Optimal and Fast Real-Time Resource Slicing With Deep Dueling Neural Networks

Nguyen Van Huynh, *Student Member, IEEE*, Dinh Thai Hoang, *Member, IEEE*, Diep N. Nguyen, *Senior Member, IEEE*, and Eryk Dutkiewicz, *Senior Member, IEEE*

***Abstract*— Effective network slicing requires an infrastructure/network provider to deal with the uncertain demands and real-time dynamics of the network resource requests. Another challenge is the combinatorial optimization of numerous resources, e.g., radio, computing, and storage. This paper develops an optimal and fast real-time resource slicing framework that maximizes the long-term return of the network provider while taking into account the uncertainty of resource demands from tenants. Specifically, we first propose a novel system model that enables the network provider to effectively slice various types of resources to different classes of users under separate virtual slices. We then capture the real-time arrival of slice requests by a semi-Markov decision process. To obtain the optimal resource allocation policy under the dynamics of slicing requests, e.g., uncertain service time and resource demands, a Q-learning algorithm is often adopted in the literature. However, such an algorithm is notorious for its slow convergence, especially for problems with large state/action spaces. This makes Q-learning practically inapplicable to our case, in which multiple resources are simultaneously optimized. To tackle it, we propose a novel network slicing approach with an advanced deep learning architecture, called deep dueling, that attains the optimal average reward much faster than the conventional Q-learning algorithm. This property is especially desirable to cope with the real-time resource requests and the dynamic demands of the users. Extensive simulations show that the proposed framework yields up to 40% higher long-term average return while being few thousand times faster, compared with the state-of-the-art network slicing approaches.**

***Index Terms*— Network slicing, MDP, Q-learning, deep reinforcement learning, deep dueling, resource allocation.**

I. Introduction

The latest Cisco Visual Networking Index forecast a sevenfold increase in global mobile data traffic from 2016 to 2021, with 5G traffic expected to start having a relatively small but measurable impact on mobile growth starting in 2020. Machine-to-machine (M2M) connections will represent 29% (3.3 billion) of total mobile connections - up from 5% (780 million) in 2016 [1]. M2M has been the fastest growing mobile connection type as global IoT applications continue to gain traction in consumer and business environments. However, legacy mobile networks are mostly designed to provide services for mobile broadband users and are unable to meet adjustable parameters like priority and quality of service (QoS) for emerging services. Therefore, mobile operators may find difficulties in getting deeply into these emerging vertical services with different service requirements for network design and development. In order to enhance operators' products for vertical enterprises and provide service customization for emerging massive connections, as well as to give more control to enterprises and mobile virtual network operators, the concept of network slicing has been recently introduced to allow the independent usage of a part of network resources by a group of mobile terminals with special requirements.

Network slicing was introduced by Next Generation Mobile Networks Alliance [2], and it has quickly received a lot of attention from both academia and industry. In general, network slicing is a novel virtualization paradigm that enables multiple logical networks, i.e., slices, to be created according to specific technical or commercial demands and simultaneously run on top of the physical network infrastructure. The core idea of the network slicing is using software-defined networking (SDN) and network functions virtualization (NFV) technologies for controlling network operations and virtualizing the physical infrastructure. In particular, SDN provides a separation between the network control and data planes, improving the flexibility of network function management and efficiency of data transfer. Meanwhile, NFV allows various network functions to be virtualized, i.e., in virtual machines. As a result, the functions can be moved to different locations, and the corresponding virtual machines can be migrated to run on commoditized hardware dynamically depending on the demands and requirements [3], [4].

The key benefit of network slicing is to enable providers to offer network services on an as-a-service basis which enhances operational efficiency while reducing time-to-market for new services [5]. However, to achieve this goal, there are two main challenges in managing network resources. First, many emerging services require not only radio resources for communications but also computing and storage resources to meet requirements about quality-of-service (QoS) of users. For example, IoT services often require simultaneously radio, computing, and storage resources to transmit data and pre-processing intensive tasks because of the resource

Manuscript received July 25, 2018; revised December 20, 2018; accepted February 28, 2019. Date of publication March 12, 2019; date of current version May 15, 2019. *(Corresponding author: Dinh Thai Hoang.)*

The authors are with the School of Electrical and Data Engineering, University of Technology Sydney, Broadway, NSW 2007, Australia (e-mail: huynh.nguyenvan@student.uts.edu.au; hoang.dinh@uts.edu.au; diep.nguyen@uts.edu.au; eryk.dutkiewicz@uts.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2904371

constraints on IoT devices. Additionally, the network provider may posses multiple data centers with several servers containing diverse resources, e.g., computing and storage, connected together [49]. Thus, how to concurrently manage multiple interconnected resources is an emerging challenge for the network provider. Second, due to the dynamic demands of services, e.g., the frequency of requests and occupation time, and the limitation of resources, how to dynamically allocate resources in a real-time manner to maximize the long term revenue is another challenge of the network provider. As a result, there are some recent works proposing solutions to address these issues.

A. Related Work

A number of research works have been introduced recently to address the network slicing resource allocation problem for the network provider [6]–[16]. In particular, Jiang *et al.* [6] and Soliman and Leon-Garcia [7] developed a two-tier admission control and resource allocation model to answer two fundamental questions, i.e., whether a slice request is accepted and how much radio resource is allocated to the accepted slice. To address this problem, Jiang *et al.* [6] used an extensive searching method to achieve the globally optimal resource allocation solution for the network provider. However, this searching method cannot be applied to complex systems with a large number of resources. To address this problem, a heuristic scheme with three main steps was introduced in [7] to effectively allocate resources to the users. Yet this heuristic scheme cannot guarantee to achieve the optimal solution for the network provider. In addition, both network slicing resource allocation solutions proposed in [6] and [7] are heuristic methods with only radio resources taken into consideration. Thus, these solutions may not be appropriate to implement in dynamic network slicing resource allocation systems with a wide range of resource demands and services.

To deal with the dynamic of services, e.g., users' resource demands and their occupation time, Sciancalepore *et al.* [13] proposed a model to predict the future demands of slices, thereby maximizing the system resource utilization for the provider. The key idea of this approach is to use the Holt-Winters approach [17] to predict network slices' demands through tracking the traffic usage of users in the past. However, the accuracy of this prediction depends largely on the heavy-tailed distribution functions along with many control parameters such as scale factor, least-action trip planning, and potential gain. Furthermore, this approach only considers the short-term reward for the provider, and thus the long-term profit may not be able to obtain. Therefore, Bega *et al.* [14] and Aijaz [15], [16] proposed reinforcement learning algorithms to address these problems. Among dynamic resource allocation methods, reinforcement learning has been considering to be the most effective way to maximize the long-term reward for dynamic systems as this method allows the network controller to adjust its actions in a real-time manner to obtain the optimal policy through the trial-and-error learning process [18]. However, this method often takes a long period to converge to the optimal solution, especially for a large-scale system.

