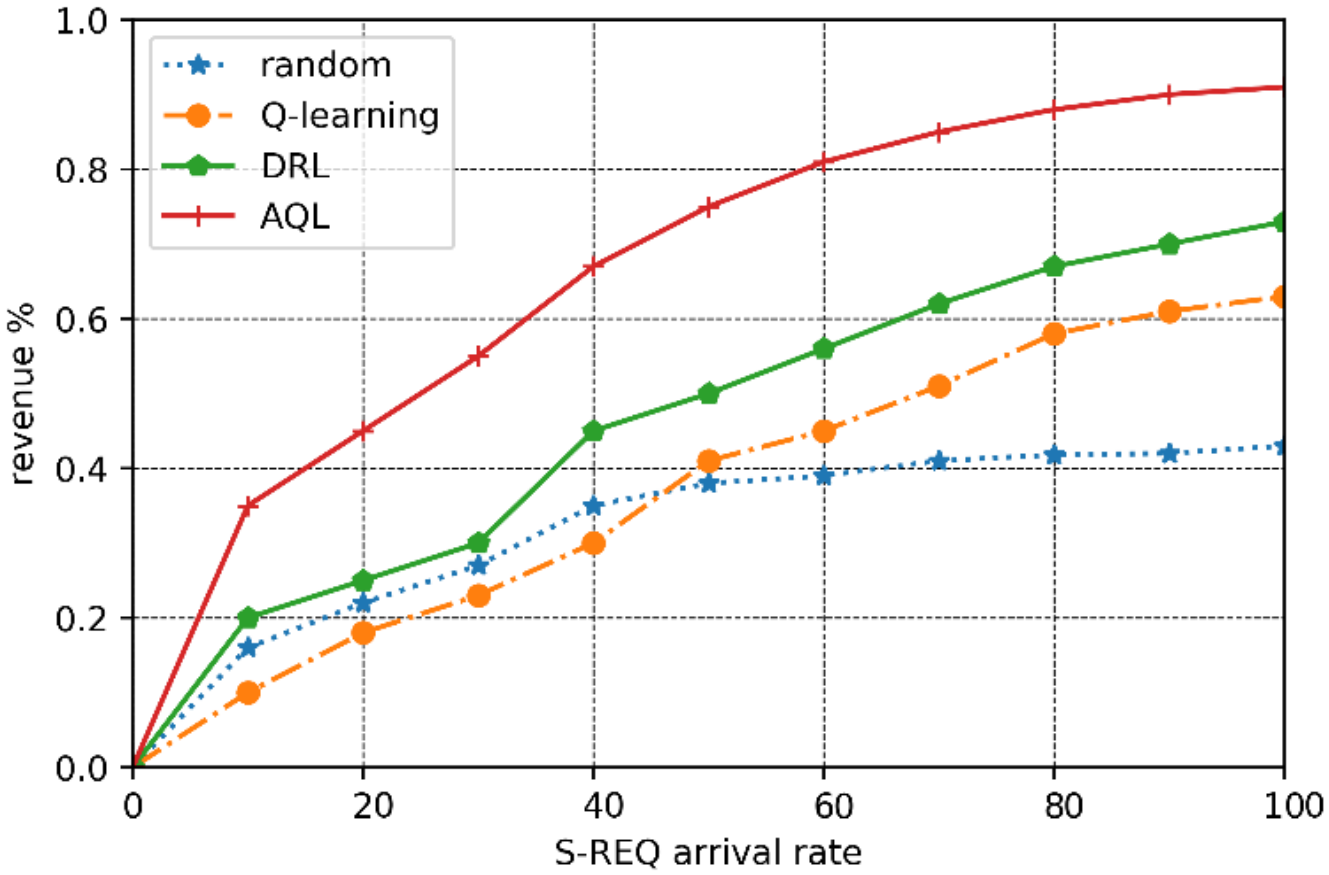

Figure 4

Average earned revenue

