

# Deep Reinforcement Learning for Resource Management in Network Slicing

Rongpeng Li, Zhifeng Zhao, Qi Sun, Chi-Lin I, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang

**Abstract**—Network slicing is born as an emerging business to operators, by allowing them to sell the customized slices to various tenants at different prices. In order to provide better-performing and cost-efficient services, network slicing involves challenging technical issues and urgently looks forward to intelligent innovations to make the resource management consistent with users’ activities per slice. In that regard, deep reinforcement learning (DRL), which focuses on how to interact with the environment by trying alternative actions and reinforcing the tendency actions producing more rewarding consequences, is assumed to be a promising solution. In this paper, after briefly reviewing the fundamental concepts of DRL, we investigate the application of DRL in solving some typical resource management for network slicing scenarios, which include radio resource slicing and priority-based core network slicing, and demonstrate the advantage of DRL over several competing schemes through extensive simulations. Finally, we also discuss the possible challenges to apply DRL in network slicing from a general perspective.

**Index Terms**—Deep Reinforcement Learning, Network Slicing, Neural Networks, Q-Learning, Resource Management

## I. Introduction

The fifth-generation cellular networks (5G) is assumed to be the key infrastructure provider for the next decade, by means of profound changes in both radio technologies and network architecture design [1]–[4]. Besides the pure performance metrics like rate, reliability and allowed connections, the scope of 5G also incorporates the transformation of the mobile network ecosystem and accommodates heterogeneous services using one infrastructure. In order to achieve such a goal, 5G will fully glean the recent advances in the network virtualization and programmability [1], [2], and provide a novel technique named network slicing [1], [5]–[7]. Network slicing tries to get rid of the current, relatively monolithic architecture like the forth-generation cellular networks (4G) and slice the whole network into different parts, each of which is tailed to meet specific service requirement. Therefore, network slicing is born as

an emerging business to operators and allows them to sell the customized network slices to various tenants at different prices. In a word, network slicing could act as a service (NSaaS) [5]. NSaaS is quite similar to the mature business “infrastructure as a service (IaaS)”, the benefit of which service providers like Amazon and Microsoft have happily enjoyed for a while. However, in order to provide better-performing and cost-efficient services, network slicing involves more challenging technical issues even for the real-time resource management on existing slices, since (a) for radio access networks, spectrum is a scarce resource and it is meaningful to guarantee the spectrum efficiency (SE) [8], while for core networks, virtualized functionalities are limited by computing resources; (b) the service level agreements (SLAs) with slice tenants usually impose stringent requirements on quality of experience (QoE) perceived by users [9]; and (c) the actual demand of each slice heavily depends on the request patterns of mobile users. Hence, in the 5G era, it is critical to investigate how to intelligently respond to the dynamics of service request from mobile users [7], so as to obtain satisfactory QoE in each slice at the cost of acceptable spectrum or computing resources [4]. There has been several works towards the resource management for the network slicing, particularly in specific scenarios like edge computing [?] and Internet of things [?]. However, it is still very appealing to discuss a approach in generalized scenarios. In that regard, [?] proposes to adopt genetic algorithm as an evolutionary means for inter-slice resource management. However, [?] does not reflect the explicit relationship that one slice might require more resources due to its more stringent SLA.

On the other hand, partially inspired by the psychology of human learning, the learning agent in reinforcement learning (RL) algorithm focuses on how to interact with the environment (represented by states) by trying alternative actions and reinforcing the tendency actions producing more rewarding consequences [10]. Besides, reinforcement learning also embraces the theory of optimal control and adopts some ideas like value functions and dynamic programming. However, reinforcement learning faces some difficulties in dealing with large state space, since it is challenging to traverse every state and obtain a value function or model for every station-action pair in a direct and explicit manner. Hence, benefiting from the advances in graphics processing units (GPUs) and the less concern for the computing power, some researchers

R. Li, Z. Zhao, M. Zhao and H. Zhang are with Zhejiang University, Hangzhou 310027, China, (email: {lirongpeng, zhaozf, mjzhao, honggangzhang}@zju.edu.cn).

Q. Sun and C.-L. I are with China Mobile Research Institute, Beijing 100053, China (email: {sunqiyjy, icl}@chinamobile.com).

C. Yang is with Beihang University, Beijing, 100191, China (email: cyyang@buaa.edu.cn)

X. Chen is with VTT Technical Research Centre of Finland, Oulu FI-90571, Finland (email: xianfu.chen@vtt.fi).

This work was supported in part by National Key R&D Program of China (No. 2018YFB0803702), National Natural Science Foundation of China (No. 61701439, 61731002), Zhejiang Key Research and Development Plan (No. 2018C03056).

propose to sample only a fraction of states and further apply neural networks (NN) to train a sufficiently accurate value function or model. Following this idea, Google DeepMind has pioneered to combine NN with one typical RL algorithm (i.e.,  $Q$ -Learning), and proposed one deep reinforcement learning (DRL) algorithm with enough performance stabilities [11], [12].

The well-known success of AlphaGo [11] and following exciting results to apply DRL to solve resource allocation issues in some specific fields like power control [13], green communications [14], cloud radio access networks [15], mobile edge computing and caching [?], [?], [16], have aroused some research interest to apply DRL to the field of network slicing. However, given the challenging technical issues in the resource management on existing slices, it is critical to carefully investigate the performance of applying DRL in the following aspects:

- The basic concern is whether or not the application of DRL is feasible. More specifically, does DRL produce satisfactory QoE results while consuming acceptable network resources (e.g., spectrum)?
- The research community has proposed some schemes for the resource management in network slicing scenarios. For example, the resource management could be conducted by either following a meticulously designed prediction algorithm, or equally dividing the available resource into each slice. The former implies one reasonable option, while the latter saves a lot of computational cost. Hence, a comparison between DRL and these interesting schemes is also necessary.

In this paper, we strive to address these issues.

The remainder of the paper is organized as follows. Section II starts with the fundamentals of RL and talks about the motivation to evolve towards DRL from RL. As the main part of the paper, Section III addresses two resource management issues in network slicing and highlights the advantages of DRL by extensive simulation analyses. Section IV concludes the paper and points out some research directions to apply DRL in a general manner.

## II. From Reinforcement Learning to Deep Reinforcement Learning

In this section, we give a brief introduction over RL or more specifically  $Q$ -Learning, and then talk about the motivation to evolve from  $Q$ -Learning to Deep  $Q$ -Learning (DQL).