In all aforementioned work, the authors considered optimizing only radio resources, while other resources are completely ignored. However, as stated in [2], [19], and [20], a typical network slice is composed of three main components, i.e., radio, computing, and storage. Consequently, considering only radio resources when orchestrating slices may be not able to achieve the optimal solution. Specifically, in Fig. 6, we show an illustration to demonstrate the inefficiency of optimizing only radio resources. Therefore, in this paper, we introduce a Semi-Markov decision processes (SMDP) framework [21] which allows the network provider to effectively allocate all three resources, i.e., radio, computing, and storage, to the users in a real-time manner. However, when we jointly consider combinatorial resources, i.e., radio, storage, and computing resources, together with the uncertainty of demands, the optimization problem becomes very complex as we need to simultaneously deal with a very large state space with multi-dimension and real-time dynamic decisions. Thus, we propose a novel network slicing framework with an advanced deep learning architecture using two streams of fully connected hidden layers, i.e., deep dueling neural network [42], combined with Q-learning method to effectively address this problem. Our preliminary simulation results [22] show that the proposed approach not only can effectively deal with the dynamic of the system but also significantly improve the system performance compared with all other current network slicing resource allocation approaches. It is worth noting that the VNF placement, routing, and connectivity resource allocation problems have been well investigated in the literature. Hence, in this paper, we focus on dealing with the uncertainty, dynamics, and heterogeneity of slice requests. Note that, our system model can be straightforwardly extended to the case with diverse connectivity among servers and data centers by accommodating additional states to the system state space. Additionally, our proposed framework can handle very well a large state space (one of our key contributions in this paper).

B. Main Contributions

The main contributions of this paper are as follows:

- We develop a dynamic network resource management model based on semi-Markov decision process framework which allows the network provider to jointly allocate computing, storage, and radio resources to different slice requests in a real-time manner and maximize the long-term reward under a number of available resources.
- To find the optimal policy under the uncertainty of slice service demands, we deploy the Q-learning algorithm which can achieve the optimal solution through reinforcement learning processes. However, the Q-learning algorithm may not be able to effectively achieve the optimal policy due to the curse-of-dimensionality problem when we jointly optimize multiple resources concurrently. Thus, we develop a deep double Q-learning approach which utilizes the advantage of a neural network to train the learning process of the Q-learning algorithm,

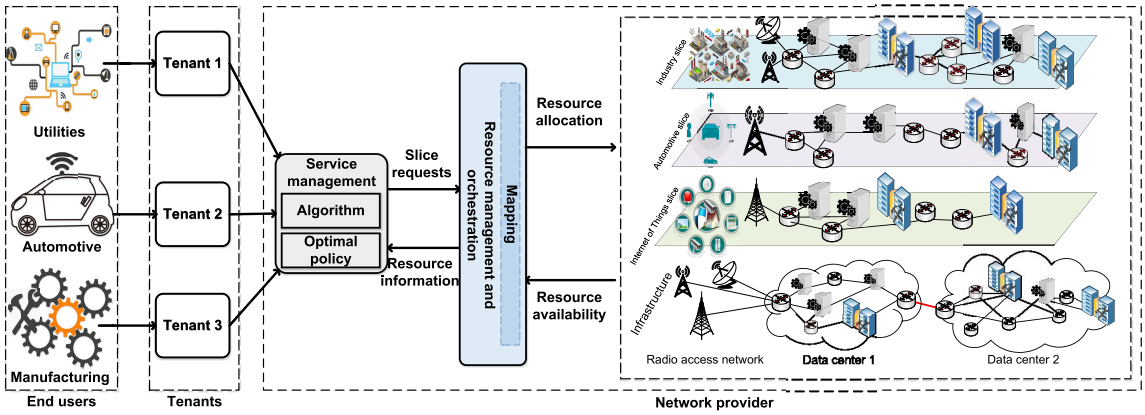


Fig. 1. Network resource slicing system model.

thereby attaining much better performance than that of the Q-learning algorithm.

- To further enhance the performance of the system, we propose the novel network slicing approach with the deep dueling neural network architecture [42], which can outperform all other current reinforcement learning techniques in managing network slicing. The key idea of the deep dueling is using two streams of fully connected hidden layers to concurrently train the learning process of the Q-learning algorithm, thereby improving the training process and achieving an outstanding performance for the system.
- Finally, we perform extensive simulations with the aim of not only demonstrating the efficiency of proposed solutions in comparison with other conventional methods but also providing insightful analytical results for the implementation of the system. Importantly, through simulation results, we demonstrate that our proposed framework can improve the performance of the system up to 40% compared with other current approaches.

The rest of the paper is organized as follows. Section II and Section III describe the system model and the problem formulation, respectively. Section IV introduces the Q-learning algorithm. Further on, we present the deep double Q-learning and deep dueling algorithms in Section V. Evaluation results are then discussed in Section VI. Finally, conclusions and future works are given in Section VII.

II. System Model

In Fig. 1, we consider a general network slicing model with three major parties [13], [14], [20]:

- *Network Provider:* is the owner of the network infrastructure who provides resource slices including radio, computing, and storage, to the tenants.
- *Tenants:* request and lease resource slices to meet service demands of their subscribers.
- *End Users:* run their applications on the slices of the above subscribed tenants.

We consider three tenants corresponding to three popular classes of services, i.e., utilities, automotive, and manufacturing, as shown in Fig. 1. Each class of service possesses several specific features regarding its functional, behavioral perspective, and requirements. For example, a vehicle may need an ultra-reliable slice for telemetry assisted driving [19]. For slices requested from industry, security, resilience, and reliability of services are of higher priority [32], [33]. Thus, when a tenant sends a network slice request to the network provider, the tenant will specify resources requested and additional service requirements, e.g., security and reliability (defined in the slice blueprint). As a result, tenants may pay different prices for their requests, depending on their service demands. Upon receiving a slice request, the service management component (in Fig. 1) analyzes the requirements and makes a decision to accept or reject the request based on its optimal policy.

The service management block consists of two components: (i) the optimal policy and (ii) the algorithm. When a slice request arrives at the system, the optimal policy component will make a decision, i.e., accept or reject, based on the (current) optimal policy obtained by the algorithm component. As the decision can be made immediately, the decision latency is virtually zero. For the algorithm component, the optimal policy is calculated and updated periodically. It is worth noting that our algorithm observes the results after performing the decision, and uses the observations together with the characteristics of slice requests for its training process. By doing so, our algorithm can learn from previous experiences and is able to deal with the uncertainty of slice requests. If the request is accepted, the service management will transfer the slice request to the resource management and orchestration (RMO) block to allocate resources. Once a slice request is accepted, the network provider will receive an immediate reward (an amount of money) paid by the tenant for granted resources and services.

In practice, the network provider may possess multiple data centers for network slicing services. Each data center contains a set of servers with diverse resources, e.g., computing and storage, which are used to support VNFs services. Servers in the data center are connected together, and the data centers are connected via backhaul links. Then, the network slicing and resource allocation processes to each slice are taken place as follows.