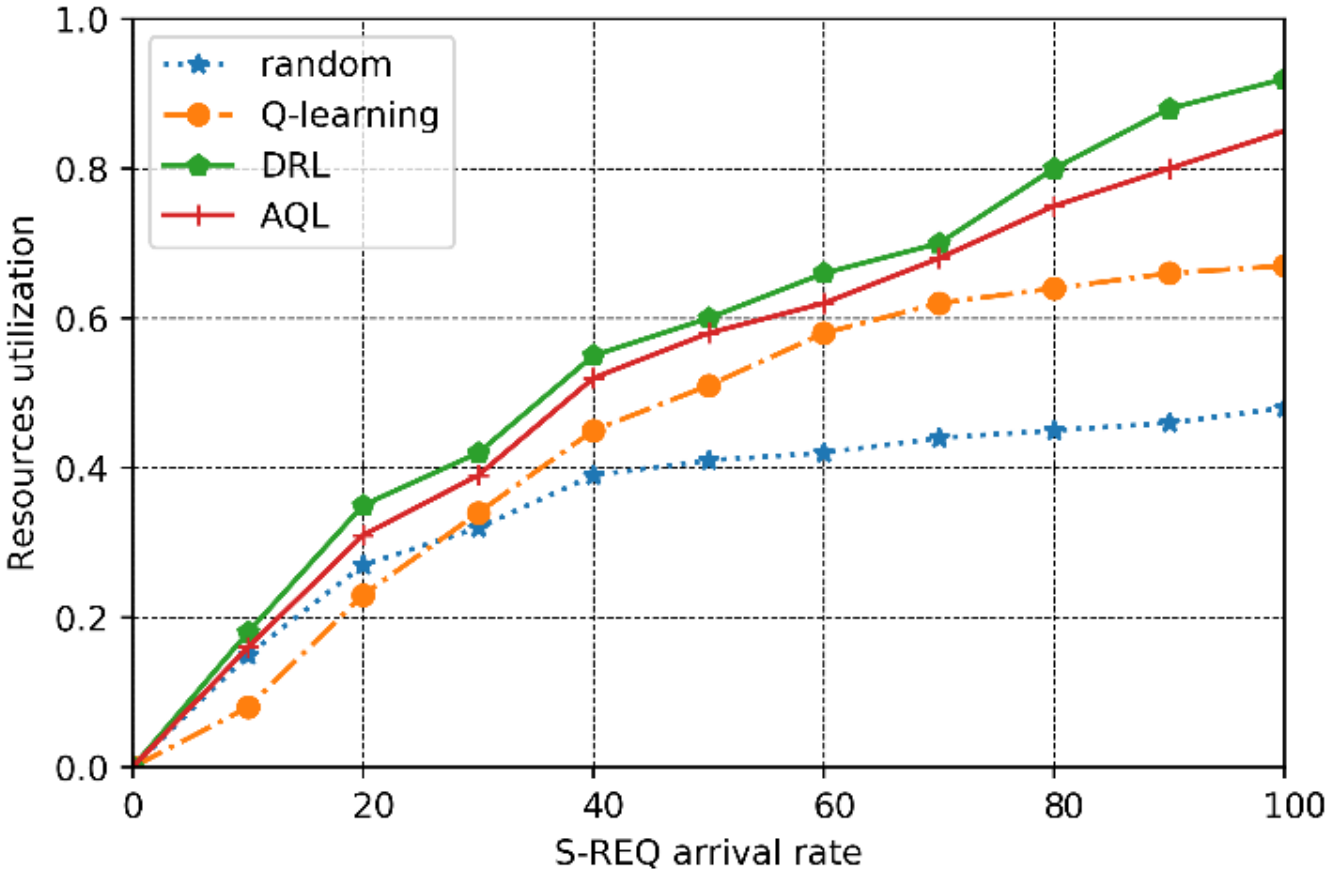

Figure 5

The average resource usage