### A. Reinforcement Learning

RL learns how to interact with the environment to achieve maximum cumulative return (or average return), and has been successfully applied in the fields like robot control, self driving, and chess playing for years. Mathematically, RL follows the typical concept of Markov decision process (MDP), while the MDP is a generalized framework for modeling decision-making problems in cases where the result is partially random and affected by the

applied decision. An MDP can be formulated by a 5-tuple as  $M = \langle S, A, P(s'|s, a), R, \gamma \rangle$ , where  $S$  and  $A$  denote a finite state space and action set, respectively.  $P(s'|s, a)$  indicates the probability that the action  $a \in A$  under state  $s \in S$  at slot  $t$  leads to state  $s' \in S$  at slot  $t + 1$ .  $R(s, a)$  is an immediate reward after performing the action  $a$  under state  $s$ , while  $\gamma \in [0, 1]$  is a discount factor to reflect the diminishing importance of current reward on future ones. Usually, the goal of MDP is to find a policy  $a = \pi(s)$  that determines the selected action  $a$  under state  $s$ , so as to maximize the value function, which is typically defined as the expected discounted cumulative reward by the Bellman equation:

$$\begin{aligned} V^\pi(\hat{s}) &= E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R(s^{(k)}, \pi(s^{(k)})) | s^{(0)} = \hat{s} \right] \\ &= E_\pi \left[ R(\hat{s}, \pi(\hat{s})) + \gamma \sum_{s' \in S} P(s' | \hat{s}, \pi(\hat{s})) V^\pi(s') \right]. \end{aligned} \quad (1)$$

Dynamic programming could be exploited to solve the Bellman equation when the state transition probability  $P(s'|s, a)$  is known apriori with no random factors. But inspired by both control theory and behaviorist psychology, RL aims to obtain the optimal policy  $\pi^*$  under circumstances with unknown and partially random dynamics. Since RL does not have explicit knowledge over whether it has come close to its goal, it needs the balance between exploring new potential actions and exploiting the already learnt experience. So far, there has been some classical RL algorithms like  $Q$ -learning, actor-critic method, SARSA, TD( $\lambda$ ), etc [10]. Given by the detailed methodologies and practical application scenarios, we can classify these RL algorithms according to different criteria:

- Model-based versus Model-free: Model-based algorithms imply the agent tries to learn the model of how the environment works from its observations and then plan a solution using that model. Once the agent gains adequately accurate model, it can use a planning algorithm with its learned model to find a policy. Model-free algorithms means the agent does not directly learn how to model the environment. Instead, like the classical example of  $Q$ -learning, the agent estimates the  $Q$ -values (or roughly the value function) of each state-action pair and derives the optimal policy by choosing the action yielding the largest  $Q$ -value in the current state. Different from the model-based algorithm, the well-learned model-free algorithm like  $Q$ -learning cannot predict the next state and value before taking the action.
- Monte-Carlo Update versus Temporal-Difference Update: Generally, the value function update could be conducted in two ways, that is, the Monte-Carlo update and the temporal-difference (TD) update. A Monte-Carlo update means the agent updates its estimation for a state-action pair by calculating the mean return from a collection of episodes. A TD update approximates the estimation by comparing estimates

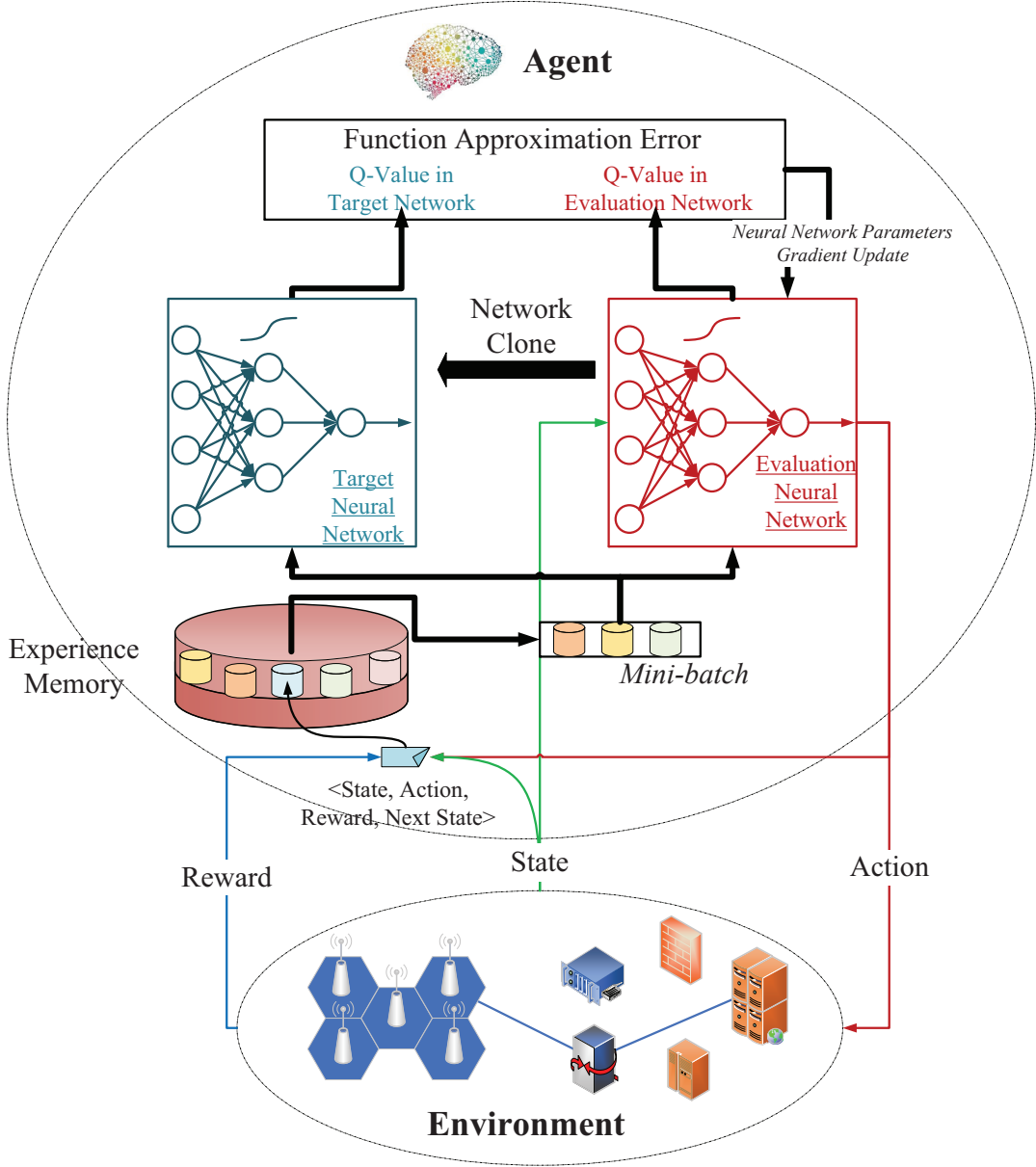


Fig. 1. An illustration of deep  $Q$ -learning.

at two consecutive episodes. For example,  $Q$ -learning updates its  $Q$ -value by the TD update as  $Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a))$ , where  $\alpha$  is the learning rate. Specifically, the term  $R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)$  is also named as the TD error, since it captures the difference between the current (sampled) estimate  $R(s, a) + \gamma \max_{a'} Q(s', a')$  and previous one  $Q(s, a)$ .

- On-policy versus Off-policy: The value function update is also coupled with the executed update policy. Before updating the value function, the agent also needs to sample and learn the environment by performing some non-optimal policy. If the update policy is irrelevant to the sampling policy, the agent is called to perform an off-policy update. Taking the example of  $Q$ -learning, this off-policy

agent updates the  $Q$ -value by choosing the action corresponding to the best  $Q$ -value, while it could learn the environment by adopting sampling policies like  $\epsilon$ -greedy or Boltzmann distribution to balance the “exploration and exploitation” problem [10]. The  $Q$ -learning proves to converge regardless of the chosen sampling policy. On the contrary, the SARSA agent is on-policy, since it updates the value function by  $Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a) + \gamma Q(s', a') - Q(s, a))$  where  $a'$  and  $a$  need to be chosen according to the same policy.