- A slice request is associated with the *network slice blueprint* (i.e., a template) that describes the structure, configuration, and workflow for instantiating and

controlling the network slice instance for the service during its life cycle [20], [23]–[25]. The service/slice instance includes a set of network functions and resources to meet the end-to-end service requirements.

- When a slice request arrives at the system, the orchestrator will interpret the blueprint [23]–[25]. In particular, all information of the infrastructure such as (i) NFV services provided by servers, (ii) resources availability at servers, and (iii) the connectivities among servers are checked.
- Based on the aforementioned information, the orchestrator will find the optimal servers and links to place VNFs to meet the required end-to-end services of the slice (i.e., VNF placement procedure).
- During the life cycle of the slice, the orchestrator can change the allocated computing and storage resources by using the scaling-in and scaling-out mechanisms. In addition, the connectivities among VNFs and their locations can be changed when there is no sufficient resources or the behavior of the slice is changed, e.g., update, migrate, or terminate the network slice.

Note that the VNF placement, routing, and connectivity resource allocation problems have been well investigated in the literature, e.g., [26]–[31]. For example, in [31], the AAP algorithm is introduced to admit and route connection requests by finding possible paths satisfying a cost criteria. Instead of focusing on VNF placement, routing, and connectivity resource allocation problems, in this paper, we mainly focus on dealing with the uncertainty, dynamics, and heterogeneity of slice requests. Thus, we consider a simplified yet practical model and propose the novel framework using the deep dueling neural network architecture [42] to address the aforementioned problems, which are the main aims and key contributions of this work. Specifically, we assume that there are C classes of slices, denoted by $\mathcal{C} = \{1, \ldots, c, \ldots, C\}$. Each slice from class c requires r_c^{re}, ω_c^{re}, and δ_c^{re} units of radio, computing, and storage resources, respectively. If a slice request from class c is accepted, the provider will receive an immediate reward r_c. The maximum radio, computing, and storage resources of the network provider are denoted by Θ, Ω, and Δ units, respectively. Let n_c denote the number of slices from class c being simultaneously run/served in the system. At any time, the following resource constraints guarantee that the allocated resources do not exceed the available resources of the infrastructure:

$$\Theta \geq \sum_{c=1}^{C} r_c^{re} n_c, \quad \Omega \geq \sum_{c=1}^{C} \omega_c^{re} n_c, \text{ and } \Delta \geq \sum_{c=1}^{C} \delta_c^{re} n_c. \tag{1}$$

III. Problem Formulation

To maximize the long-term return for the provider while accounting for the real-time arrivals of slice requests, we recruit the semi-Markov decision process (SMDP) [21]. An SMDP is defined by a tuple $< t_i, \mathcal{S}, \mathcal{A}, \mathcal{L}, r >$, where t_i is a decision epoch, $\mathcal{S}$ is the system's state space, $\mathcal{A}$ is the action space, $\mathcal{L}$ captures the state transition probabilities and the state sojourn time, and r is the reward function. Unlike discrete Markov decision processes where decisions are made in every time slots, in an SMDP, we only need to make decisions when an event occurs. This makes the SMDP framework more effective to capture real-time network slicing systems.

A. Decision Epoch

Under our network slicing system model, the provider needs to make a decision upon receiving requests from tenants. Thus, the decision epoch can be defined as the inter-arrival time between two successive slice requests.

B. State Space

The system state $\mathbf{s}$ of the SMDP at the current decision epoch captures the number of slices n_c from a given class c ($\forall c \in \mathcal{C}$) being simultaneously run/served in the system. Formally, we define $\mathbf{s}$ as an $1 \times C$ vector:

$$\mathbf{s} \triangleq [n_1, \ldots, n_c, \ldots n_C]. \tag{2}$$

Given the network provider's resource constraints in (1), the state space $\mathcal{S}$ of all possible states $\mathbf{s}$ is defined as:

$$\mathcal{S} \triangleq \Bigg\{ \mathbf{s} = [n_1, \ldots, n_c, \ldots n_C] : \Theta \geq \sum_{c=1}^{C} r_c^{re} n_c; \; \Omega \geq \sum_{c=1}^{C} \omega_c^{re} n_c; \Delta \geq \sum_{c=1}^{C} \delta_c^{re} n_c \Bigg\}. \tag{3}$$

At the current system state $\mathbf{s}$, we define the *event vector* $\mathbf{e} \triangleq [e_1, \ldots, e_c, \ldots, e_C]$ with $e_c \in \{1, -1, 0\}, \forall c \in \mathcal{C}$. e_c equals to "1" if a new slice request from class c arrives, e_c equals to "-1" if a slice's resources are being released (also referred to as a slice completion/departing) to the system's resource, and e_c equals "0" otherwise (i.e., no slice request arrives nor completes/departs from the system). The set $\mathcal{E}$ of all the possible events is then defined as follows:

$$\mathcal{E} \triangleq \Big\{ \mathbf{e} : e_c \in \{-1, 0, 1\}; \sum_{c=1}^{C} |e_c| \leq 1 \Big\}, \tag{4}$$

where the *trivial event* $\mathbf{e}^* \triangleq (0, \ldots, 0) \in \mathcal{E}$ means no request arrival or completion/departing from all C classes.

C. Action Space

At state $\mathbf{s}$, if a slice request arrives (i.e., there exists $c \in \mathcal{C}$ such that $e_c = 1$), the network provider can choose either to accept or reject this request to maximize its long-term return. Let $a_{\mathbf{s}}$ denote the action to be taken at state $\mathbf{s}$ where $a_{\mathbf{s}} = 1$ if an arrival slice is accepted and $a_{\mathbf{s}} = 0$ otherwise. The state-dependent action space $\mathcal{A}_{\mathbf{s}}$ can be defined by:

$$\mathcal{A}_{\mathbf{s}} \triangleq \{a_{\mathbf{s}}\} = \{0, 1\}. \tag{5}$$

D. State Transition Probability

As aforementioned, in this work, we propose reinforcement learning approaches which can obtain the optimal policy for the network provider without requiring information from the environment (to cope with the uncertain demands and dynamics of slice requests). However, to lay a theoretical foundation and to evaluate the performance of our proposed solutions, we first assume that the arrival process of slice requests from

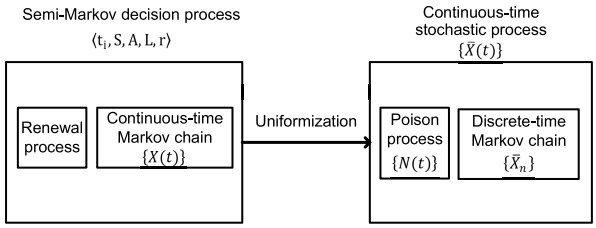


Fig. 2. Uniformization technique.

class c follows the Poisson distribution with mean rate λ_c and its network resource occupation time follows the exponential distribution with mean $1/\mu_c$. The assumptions allow us to analyze the dynamics of the SMDP, which is characterized by the state transition probabilities of the underlying Markov chain. In particular, our SMDP model consists of a renewal process and a continuous-time Markov chain $\{X(t : t \geq 0)\}$ in which the sojourn time in a state is a continuous random variable. We then can adopt the uniformization technique [36] to determine the probabilities for events and derive the transition probabilities $\mathcal{L}$. As shown in Fig. 2, the uniformization technique transforms the original continuous-time Markov chain $\{X(t) : t \geq 0\}$ into an equivalent stochastic process $\{\overline{X}(t), t \geq 0\}$ in which the transition epochs are generated by a Poisson process $\{N(t) : t \geq 0\}$ at a uniform rate and the state transitions are governed by the discrete-time Markov chain $\{\overline{X}_n\}$ [37], [38]. The details of the uniformization technique are as the following.

Our Markov chain $\{X(t)\}$ can be considered as a time-homogeneous Markov chain. Suppose that $\{X(t)\}$ is in state $\mathbf{s}$ at the current time t. If the system leaves state $\mathbf{s}$, it transfers to state $\mathbf{s}'$ ($\neq \mathbf{s}$) with probability $p_{\mathbf{s},\mathbf{s}'}(t)$. The probability that the process will leave state $\mathbf{s}$ in the next Δt to state $\mathbf{s}'$ is expressed as follows:

$$P\{X(t+\Delta t) = \mathbf{s}' | X(t) = \mathbf{s}\} = \begin{cases} z_{\mathbf{s}} \Delta t \times p_{\mathbf{s},\mathbf{s}'}(t) + o(\Delta t), & \mathbf{s}' \neq \mathbf{s}, \\ 1 - z_{\mathbf{s}} \Delta t + o(\Delta t), & \mathbf{s}' = \mathbf{s}, \end{cases} \quad (6)$$

as $\Delta t \to 0$ and $z_{\mathbf{s}}$ is the occurrence rate of the next event expressed as follows:

$$z_{\mathbf{s}} = \sum_{c=1}^{C} (\lambda_c + n_c \mu_c). \quad (7)$$

In the uniformization technique, we consider that the occurrence rate $z_{\mathbf{s}}$ of the states are identical, i.e., $z_{\mathbf{s}} = z$ for all $\mathbf{s}$. Thus, the transition epochs can be generated by a Poisson process with rate z. To formulate the uniformization technique, we choose a number z with

$$z = \max_{\mathbf{s} \in \mathcal{S}} z_{\mathbf{s}}. \quad (8)$$

Now, we define a discrete-time Markov chain $\{\overline{X}_n\}$ whose one-step transition probabilities $\overline{p}_{\mathbf{s},\mathbf{s}'}(t)$ are given by:

$$\overline{p}_{\mathbf{s},\mathbf{s}'}(t) = \begin{cases} (z_{\mathbf{s}}/z) p_{\mathbf{s},\mathbf{s}'}(t), & \mathbf{s}' \neq \mathbf{s}, \\ 1 - z_{\mathbf{s}}/z, & \text{otherwise}, \end{cases} \quad (9)$$

for all $\mathbf{s} \in \mathcal{S}$. Let $\{N(t), t \geq 0\}$ be a Poisson process with rate z such that the process is independent of the discrete-time Markov chain $\{\overline{X}_n\}$. We then define the continuous-time stochastic process $\{\overline{X}(t), t \geq 0\}$ as follows:

$$\overline{X}(t) = \overline{X}_{N(t)}, \quad t \geq 0. \quad (10)$$

Equation (10) represents that the process $\{\overline{X}(t)\}$ makes state transitions at epochs generated by a Poisson process with rate z and the state transitions are governed by the discrete-time Markov chain $\{\overline{X}_n\}$ with one-step transition probabilities $\overline{p}_{\mathbf{s},\mathbf{s}'}(t)$ in (9). When the Markov chain $\{\overline{X}_n\}$ is in state $\mathbf{s}$, the system leaves to next state with probability $z_{\mathbf{s}}/z$ and is a self-transition with probability $1 - z_{\mathbf{s}}/z$. In fact, the transitions out of state $\mathbf{s}$ are delayed by a time factor of $z/z_{\mathbf{s}}$, while a factor of $z_{\mathbf{s}}/z$ corresponds to the time until a state transition from state $\mathbf{s}$. In addition, in our system model there is no terminal state, i.e., the discrete-time Markov chain describing the state transitions in the transformed process has to allow for self-transitions leaving the state of the process unchanged. Therefore, the continuous $\{\overline{X}(t)\}$ is probabilistically identical to the original continuous-time Markov chain $\{X(t)\}$. This statement can be expressed as the following equation:

$$\begin{aligned} &P\{\overline{X}(t+\Delta t) \\ &= \mathbf{s}' | \overline{X}(t) = \mathbf{s}\} = z\Delta t \times \overline{p}_{\mathbf{s},\mathbf{s}'} + o(\Delta t) \\ &= z_{\mathbf{s}} \Delta t \times p_{\mathbf{s},\mathbf{s}'} + o(\Delta t) \\ &= q_{\mathbf{s},\mathbf{s}'} \Delta t + o(\Delta t) \\ &= P\{X(t+\Delta t) = \mathbf{s}' | X(t) = \mathbf{s}\} \text{ for } \Delta t \to 0, \quad \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S} \\ &\quad \text{and } \mathbf{s} \neq \mathbf{s}', \end{aligned} \quad (11)$$

where $q_{\mathbf{s},\mathbf{s}'}$ is the infinitesimal transition rate of the continuous-time Markov chain $\{X(t)\}$ and is expressed as follows:

$$q_{\mathbf{s},\mathbf{s}'} = z_{\mathbf{s}} p_{\mathbf{s},\mathbf{s}'}, \quad \forall \mathbf{s},\ \mathbf{s}' \in \mathcal{S} \ \text{ and } \ \mathbf{s}' \neq \mathbf{s}. \quad (12)$$

Clearly, in our system, the occurrence rate of the next event $z_{\mathbf{s}} = \sum_{c=1}^{C} (\lambda_c + n_c \mu_c)$ are positive and bounded in $\mathbf{s} \in \mathcal{S}$. Thus, it is proved that the infinitesimal transition rates determine a unique continuous-time Markov chain $\{X(t)\}$ [37]. We then make a necessary corollary as follows:

Corollary 1: The probabilities $p_{\mathbf{s},\mathbf{s}'}(t)$ are given by:

$$p_{\mathbf{s},\mathbf{s}'}(t) = \sum_{n=0}^{\infty} e^{-zt} \frac{z t^n}{n!} \overline{p}^{(n)}_{\mathbf{s},\mathbf{s}'}, \quad \forall \mathbf{s},\ \mathbf{s}' \in \mathcal{S} \text{ and } t \geq 0, \quad (13)$$

where the probabilities $\overline{p}^{(n)}_{\mathbf{s},\mathbf{s}'}$ can be recursively computed from

$$\overline{p}^{(n)}_{\mathbf{s},\mathbf{s}'} = \sum_{k \in \mathcal{S}} \overline{p}^{(n-1)}_{\mathbf{s},k} \overline{p}_{k,\mathbf{s}'}, \quad n = 1, 2, \ldots \quad (14)$$

starting with $\overline{p}^{(0)}_{\mathbf{s},\mathbf{s}} = 1$ and $\overline{p}^{0}_{\mathbf{s},\mathbf{s}'} = 0$ $\forall \mathbf{s}' \neq \mathbf{s}$.