#### B. From $Q$ -Learning to Deep $Q$ -Learning

We first summarize the details of  $Q$ -Learning. Generally speaking,  $Q$ -Learning belongs to a model-free, TD update, off-policy RL algorithm, and consists of three major steps:

- 1) The agent chooses an action  $a$  under state  $s$  according to some policy like  $\epsilon$ -greedy. Here, the  $\epsilon$ -greedy policy means the agent chooses the action with the largest Q-value  $Q(s, a)$  with a probability of  $\epsilon$ , and equally chooses the other actions with a probability of  $\frac{1-\epsilon}{|A|}$ , where  $|A|$  denotes the size of the action space.
- 2) The agent learns the reward  $R(s, a)$  from the environment, and the state transitions to the next state  $s'$ .
- 3) The agent updates the Q-value function in a TD manner as  $Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a))$ .

Classical RL algorithms usually rely on two different ways (i.e., explicit table or function approximation) to store the estimated value functions. For the table storage, RL algorithm uses an array or hash table to store the learnt results for each state-action pair. For large state space, it not only requires intensive storage, but also is unable to quickly transverse the complete the state-action pair. Due to the curse of dimensionality, function approximation sounds more appealing.

The most straightforward way for function approximation is a linear approach. Taking the example of  $Q$ -learning, the Q-value function could be approximated by a linear combination of  $n$  orthogonal bases  $\psi(s, a) = \{\psi_1(s, a), \dots, \psi_n(s, a)\}$ , that is,  $Q(s, a) = \theta_0 \cdot 1 + \theta_1 \cdot \psi_1(s, a) + \dots + \theta_n \cdot \psi_n(s, a) = \boldsymbol{\theta}^T \boldsymbol{\psi}(s, a)$ , where  $\theta_0$  is a biased term with 1 absorbed into the  $\boldsymbol{\psi}$  for simplicity of representation and  $\boldsymbol{\theta}$  is a vector with the dimension of  $n$ . The function approximation in the  $Q$ -learning means that  $Q(s, a) = \boldsymbol{\theta}^T \boldsymbol{\psi}(s, a)$  should be as close as the learnt “target” value  $Q^+(s, a) = \sum_s P(s'|s, a)[R(s, a) + \gamma \max_{a'} Q^+(s', a')]$  over all the state-action pairs. Since it is infeasible to transverse all the state-action pairs, the “target” value could be approximated based on the mini-batch samples and  $Q^+(s, a) \approx R(s, a) + \gamma \max_{a'} Q^+(s', a')$ . In order to make  $Q(s, a) = \boldsymbol{\theta}^T \boldsymbol{\psi}(s, a)$  approach the “target” value  $Q^+(s, a)$ , the objective function could be defined as

$$\begin{aligned} L(\boldsymbol{\theta}) &= \frac{1}{2} (Q^+(s, a) - Q(s, a))^2 \\ &= \frac{1}{2} (Q^+(s, a) - \boldsymbol{\theta}^T \boldsymbol{\psi}(s, a))^2. \end{aligned} \quad (2)$$

The parameter  $\boldsymbol{\theta}$  minimizing  $L(\boldsymbol{\theta})$  could be achieved by a gradient-based approach as

$$\begin{aligned} &\boldsymbol{\theta}^{(i+1)} \\ &\leftarrow \boldsymbol{\theta}^{(i)} - \alpha \nabla L(\boldsymbol{\theta}^{(i)}) \\ &= \boldsymbol{\theta}^{(i)} - \alpha (Q^+(s, a) - \boldsymbol{\theta}^T \boldsymbol{\psi}(s, a)) \boldsymbol{\psi}(s, a). \end{aligned} \quad (3)$$

For a large state-action space, the function approximation reduces the number of unknown parameters to a vector with dimension  $n$  and the related gradient method further solves the parameter approximation in an computationally efficient manner.

Apparently, the linear function approximation could not accurately model the estimated value function. Hence, researchers have proposed to replace the approximation  $Q(s, a; \boldsymbol{\theta})$  by some non-linear means. In that regard, NN is skilled in approximating non-linear functions [17]. Therefore, in AlphaGo [11], [12], NN has been exploited and the loss function can be re-defined as  $L(\boldsymbol{\theta}) = \frac{1}{2} (Q^+(s, a) - Q(s, a; \boldsymbol{\theta}))^2$ . Besides, deep neural network has made novel progress in the following aspects:

- Experience Replay [12]: The agent stores the past experience (i.e., the tuple  $e_t = \langle s_t, a_t, s'_t, R(s_t, a_t) \rangle$ ) at episode  $t$  into a dataset  $D_t = (e_1, \dots, e_t)$  and uniformly selects some (mini-batch) items from the dataset to update the Q-value neural network  $Q(s, a; \boldsymbol{\theta})$ .
- Network Cloning: The agent uses a separate network  $\hat{Q}$  to guide how to select an action  $a$  in state  $s$ , and the network  $\hat{Q}$  is replaced by  $Q$  every  $C$  episodes. Simulation results demonstrate that this network cloning enhances the learning stability [12].

Both experience replay and network cloning motivate to choose the off-policy  $Q$ -learning, since the sampling policy is only contingent on previously trained Q-value NN and the updating policy, which relies on the information from the new episodes, is irrespective of the sampling policy. On the other hand, the DQL agent could collect the information (i.e., state-action-reward pair) and train its policy in background. Also, the learned policy is stored in the neural networks and can be conveniently transferred among similar scenarios. In other words, the DQL could efficiently perform and timely make the resource allocation decision according to its already learned policy.

Finally, we illustrate the deep  $Q$ -learning in Fig. 1 and summarize the general steps in Algorithm 1.

### III. Resource Management for Network Slicing

Resource management is a permanent topic during the evolution of wireless communication. Intuitively, resource management for network slicing can be considered from several different perspectives.

- Radio Resource and Virtualized Network Functions: As depicted in Fig. 2, resource management for network slicing involves both radio access part and core network part with slightly different optimization goals. Due to the limited spectrum resource, the resource management for the radio access puts considerable efforts in allocating resource blocks (RBs) to one slice, so as to maintain acceptable SE while trying to bring appealing rate and small delay. The widely adopted optical transmission in core networks has shifted the optimization of core network to design common or dedicated virtualized network functions (VNFs), so as to appropriately forward the packets from one specific slice with minimal scheduling delay. By balancing the relative importance of resource utilization (e.g, SE) and QoE satisfaction ratio, the resource management problem could be formulated