In the next theorem, we prove that two processes $\{\overline{X}(t)\}$ and $\{X(t)\}$ are probabilistically equivalent.

Theorem 1: $\{\overline{X}(t)\}$ and $\{X(t)\}$ are probabilistically equivalent as

$$p_{\mathbf{s},\mathbf{s}'}(t) = P\{\overline{X}(t) = \mathbf{s}' | X(0) = \mathbf{s}\}, \quad \forall \mathbf{s},\ \mathbf{s}' \in \mathcal{S} \text{ and } t \leq 0. \quad (15)$$

The proof of Theorem 1 is given in Appendix A. ■

From (13) and (14), the computational complexity of uniformization method is derived as $O(vt|\mathcal{S}|^2)$, where $|\mathcal{S}|$ is

the number of states of the system. Based on z and $z_{\mathbf{s}}$, we can determine the probabilities for events as follows. The probability for an arrival slice from class c occurring in the next event $\mathbf{e}$ equals λ_c/z. The probability for a departure slice from class c occurring in the next event $\mathbf{e}$ equals $n_c\mu_c/z$, and the probability for a trivial event occurring in the next event $\mathbf{e}$ is $1-z_{\mathbf{s}}/z$. Hence, we can derive the transition probability $\mathcal{L}$.

E. Reward Function

The immediate reward after action $a_{\mathbf{s}}$ is executed at state $\mathbf{s}\in\mathcal{S}$ is defined as follows:

$$r(\mathbf{s},a_{\mathbf{s}})=\begin{cases} r_c, & \text{if } e_c=1,\ \ a_{\mathbf{s}}=1, \text{ and } \mathbf{s}'\in\mathcal{S},\\ 0, & \text{otherwise.}\end{cases} \tag{16}$$

At state $\mathbf{s}$, if an arrival slice is accepted, i.e., $a_{\mathbf{s}}=1$, the system will move to next state $\mathbf{s}'$ and the network provider receives an immediate reward r_c. In contrast, the immediate reward is equal to 0 if an arrival slice is rejected or there is no slice request arriving at the system. The value of r_c represents the amount of money paid by the tenant based on resources and additional services required.

As our system's statistical properties are time-invariant, i.e., stationary, the decision policy π of the SMDP model, which is a pure strategy, i.e., accept or reject an arrival request, can be defined as a time-invariant mapping from the state space to the action space: $\mathcal{S}\rightarrow\mathcal{A}_{\mathbf{s}}$. Thus, the long-term average reward starting from state $\mathbf{s}$ can be formulated as follows:

$$\mathcal{R}_\pi(\mathbf{s})=\lim_{K\rightarrow\infty}\frac{\mathbb{E}\{\sum_{k=0}^{K}r(\mathbf{s}_k,\pi(\mathbf{s}_k))|\mathbf{s}_0=\mathbf{s}\}}{\mathbb{E}\{\sum_{k=0}^{K}\tau_k|\mathbf{s}_0=\mathbf{s}\}},\quad \forall\mathbf{s}\in\mathcal{S}, \tag{17}$$

where τ_k is the time interval between the k-th and $(k+1)$-th decision epoch, r is the immediate reward of the system, and $\pi(\mathbf{s})$ is the action corresponding to the policy π at state $\mathbf{s}$.

In the following theorem, we will prove that the limit in Equation (17) exists.

Theorem 2: Given the state space $\mathcal{S}$ is countable and there is a finite number of decision epochs within a certain considered finite time, we have:

$$\begin{aligned}\mathcal{R}_\pi(\mathbf{s})&=\lim_{K\rightarrow\infty}\frac{\mathbb{E}\{\sum_{k=0}^{K}r(\mathbf{s}_k,\pi(\mathbf{s}_k))|\mathbf{s}_0=\mathbf{s}\}}{\mathbb{E}\{\sum_{k=0}^{K}\tau_k|\mathbf{s}_0=\mathbf{s}\}}\\&=\frac{\overline{\mathcal{L}}_\pi r(\mathbf{s},\pi(\mathbf{s}))}{\overline{\mathcal{L}}_\pi y(\mathbf{s},\pi(\mathbf{s}))},\quad\forall\mathbf{s}\in\mathcal{S},\end{aligned} \tag{18}$$

where $y(\mathbf{s},\pi(\mathbf{s}))$ is the expected time interval between adjacent decision epochs when action $\pi(\mathbf{s})$ is taken under state $\mathbf{s}$, and

$$\overline{\mathcal{L}}_\pi=\lim_{K\rightarrow\infty}\frac{1}{K}\sum_{k=0}^{K-1}\mathcal{L}_\pi^k, \tag{19}$$

where $\mathcal{L}_\pi^k$ and $\overline{\mathcal{L}}_\pi$ are the transition probability matrix and the limiting matrix of the embedded Markov chain for policy π, respectively.

Proof: We first have the following lemma.

Lemma 1: Given the transition probability matrix $\mathcal{L}_\pi$, the limiting matrix $\overline{\mathcal{L}}_\pi$ exists.

The proof of Lemma 1 is given in Appendix B.

Since $\mathcal{L}_\pi$ is the transition probability matrix, $\overline{\mathcal{L}}_\pi$ exists as stated in Lemma 1. As the total of probabilities that the system transform from state $\mathbf{s}$ to other states is equal to 1, we have:

$$\sum_{\mathbf{s}'\in\mathcal{S}}\overline{\mathcal{L}}_\pi(\mathbf{s}'|\mathbf{s})=1. \tag{20}$$

From (20), we derive $\mathcal{L}_\pi^n$ and $\overline{\mathcal{L}}_\pi$ as follows:

$$\begin{aligned}\overline{\mathcal{L}}_\pi r(\mathbf{s},\pi(\mathbf{s}))&=\lim_{K\rightarrow\infty}\frac{1}{K+1}\mathbb{E}\{\sum_{k=0}^{K}r(\mathbf{s}_k,\pi(\mathbf{s}_k))\},\quad\forall\mathbf{s}\in\mathcal{S},\\ \overline{\mathcal{L}}_\pi y(\mathbf{s},\pi(\mathbf{s}))&=\lim_{N\rightarrow\infty}\frac{1}{N+1}\mathbb{E}\{\sum_{n=0}^{N}\tau_n\},\quad\forall\mathbf{s}\in\mathcal{S}.\end{aligned} \tag{21}$$

Therefore, (18) is obtained by taking ratios of these two quantities. Note that the limit of the ratios equals to the ratio of the limits, and that, when taking the limit of the ratios, the factor $\frac{1}{K+1}$ can be removed from the numerator and denominator. ■