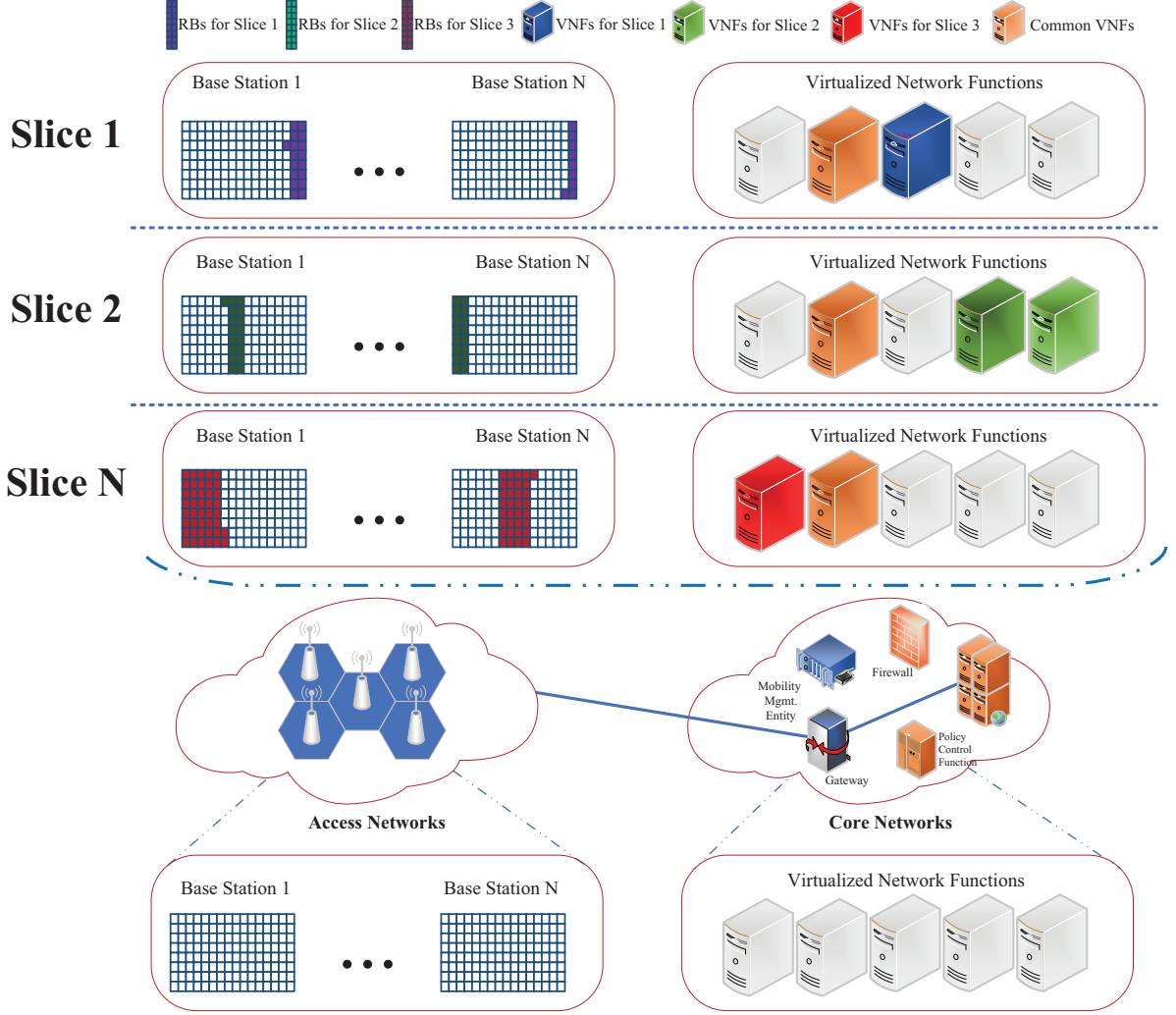


Fig. 2. An illustration of resource management for network slicing.

as  $R = \zeta \cdot \text{SE} + \beta \cdot \text{QoE}$ , where  $\zeta$  and  $\beta$  denotes the importance of SE and QoE.

- **Equal or Prioritized Scheduling:** As part of the control plane, IETF [18] has defined the common control network function (CCNF) to all or several slices. The CCNF includes the access and mobility management function (AMF) as well as the network slice selection function (NSSF), which is in charge of selecting core network slice instances. Hence, besides equally treating flows from different slices, the CCNF might differentiate flows. For example, flows from ultra-reliable low-latency communications (URLLC) service can be scheduled and provisioned in higher priority, so as to experience as little latency as possible. In this case, in order to balance the resource utilization (RU) and the waiting time (WT) of flows, the objective goal could be similarly written as a weighted summation of RU and WT.

Based on the aforementioned discussions, we can safely reach a conclusion that, the objective of resource management for network slicing should take account of several

variables and a weighted summation of these variables can be considered as the reward for the learning agent.

#### A. Radio Resource Slicing

In this part, we address how to apply DRL for radio resource slicing. Mathematically, given a list of existing slices  $1, \dots, N$  sharing the aggregated bandwidth  $W$  and having fluctuating demands  $\mathbf{d} = (d_1, \dots, d_N)$ , DQL tries to give a bandwidth sharing solution  $\mathbf{w} = (w_1, \dots, w_N)$ , so as to maximize the long-term reward expectation  $\mathbb{E}\{R(\mathbf{w}, \mathbf{d})\}$  where the notation  $\mathbb{E}(\cdot)$  denotes to take the expectation of the argument, that is,

$$\begin{aligned}
 & \arg_{\mathbf{w}} \max \mathbb{E}\{R(\mathbf{w}, \mathbf{d})\} \\
 & = \arg_{\mathbf{w}} \max \mathbb{E}\{\zeta \cdot \text{SE}(\mathbf{w}, \mathbf{d}) + \beta \cdot \text{QoE}(\mathbf{w}, \mathbf{d})\} \\
 & \text{s.t.: } \mathbf{w} = (w_1, \dots, w_N) \\
 & \quad w_1 + \dots + w_N = W \\
 & \quad \mathbf{d} = (d_1, \dots, d_N) \\
 & \quad d_i \sim \text{Certain Traffic Model}, \forall i \in [1, \dots, N]
 \end{aligned} \tag{4}$$

TABLE I  
A Brief Summary of Key Settings in DRL for Network Slicing Simulations

(a) The Mapping from Resource Management for Network Slicing to DRL			
	Radio Resource Slicing		Priority-based Core Network Slicing
State	The number of arrived packets in each slice within a specific time window		The priority and time-stamp of last arrived five flows in each service function chain (SFC)
Action	Allocated bandwidth to each slice		Allocated SFC for the flow at current time-stamp
Reward	Weighted sum of SE and QoE in 3 sliced bands		Weighted sum of average time in 3 SFCs
(b) Parameter settings for radio resource slicing			
	VoLTE	Video	URLLC
Bandwidth	10 MHz		
Scheduling	Round robin per slot (0.5 ms)		
Slice Band Adjustment (Q-Value Update)	1 second (2000 scheduling slots)		
Channel	Rayleigh fading		
User No. (100 in all)	46	46	8
Distribution of Inter-Arrival Time	Uniform [Min = 0, Max = 160ms]	Truncated Pareto [Exponential Para = 1.2, Mean = 6 ms, Max = 12.5 ms]	Exponential [Mean = 180 ms]
Distribution of Packet Size	Constant (40 Byte)	Truncated Pareto [Exponential Para = 1.2, Mean = 100 Byte, Max = 250 Byte]	Truncated Lognormal [Mean = 2 MB, Standard Deviation = 0.722 MB, Maximum =5 MB]
SLA: Rate	51 kbps	5 Mbps	10 Mbps
SLA: Latency	10 ms	10 ms	5 ms

Algorithm 1 The general steps of deep reinforcement learning.

input: An evaluation network  $Q$  with weights  $\theta$ ; a target network  $\hat{Q}$  with weights  $\hat{\theta} = \theta$ .

initialize: A replay memory dataset  $D$  with size of  $N$ ; the episode index  $t = 0$ .