Note that in our SMDP model, the embedded Markov chain is unichain including a single recurrent class and a set of transient states for all pure policies π [21]. Hence, the average reward $\mathcal{R}_\pi(\mathbf{s})$ is independent to the initial state, i.e., $\mathcal{R}_\pi(\mathbf{s})=\mathcal{R}_\pi,\forall\mathbf{s}\in\mathcal{S}$. The average reward maximization problem is then written as:

$$\begin{aligned}\max_{\pi}\ \mathcal{R}_\pi&=\frac{\overline{\mathcal{L}}_\pi r(\mathbf{s},\pi(\mathbf{s}))}{\overline{\mathcal{L}}_\pi y(\mathbf{s},\pi(\mathbf{s}))}\\ \text{s.t.}\ &\sum_{\mathbf{s}'\in\mathcal{S}}\overline{\mathcal{L}}_\pi(\mathbf{s}'|\mathbf{s})=1,\quad\forall\mathbf{s}\in\mathcal{S}.\end{aligned} \tag{22}$$

Our objective is to find the optimal admission policy that maximizes the average reward of the network provider, i.e.,

$$\pi^*=\operatorname*{argmax}_{\pi}\mathcal{R}_\pi. \tag{23}$$

As aforementioned, the network resources may come from multiple data centers with diverse connectivity among servers and data centers. In such a case, the above formulation can be straightforwardly extended by accommodating additional states to the system state space. Specifically, one can define the system state space that includes (i) services of requests together with their corresponding resources and order, i.e., the network slice blueprint, (ii) available resources and services at the servers, and (iii) connectivity among servers and data centers. This means that we just need to increase the state space, compared with the current state space (with three types of resources, as an example) in the current formulation, to capture additional resources and options. Then, the proposed admission/rejection framework can be implemented at the orchestrator to allocate the available resources to requested slices. Specifically, based on this state space, when a slice request arrives, the orchestrator is able to check whether to allocate an optimal possible link to the request (using existing network slicing mechanisms) and then makes a decision to accept or reject the request. In addition, after making a decision to allocate the resources for a slice request (i.e., after the initial deployment of VNFs), if the running slice requires

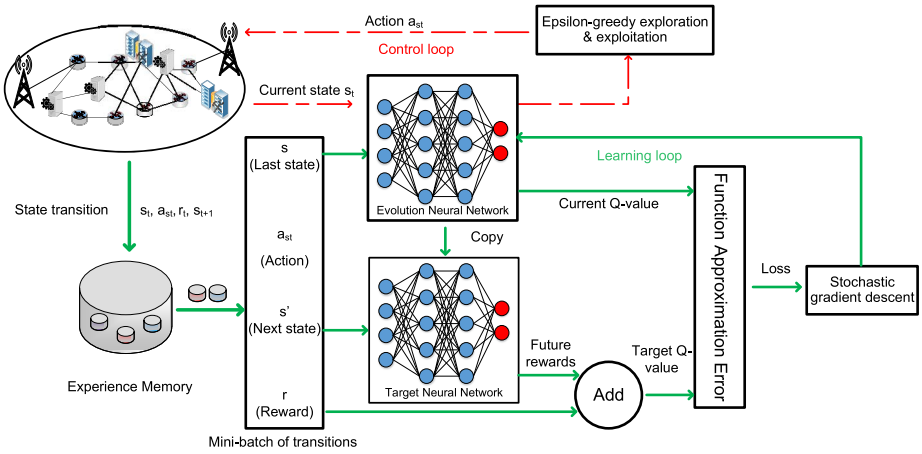


Fig. 3. Deep double Q-learning model.

to add more resources or remove some resources (i.e., scaling out or scaling in, respectively), we can consider some new events (i.e., requests to add or remove resources from running slices) to the system state space. Again, this implies that we only need to add more states to the system state space of the current model. Note that, the action space will be kept the same, i.e., only two actions (accept or reject), and we just need to set new rewards for accepting/rejecting requests from running slices in the problem formulation.

Note that the problem (22) requires environment information, i.e., arrival and completion rates of slice requests, to construct the transition probability matrix $\mathcal{L}$. Nevertheless, due to the uncertain demands and the dynamics of slice requests from tenants, these environment parameters may not be available and can be time-varying. To cope with the demand uncertainty, we consider deep double Q-learning and deep dueling algorithms to find the optimal admission policy at the RMO to maximize the long-term average reward.

IV. Q-Learning for Dynamic Resource Allocation Under Uncertainty of Slice Service Demands

Q-learning [39] is a reinforcement learning technique (to learn environment parameters while they are not available) to find the optimal policy. In particular, the algorithm implements a Q-table to store the value for each pair of state and action. Given the current state, the network provider will make an action based on its current policy. After that, the Q-learning algorithm observes the results, i.e., reward and next state, and updates the value of the Q-table accordingly. In this way, the Q-learning algorithm will be able to learn from its decisions and adjust its policy to converge to the optimal policy after a finite number of iterations [39]. In this paper, we aim to find the optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}_{\mathbf{s}}$ for the network provider to maximize its long-term average reward. Specifically, let denote $\mathcal{V}^{\pi}(\mathbf{s}) : \mathcal{S} \rightarrow \mathbb{R}$ as the expected value function obtained by policy π from each state $\mathbf{s} \in \mathcal{S}$:

$$\begin{aligned}\mathcal{V}^{\pi}(\mathbf{s}) &= \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty}\gamma^t r_t(\mathbf{s}_t, a_{\mathbf{s}_t})|\mathbf{s}_0 = \mathbf{s}\Big] \\ &= \mathbb{E}_{\pi}\Big[r_t(\mathbf{s}_t, a_{\mathbf{s}_t}) + \gamma\mathcal{V}^{\pi}(\mathbf{s}_{t+1})|\mathbf{s}_0 = \mathbf{s}\Big], \quad (24)\end{aligned}$$

where $0 \leq \gamma < 1$ is the discount factor which determines the importance of the long-term reward [39]. In particular, if γ is close to 0, the RMO will prefer to select actions to maximize its short-term reward. In contrast, if γ is close to 1, the RMO will make actions such that its long-term reward is maximized. $r_t(\mathbf{s}_t, a_{\mathbf{s}_t})$ is the immediate reward achieved by taking action $a_{\mathbf{s}_t}$ at state $\mathbf{s}_t$. Given a state $\mathbf{s}$, policy $\pi(\mathbf{s})$ is obtained by taking action $a_{\mathbf{s}}$ whose the value function is highest [39]. Since we aim to find optimal policy π^*, an optimal action at each state can be found through the optimal value function expressed by:

$$\mathcal{V}^*(\mathbf{s}) = \max_{a_{\mathbf{s}}}\Big\{\mathbb{E}_{\pi}[r_t(\mathbf{s}_t, a_{\mathbf{s}_t}) + \gamma\mathcal{V}^{\pi}(\mathbf{s}_{t+1})]\Big\}, \quad \forall \mathbf{s} \in \mathcal{S}. \quad (25)$$

The optimal Q-functions for state-action pairs are denoted by:

$$\mathcal{Q}^*(\mathbf{s}, a_{\mathbf{s}}) \triangleq r_t(\mathbf{s}_t, a_{\mathbf{s}_t}) + \gamma\mathbb{E}_{\pi}[\mathcal{V}^{\pi}(\mathbf{s}_{t+1})], \quad \forall \mathbf{s} \in \mathcal{S}. \quad (26)$$

Then, the optimal value function can be written as follows:

$$\mathcal{V}^*(\mathbf{s}) = \max_{a_{\mathbf{s}}}\{\mathcal{Q}^*(\mathbf{s}, a_{\mathbf{s}})\}. \quad (27)$$

By making samples iteratively, the problem is reduced to determining the optimal value of Q-function, i.e., $\mathcal{Q}^*(\mathbf{s}, a_{\mathbf{s}})$, for all state-action pairs. In particular, the Q-function is updated according to the following rule:

$$\begin{aligned}&\mathcal{Q}_t(\mathbf{s}_t, a_{\mathbf{s}_t}) \\ &= \mathcal{Q}_t(\mathbf{s}_t, a_{\mathbf{s}_t}) \\ &\quad + \alpha_t\Big[r_t(\mathbf{s}_t, a_{\mathbf{s}_t}) + \gamma\max_{a_{\mathbf{s}_{t+1}}}\mathcal{Q}_t(\mathbf{s}_{t+1}, a_{\mathbf{s}_{t+1}}) - \mathcal{Q}_t(\mathbf{s}_t, a_{\mathbf{s}_t})\Big].\end{aligned} \quad (28)$$

The key idea behind this rule is to find the temporal difference between the predicted Q-value, i.e., $r_t(\mathbf{s}_t, a_{\mathbf{s}_t}) + \gamma\max_{a_{\mathbf{s}_{t+1}}}\mathcal{Q}_t(\mathbf{s}_{t+1}, a_{\mathbf{s}_{t+1}})$ and its current value, i.e., $\mathcal{Q}_t(\mathbf{s}_t, a_{\mathbf{s}_t})$. In (28), the learning rate α_t determines the impact of new information to the existing value. During the learning process, α_t can be adjusted dynamically, or it can be chosen to be a constant. However, to guarantee the convergence for the Q-learning algorithm, α_t is deterministic, nonnegative, and satisfies the following conditions [39]:

$$\alpha_t \in [0, 1), \quad \sum_{t=1}^{\infty}\alpha_t = \infty, \text{ and } \sum_{t=1}^{\infty}(\alpha_t)^2 < \infty. \quad (29)$$

Based on (28), the RMO can employ the Q-learning to obtain the optimal policy. Specifically, the algorithm first initializes the table entry $\mathcal{Q}(\mathbf{s}, a_{\mathbf{s}})$ arbitrarily, e.g., to zero for each state-action pair $(\mathbf{s}, a_{\mathbf{s}})$. From the current state $\mathbf{s}_t$, the algorithm will choose action $a_{\mathbf{s}_t}$ and observe results after performing this action. In practice, to select action $a_{\mathbf{s}_t}$, ϵ-greedy algorithm [18], [40] is often used. Specifically, this method introduces a parameter ϵ which suggests for the controller in choosing a random action with probability ϵ or select an action that maximizes the $\mathcal{Q}(\mathbf{s}_t, a_{\mathbf{s}_t})$ with probability $1-\epsilon$. Doing so, the algorithm can explore the whole state space. Hence, we need to balance between the exploration time, i.e., ϵ, and the exploitation time, i.e., $1-\epsilon$, to speed up the convergence of the Q-learning algorithm. The algorithm then determines the next state and reward after performing the chosen action and update the table entry for $\mathcal{Q}(\mathbf{s}_t, a_{\mathbf{s}_t})$ based on (28). Once either all Q-values converge or the certain number of iterations is reached, the algorithm will be terminated. This algorithm yields the optimal policy indicating an action to be taken at each state such that $\mathcal{Q}^*(\mathbf{s}, a_{\mathbf{s}})$ is maximized for all states in the state space, i.e., $\pi^*(\mathbf{s}) = \text{argmax}_{a_{\mathbf{s}}} \mathcal{Q}^*(\mathbf{s}, a_{\mathbf{s}})$. Under (29), it was proved in [39] that the Q-learning algorithm will converge to the optimum action-values with probability one.

Several studies in the literature reported the application of the Q-learning algorithm to address the network slicing problem, e.g., [14]. Note that the Q-learning algorithm can efficiently obtain the optimal policy when the state space and action space are small. However, when the state or action space is large, the Q-learning algorithm often converges prohibitively slow. For the combinatorial resource slicing problem in this work, the state space is can be as large as tens of thousands. That makes the Q-learning algorithm practically inapplicable (especially for real-time resource slicing) [34]. In the sequel, leveraging the deep double Q-learning and deep dueling networks, we develop optimal and fast algorithms to overcome this shortcoming.

V. Fast and Optimal Resource Slicing With Deep Neural Network

A. Deep Double Q-Learning

In this section, we introduce the deep double Q-learning algorithm to address the slow-convergence problem of Q-learning algorithm. Originally, the deep double Q-learning algorithm was developed by Google DeepMind in 2016 [41] to teach machines to play games without human control. Specifically, the deep double Q-learning algorithm is introduced to further improve the performance of the deep Q-learning algorithm [35]. The key idea of the deep double Q-learning algorithm is to select an action by using the primary network. It then uses the target network to compute the target Q-value for the action.

As stated in [35], the performance of reinforcement learning approaches might not be stable or even diverges when a nonlinear function approximator is used. This is attributed to the fact that a small change of Q-values may greatly affect the policy. Thereby the data distribution and the correlations between the Q-values and the target values are varied. To address this issue, we use the experience replay mechanism, the target Q-network, and a proper feature set selection:

- *Experience Replay Mechanism:* The algorithm will store transitions $(\mathbf{s}_t, a_{\mathbf{s}_t}, r_t, \mathbf{s}_{t+1})$ in a replay memory, i.e., memory pool, instead of running on state-action pairs as they occur during experience. The learning process is then performed based on random samples from the memory pool. By doing so, the previous experiences are exploited more efficiently as the algorithm can learn them many times. Additionally, by using the experience mechanism, the data is more like independent and identically distributed. That removes the correlations between observations.
- *Target Q-Network:* In the training process, the Q-value will be shifted. Thus, the value estimations can be out of control if a constantly shifting set of values is used to update the Q-network. This destabilizes the algorithm. To address this issue, we use the target Q-network to frequently (but slowly) update to the primary Q-networks values. That significantly reduces the correlations between the target and estimated Q-values, thereby stabilizing the algorithm.
- *Feature Set:* For each state, we determine four features including radio, computing, storage, and event trigger, i.e., a slice request arrives. These features are then fed into the deep neural network to approximate the Q-values for each action of a state. As such, all aspects of each state are trained in the deep neural network, resulting in a higher convergence rate.