- 1: repeat
- 2:   At episode  $t$ , the DQL agent observes the state  $s_t$ .
- 3:   The agent chooses action  $a_t$  with a probability  $\epsilon$  or selects  $a_t$  satisfying  $a_t = \arg \max_a Q(s_t, a; \theta)$ .
- 4:   After executing the action  $a_t$ , the agent observes the reward  $R(s_t, a_t)$  and a new state  $s_{t+1} = s'_t$  for the system.
- 5:   The agent stores the episode experience  $e_t = \langle s_t, a_t, s'_t, R(s_t, a_t) \rangle$  into  $D$ .
- 6:   The agent samples a minibatch of experiences from  $D$  and sets  $Q^+(s_t, a_t) = R(s_t, a_t) + \gamma \max_{a'} Q^+(s'_t, a')$ . In cases where episode terminates at  $t$ ,  $Q^+(s_t, a_t) = R(s_t, a_t)$ .
- 7:   The agent updates the weights  $\theta$  for the evaluation network by a gradient-based approach in (3).
- 8:   The agent clones the evaluation network  $Q$  to the target network  $\hat{Q}$  every  $C$  episodes by assigning the weights  $\hat{\theta}$  as  $\hat{\theta} = \theta$ .
- 9:   The episode index is updated by  $t \leftarrow t + 1$ .
- 10: until A predefined stopping condition (e.g., the gap between  $\theta$  and  $\hat{\theta}$ , the episode length, etc) is satisfied.

The key challenge to solve (4) lies in the volatile demand variations without having known a priori due to the traffic model. Hence, DQL is exactly the matching solution to solve the problem.

We evaluate the performance to adopt DQL to solve (4) by simulating a scenario containing one single BS with three types of services (i.e., VoIP, video, URLLC). There exist 100 registered subscribers randomly located within a 40 meter-radius circle surrounding the BS. These subscribers generate service models summarized in Table I(b). VoIP and video services exactly take the parameter settings of VoLTE and video streaming models, while URLLC service takes the parameter settings of FTP 2 model [19]. It can be observed from Table I(b), URLLC has less frequent packets compared with the others, while VoLTE requires the smallest bandwidth for its packets.

We consider DQL by using the mapping in Table I(a) to optimize the weighted summation of system SE and slice QoE. Specifically, we perform round-robin scheduling method within each slice at the granularity of 0.5 ms. In other words, we sequentially allocate the bandwidth of each slice to the active users within each slice every 0.5 ms. Besides, we adjust the bandwidth allocation to each slice per second. Therefore, the DQL agent updates its Q-value neural network every second. We compare the simulation results with the following three methods, so as to explain the importance of DQL.

- Demand-prediction based method: The method tries to estimate the possible demand by using

long short-term memory (LSTM) to predict the number of active users requesting VoIP, video and URLLC respectively. Afterwards, the bandwidth is allocated by two ways: (1) DP-No allocates the whole bandwidth to each slice proportional to the number of predicted packets. In particular, assuming that the total bandwidth is  $B$  and the predicted number of packets for VoIP, video and URLLC is  $N_{\text{VoIP}}$ ,  $N_{\text{Video}}$  and  $N_{\text{URLLC}}$ , the allocated bandwidth to these three slices (i.e., VoIP, video and URLLC) is  $\frac{B \cdot N_{\text{VoIP}}}{N_{\text{VoIP}} + N_{\text{Video}} + N_{\text{URLLC}}}$ ,  $\frac{B \cdot N_{\text{Video}}}{N_{\text{VoIP}} + N_{\text{Video}} + N_{\text{URLLC}}}$ ,  $\frac{B \cdot N_{\text{URLLC}}}{N_{\text{VoIP}} + N_{\text{Video}} + N_{\text{URLLC}}}$ , respectively. (2) DP-BW performs the allocation by multiplying the number of predicted packets by the least required rate in Table I(b) and then computing the proportion. In this regard, assuming that the required rate for the three slices is  $R_{\text{VoIP}}$ ,  $R_{\text{Video}}$  and  $R_{\text{URLLC}}$ , the allocated bandwidth to VoIP, video and URLLC is  $\frac{B N_{\text{VoIP}} R_{\text{VoIP}}}{N_{\text{VoIP}} R_{\text{VoIP}} + N_{\text{Video}} R_{\text{Video}} + N_{\text{URLLC}} R_{\text{URLLC}}}$ ,  $\frac{B N_{\text{Video}} R_{\text{Video}}}{N_{\text{VoIP}} R_{\text{VoIP}} + N_{\text{Video}} R_{\text{Video}} + N_{\text{URLLC}} R_{\text{URLLC}}}$ ,  $\frac{B N_{\text{URLLC}} R_{\text{URLLC}}}{N_{\text{VoIP}} R_{\text{VoIP}} + N_{\text{Video}} R_{\text{Video}} + N_{\text{URLLC}} R_{\text{URLLC}}}$ , respectively. Round-robin is conducted within each slice.

- Hard slicing: Hard slicing means that each service slice is always allocated  $\frac{1}{3}$  of the whole bandwidth, since there exists 3 types of service in total. Again, round-robin is conducted within each slice.
- No slicing: Irrespective of the related SLA, all users are scheduled equally. Round-robin is conducted within all users.

We primarily consider the downlink case and adopt system SE and QoE satisfaction ratio as the evaluation metrics. In particular, the system SE is computed as the number of bits transmitted per second per unit bandwidth, where the rate from the BS to users is derived based on Shannon capacity formula. Therefore, if part of the bandwidth has been allocated to one slice but the slice has no service activities at one slot, such part of bandwidth has been wasted, thus degrading the system SE. QoE satisfaction ratio is obtained by dividing the number of completely transmitted packets satisfying rate and latency requirement by the total number of arrived packets.

Fig. 3 presents the learning process of DQL<sup>1</sup> in radio resource management. In particular, Fig. 3(a)~3(f) give the initial performance of DQL when the QoE weight is 5000 and the SE weight is 0.1. Fig. 3(g)~3(l) provide the performance during the last 50 of 50000 learning updates. From these sub-figures, it can be observed that DQL could not well learn the user activities at the very beginning and the allocated bandwidth fluctuates heavily. But after nearly 50000 updates, DQL has gained better knowledge over user activities and yielded a state bandwidth-allocation strategy. Besides, Fig. 3(m) and Fig. 3(n) show the variations of SE and QoE along with each learning epoch. From both subfigures, a larger QoE weight produces policies with superior QoE performance while bringing certain loss in the system SE performance.

<sup>1</sup>Notably,  $\gamma$  is set as 0.9.

Fig. 4 provides a detailed performance comparison among the candidate techniques, where the results for DQL are obtained after 50000 learning updates. Fig. 4(a)~4(f) gives the percentage of total bandwidth allocated to each slice using the pie charts and highlights the QoE satisfaction ratio by surrounding text. From Fig. 4(a)~4(b), a reduction in transmission antennas from 64 to 16, which implies a decrease in network capability and an increase in potential collisions across slices, leads to a re-allocation of network bandwidth inclined to the bandwidth-consuming yet activity-limited URLLC slice. Also, it can be observed from Fig. 4(f), when the downlink transmission uses 64 antennas, “no slicing” performs the best, since the transmission capability is sufficient and the scheduling period is 0.5 ms while the bandwidth allocated to each slice is adjusted per second and thus slower to catch the demand variations. When the number of downlink antenna turns to 32, the DQL-driven scheme produces 81% QoE satisfaction ratio for URLLC, while “no slicing” and “hard slicing” schemes only provision 15% and 41% satisfied URLLC packets, respectively. Notably, applying DQL mainly leads to the QoE gain of URLLC. The reason lies in that as summarized in Table I(b), the distribution of packet size for URLLC follows a truncated lognormal distribution with the mean value of 2 MByte, which is far larger than those of VoLTE and Video services. Given the larger transmission volume and strictly lower latency requirement, it is far more difficult to satisfy the QoE of URLLC. In this case, it is still satisfactory that DQL outperforms the other competitive schemes to render higher QoE gain of URLLC at a slight cost of spectrum efficiency (SE). Meanwhile, Fig. 4(d) and Fig. 4(e) demonstrate the allocation results for the demand-prediction based schemes and show significantly inferior performance, since Fig. 3(a)~3(c) and Fig. 3(g)~3(i) show the number of video packets dominates the transmission and simple packet-number based prediction could not capture the complicated relationship between demand and QoE. On the other hand, Fig. 4(g) illustrates that this QoE advantage of DQL comes at the cost of a decrease in SE. Recalling the definition of the reward in DQL, if we decrease the QoE weight from 5000 to 1, DQL could learn another bandwidth allocation policy (in Fig. 4(c)) yielding a larger SE yet a lower QoE. Fig. 4(g) ~ 4(j) further summarize the performance comparison in terms of SE or QoE satisfaction ratios, where the vertical errorbars show the standard derivation. These subfigures validate the DQL’s flexibility and advantage in resource-limited scenarios to ensure the QoE per user.

## B. Priority-based Scheduling in Common VNFs

Section III-A has discussed how to apply DRL in radio resource slicing. Similarly, if we virtualize the computation resources as VNFs for each slice, the problem to allocate computation resources to each VNF could be solved similar to the radio resource slicing case. Therefore, in this part, we talk about another important issue, that

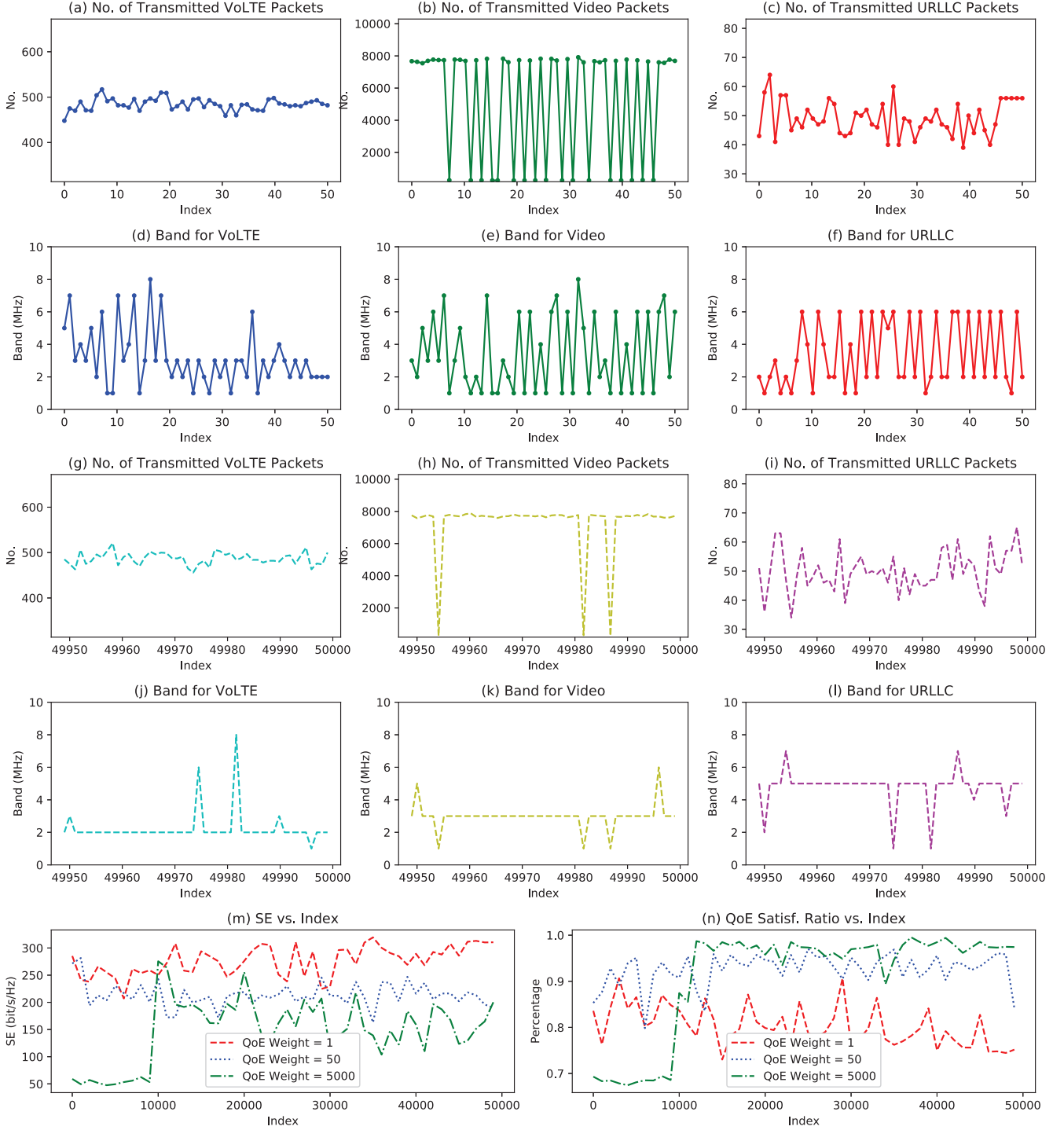


Fig. 3. The performance of DQL for radio resource slicing w.r.t. the learning steps (QoE Weight = 5000).

is, priority-based core network slicing for common VNFs. Specifically, we simulate a scenario where there exists 3 service function chains (SFCs) possessing the same basic capability but working at the expenditure of different computation processing units (CPUs) and yields different provisioning results (e.g., waiting time). Also, based on the commercial value or related SLA, flows could be classified into 3 categories (e.g., Category A, B, and C) with decreasing priority from Category A to Category C,

and a priority-based scheduling rule is defined as that SFC I prioritizes Category A flows over the others, while SFC II equally treats Category A and B users but serves Category C flows with lower priority. SFC III treats all flows equally. Besides, SFCs process flows with equal priority according to the arrival time. The eventually utilized CPUs of each SFC depend on the number of its processed flows. Besides, SFC I, II and III cost 2, 1.5, and 1 CPU(s), but incur 10, 15, and 20 ms regardless of the flow size, respectively.





Fig. 4. The performance comparison among different schemes for radio resource slicing.

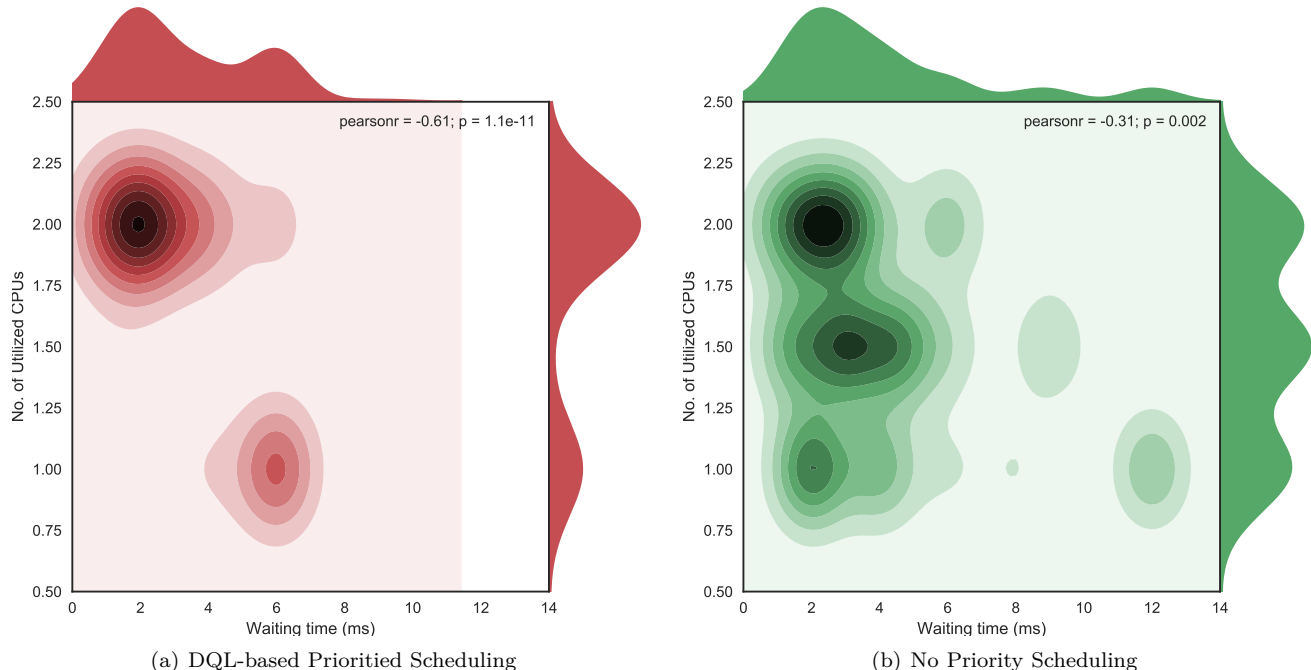


Fig. 5. Performance comparison between DQL-based priority scheduling and no priority scheduling for core network slicing.

Hence, subject to the limited number of CPUs, flows for each type will be scheduled to an appropriate SFC, so as to incur acceptable waiting time. Therefore, the scheduling of flows should match and learn the arrival of flows in three categories, and DQL is considered as a promising solution.

Similarly, it is critical to design an appropriate mapping of DRL elements to this slicing issue. As Table I(a) implies, we use a mapping slightly different from that for radio resource slicing, so as to manifest the flexibility of DQL. In particular, we abstract the state of DQL as a summary of the category and arrival time of last 5 flows and the category of the newly arrived flow, while the reward is defined as the weighted summation of processing and queue time of this flow, where a larger weight in this summation is adopted to reflect the importance of flows with higher priority. Also, we first pre-train its NN by emulating some flows with lognormal distributed inter-arrival time from the three categories' users.

We compare the DQL scheme with an intuitive “no priority” solution, which allocate the flow to the SFC yielding minimum waiting time. Fig. 5 provides the related performance by randomly generating 10000 flows and provisioning accordingly, where the vertical and horizontal axes represent the number of utilized CPUs and the waiting time of flows respectively. Specifically, the bi-dimensional shading color reflects the number of flows corresponding to the specific waiting time and utilized CPUs. In particular, the darker color implies the larger number. Compared with the “no priority” solution, the DQL-empowered slicing results provision flows with smaller average waiting time (i.e., 10.5% lower than “no priority”)

and significantly more sufficient CPU usage (i.e., 27.9% larger than “no priority”). In other words, DQL could support alternative solutions to exploit the computing resources and reduce the waiting time by first serving the users with higher commercial value.

#### IV. Conclusion and Future Directions

From the discussions in this article, we found that matching the allocated resource to slices with the users' activity demand will be the most critical challenge for effectively realizing network slicing, while DRL could be a promising solution. Starting with the introduction of fundamental concept for DQL, one typical type of DRL, we explained the working mechanism and application motivation of DQL to solve this problem. We further demonstrated the advantage of DQL in managing this demand-aware resource allocation in two typical slicing scenarios including radio resource slicing and priority-based core network slicing through extensive simulations. Our results showed that compared with the demand prediction-based and some other intuitive solutions, DQL could implicitly incorporate more deep relationship between demand (i.e., user activities) and supply (i.e., resource allocation) in resource-constrained scenarios, and enhance the effectiveness and agility for network slicing. Finally, in order to fulfill the application of DQL in a broader sense, we pointed out some noteworthy issues. We believe DRL could play a crucial role in network slicing in the future.

However, network slicing involves many aspects and a successful application of DQL needs some careful considerations: (a) Slice admission control on incoming requests

for new slices: the success of network slicing implies a dynamic and agile slice management scheme. Therefore, if requests for new slices emerge, how to apply DQL is also an interesting problem since the defined state and action space requires to adapt to the changes in the “slice” space. (b) Abstraction of states and actions: Section III has provided two ways to abstract state and action. Both methods sound practical in the related scenarios and reflect the flexibility of DQL. Hence, for new scenarios, it becomes an important issue to choose appropriate abstraction of states and actions, so as to better model the problem and save the learning cost. Up to date, it remains an open question on how to give some abstraction guidelines. (c) Latency and accuracy to retrieve rewards: The simulations in Section III has assumed the instantaneous and accurate acquirement of rewards for a state-action pair. But, such an assumption no longer holds in practical complex wireless environment, since it takes time for user equipment to report the information and the network may not successfully receive the feedback. Also, similar to the case for state and action, the abstraction of reward might be difficult and the defined reward should be as simple as possible. (d) Policy learning cost: The time-varying nature of wireless channel and user activities requires a fast policy-learning scheme. However, the current cost of policy training still lacks the necessary learning speed. For example, our pre-training for the priority-based network slicing policy takes two days in an Intel Core i7-4712MQ processor to converge the Q-value function. Though GPU could speedup the training process, the learning cost is still heavy. Therefore, there are still a lot of interesting questions to be addressed.

### Acknowledgment

The authors would like to express their sincere gratitude to Chen Yu and Yuxiu Hua of Zhejiang University for the valuable discussions to implement part of simulation codes.

### References

- [1] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, “Network slices toward 5G communications: Slicing the LTE network,” *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 146–154, 2017.
- [2] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, “NFV and SDN – Key technology enablers for 5G networks,” *IEEE J. Sel. Area. Comm.*, vol. 35, no. 11, pp. 2468–2478, Nov. 2017.
- [3] 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 INFOCOM 2017*, Atlanta, GA, USA, May 2017.
- [4] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, and H. Zhang, “Intelligent 5G: When cellular networks meet artificial intelligence,” *IEEE Wireless Commun.*, vol. 5, no. 24, pp. 175 – 183, Oct. 2017.
- [5] X. Zhou, R. Li, T. Chen, and H. Zhang, “Network slicing as a service: Enable industries own software-defined cellular networks,” *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 146–153, Jul. 2016.
- [6] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, and R. Jain, “Network slicing for 5G: Challenges and opportunities,” *IEEE Internet Comput.*, vol. 21, no. 5, pp. 20–27, 2017.

- [7] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, “The algorithmic aspects of network slicing,” *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 112–119, 2017.
- [8] N. Zhang, Y. F. Liu, H. Farmanbar, T. H. Chang, M. Hong, and Z. Q. Luo, “Network slicing for service-oriented networks under resource constraints,” *IEEE J. Sel. Area. Comm.*, vol. 35, no. 11, pp. 2512–2521, Nov. 2017.
- [9] R. Yu, G. Xue, and X. Zhang, “QoS-aware and reliable traffic steering for service function chaining in mobile networks,” *IEEE J. Sel. Area. Comm.*, vol. 35, no. 11, pp. 2522–2531, Nov. 2017.
- [10] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge University Press, 1998. [Online]. Available: <http://webdocs.cs.ualberta.ca/~sutton/books/ebook/>
- [11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: <https://www.nature.com/articles/nature16961>
- [12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: <http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html>
- [13] Y. S. Nasir and D. Guo, “Deep reinforcement learning for distributed dynamic power allocation in wireless networks,” *arxiv, p. cs.IT 1808.00490*, Aug. 2018.
- [14] J. Liu, B. Krishnamachari, S. Zhou, and Z. Niu, “Deepnap: Data-driven base station sleeping operations through deep reinforcement learning,” *IEEE Internet Things J.*, pp. 1–1, 2018.
- [15] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, “A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs,” in *Proc. IEEE ICC 2017*, Paris, France, May 2017.
- [16] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, “Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach,” *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.
- [17] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989.
- [18] X. de Foy and A. Rahman, “Network Slicing – 3GPP Use Case,” Network Working Group, IETF, Tech. Rep., Oct. 2017. [Online]. Available: <https://tools.ietf.org/id/draft-defoy-netslices-3GPP-network-slicing-02.html>
- [19] NGMN, “NGMN radio access performance evaluation methodology,” [Online]. Available: [https://www.ngmn.org/publications/all-downloads.html?tx\\_news\\_pi1%5Bnews%5D=604&cHash=94dc3082a0b35f5ec64dbf9e33d2298a](https://www.ngmn.org/publications/all-downloads.html?tx_news_pi1%5Bnews%5D=604&cHash=94dc3082a0b35f5ec64dbf9e33d2298a)

### Biographies

Rongpeng Li is now an assistant professor in College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou China. He received his Ph.D and B.E. from Zhejiang University, Hangzhou, China and Xidian University, Xi’an, China in June 2015 and June 2010 respectively, both as “Excellent Graduates”. Dr. Li was a research engineer in Wireless Communication Laboratory, Huawei Technologies Co. Ltd., Shanghai, China from August 2015 to September 2016. He returned to academia in November 2016 as a postdoctoral researcher in College of Computer Science and Technologies, Zhejiang University, Hangzhou, China, which is sponsored by the National Postdoctoral Program for Innovative Talents. His research interests currently focus on Reinforcement Learning, Data Mining and all broad-sense network problems(e.g., resource management, security, etc) and he has authored/coauthored

several papers in the related fields. He serves as an Editor of China Communications.

Zhifeng Zhao is an Associate Professor at the Department of Information Science and Electronic Engineering, Zhejiang University, China. He received the Ph.D. degree in Communication and Information System from the PLA University of Science and Technology, Nanjing, China, in 2002. Prior to that, he received the Master degree of Communication and Information System in 1999 and Bachelor degree of Computer Science in 1996, from the PLA University of Science and Technology, respectively. From September 2002 to December 2004, he acted as a postdoctoral researcher at the Zhejiang University, where his researches were focused on multimedia NGN (next-generation networks) and soft-switch technology for energy efficiency. From January 2005 to August 2006, he acted as a senior researcher at the PLA University of Science and Technology, Nanjing, China, where he performed research and development on advanced energy-efficient wireless router, Ad Hoc network simulator and cognitive mesh networking test-bed. His research area includes cognitive radio, wireless multi-hop networks (Ad Hoc, Mesh, WSN, etc.), wireless multimedia network and Green Communications.

Qi Sun received her Ph.D. degree in information and communication engineering from Beijing University of Posts and Telecommunications in 2014. After graduation, she joined the Green Communication Research Center of the China Mobile Research Institute. Her research interest focuses on 5G communications, including new waveforms, non-orthogonal multiple access, massive MIMO, full duplex.

Chih-Lin I is CMCC Chief Scientist of Wireless Technologies, launched 5G R&D in 2011, and leads C-RAN, Green and Soft initiatives. Chih-Lin received IEEE Trans. COM Stephen Rice Best Paper Award, and IEEE ComSoc Industrial Innovation Award. She was on IEEE ComSoc Board, GreenTouch Executive Board, WWRP Steering Board, M&C Board Chair, and WCNC SC Founding Chair. She is on IEEE ComSoc SPC and EDB, ETSI/NFV NOC, and Singapore NRF SAB.

Chenyang Yang received her Ph.D. degrees in Electrical Engineering from Beihang University (formerly Beijing University of Aeronautics and Astronautics, BUAA), China, in 1997. She has been a full professor with the School of Electronics and Information Engineering, BUAA since 1999. She has published over 200 papers and filed over 80 patents in the fields of energy efficient transmission, URLLC, wireless local caching, CoMP, interference management, cognitive radio, and relay, etc. She was supported by the 1st Teaching and Research Award Program for Outstanding Young Teachers of Higher Education Institutions by Ministry of Education of China. She was the chair of Beijing chapter of IEEE Communications Society during 2008-2012, and the MDC chair of APB of IEEE Communications Society during 2011-2013. She has served as TPC Member, TPC co-chair or Track co-chair for IEEE conferences. She has ever served as an associate editor for IEEE Trans. on Wireless Communication, guest editor for IEEE Journal of Selected Topics in Signal Processing and IEEE Journal of Selected Areas in Communications. Her recent research interests lie in mobile AI, wireless caching, and URLLC.

Xianfu Chen received his Ph.D. degree in Signal and Information Processing, from the Department of Information Science and Electronic Engineering at Zhejiang University, Hangzhou, China, in March 2012. He is currently a Senior Scientist with the VTT Technical Research Centre of Finland Ltd, Oulu, Finland. His research interests cover various aspects of wireless communications and networking, with emphasis on network virtualization, software-defined radio access networks, green communications, centralized and decentralized resource allocation, and the application of machine learning to cognitive radio networks. He is an IEEE member.

Minjian Zhao received the M.Sc. and Ph.D. degrees in communication and information systems from Zhejiang University, Hangzhou, China, in 2000 and 2003, respectively. He is currently a Professor with the College of Information Science and Electronic Engineering, Zhejiang University. His research interests include modulation theory, channel estimation and equalization, and signal processing for wireless communications.

Honggang Zhang is currently a Full Professor with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. He was an Honorary Visiting Professor with the University of York, U.K and an International Chair Professor of Excellence for Université Européenne de Bretagne and Supélec, France. He was the Co-Author and an Editor of two books "Cognitive Communications-Distributed Artificial Intelligence (DAI), Regulatory Policy and Economics, Implementation (John Wiley & Sons)" and "Green Communications: Theoretical Fundamentals, Algorithms and Applications (CRC Press)", respectively. He is also active in the research on green communications and was the leading Guest Editor of the IEEE Communications Magazine special issues on Green Communications. He is taking the role of an Associate Editor-in-Chief of China Communications and the Series Editors of the IEEE Communications Magazine for its Green Communications and Computing Networks Series. He served as the Chair of the Technical Committee on Cognitive Networks of the IEEE Communications Society from 2011 to 2012.