Algorithm 1 Deep Double Q-Learning Based Resource Slicing Algorithm

1: Initialize replay memory to capacity D.
2: Initialize the Q-network $\mathcal{Q}$ with random weights θ.
3: Initialize the target Q-network $\hat{\mathcal{Q}}$ with weight $\theta^- = \theta$.
4: **for** *episode=1 to T* **do**
5: With probability ϵ select a random action $a_{\mathbf{s}_t}$, otherwise select $a_{\mathbf{s}_t} = \text{argmax}\, \mathcal{Q}^*(\mathbf{s}_t, a_{\mathbf{s}_t}; \theta)$
6: Perform action $a_{\mathbf{s}_t}$ and observe reward r_t and next state $\mathbf{s}_{t+1}$
7: Store transition $(\mathbf{s}_t, a_{\mathbf{s}_t}, r_t, \mathbf{s}_{t+1})$ in the replay memory
8: Sample random minibatch of transitions $(\mathbf{s}_j, a_{\mathbf{s}_j}, r_j, \mathbf{s}_{j+1})$ from the replay memory
9: $y_j = r_j + \gamma \mathcal{Q}(\mathbf{s}_{j+1}, \text{argmax}_{a_{\mathbf{s}_{j+1}}} \hat{\mathcal{Q}}(\mathbf{s}_{j+1}, a_{\mathbf{s}_{j+1}}; \theta); \theta^-)$
10: Perform a gradient descent step on $(y_j - \mathcal{Q}(\mathbf{s}_j, a_{\mathbf{s}_j}; \theta))^2$ with respect to the network parameter θ.
11: Every C steps reset $\hat{\mathcal{Q}} = \mathcal{Q}$
12: **end for**

The details of the deep double Q-learning algorithm is provided in Algorithm 1 and explained more details in the flowchart in Fig. 4. Specifically, as shown in Fig. 4, the training phase is composed of multiple episodes. In each episode, the RMO performs an action and learns from observations corresponding to the taken action. As a result, the RMO needs to tradeoff between the exploration and exploitation processes

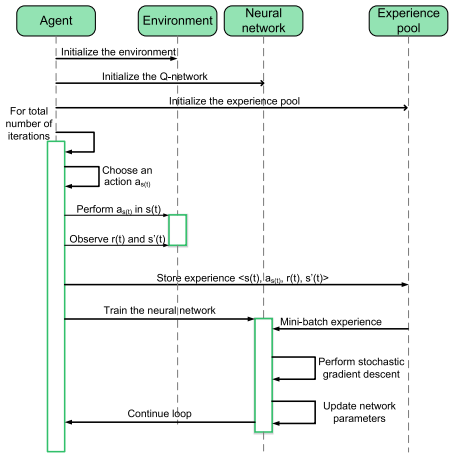


Fig. 4. Flow chart of the deep double Q-learning algorithm.

over the state space. Therefore, in each episode, given the current state, the algorithm will choose an action based on the epsilon-greedy algorithm. The algorithm will start with a fairly randomized policy and later slowly move to a deterministic policy. This means that, at the first episode, ϵ is set at a large value, e.g., 0.9, and gradually decayed to a small value, e.g., 0.1. After that, the RMO will perform the selected action and observe results, i.e., next state and reward, from taking this action. This transition is then stored in the replay memory for the training process at later episodes.

In the learning process, random samples of transitions from the replay memory will be fed into the neural network. In particular, for each state, we formulate 4 features, i.e., radio, computing, storage, and arrival event, as the input of the deep neural network. In this way, the training process is more efficient as all aspects of states are taken into account. The algorithm then updates the neural network by minimizing the following lost functions [35].

$$L_i(\theta_i) = \mathbb{E}_{(\mathbf{s},a_\mathbf{s},r,\mathbf{s}')\sim U(D)}\Bigg[\Bigg(r + \gamma \mathcal{Q}(\mathbf{s}', \underset{a_{\mathbf{s}'}}{\operatorname{argmax}}\, \hat{\mathcal{Q}}(\mathbf{s}', a_{\mathbf{s}'}; \theta); \theta^-)) - \mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \theta_i)\Bigg)^2\Bigg], \quad (30)$$

where γ is the discount factor, θ_i are the parameters of the Q-networks at episode i and θ_i^- are the parameters of the target network, i.e., $\hat{\mathcal{Q}}$. θ_i and θ_i^- are used to compute the target at episode i.

Differentiating the loss function in (30) with respect to the parameters of the neural networks, we have the following gradient:

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{(\mathbf{s},a_\mathbf{s},r,\mathbf{s}')}\Bigg[\Bigg(r + \gamma \mathcal{Q}(\mathbf{s}', \underset{a_{\mathbf{s}'}}{\operatorname{argmax}}\, \hat{\mathcal{Q}}(\mathbf{s}', a_{\mathbf{s}'}; \theta); \theta^-)) - \mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \theta_i)\nabla_{\theta_i} \mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \theta_i)\Bigg)\Bigg]. \quad (31)$$

From (31), the loss function in (30) can be minimized by the *Stochastic Gradient Descent* algorithm [46] that is the engine of most deep learning algorithms. Specifically, stochastic gradient descent is an extension of the gradient descent algorithm which is commonly used in machine learning. In general, the cost function used by a machine learning algorithm is decayed by a sum over training examples of some per-example loss function. For example, the negative conditional log-likelihood of the training data can be formulated as:

$$J(\theta) = \mathbb{E}_{(\mathbf{s},a_\mathbf{s},r,\mathbf{s}')\sim U(D)} L\big((\mathbf{s}, a_\mathbf{s}, r, \mathbf{s}'), \theta\big) = \frac{1}{D}\sum_{i=1}^{D} L\big((\mathbf{s}, a_\mathbf{s}, r, \mathbf{s}')^{(i)}, \theta\big). \quad (32)$$

For these additive cost function, gradient descent requires computing as follows:

$$\nabla_\theta J(\theta) = \frac{1}{D}\sum_{i=1}^{D} \nabla_\theta L\big((\mathbf{s}, a_\mathbf{s}, r, \mathbf{s}')^{(i)}, \theta\big). \quad (33)$$

The computational cost for operation in (33) is $O(D)$. Thus, as the size D of the replay memory is increased, the time to take a single gradient step becomes prohibitively long. As a result, the stochastic gradient descent technique is used in this paper. The core idea of using stochastic gradient descent is that the gradient is an expectation. Obviously, the expectation can be approximately estimated by using a small set of samples. In particular, we can uniformly sample a mini-batch of experiences from the replay memory at each step of the algorithm. Typically, the mini-batch size can be set to be relative small number of experiences, e.g., from 1 to a few hundreds. As such, the training time is significantly fast. The estimate of the gradient under the stochastic gradient descent is then rewritten as follows:

$$g = \frac{1}{D'}\nabla_\theta \sum_{i=1}^{D'} L\big((\mathbf{s}, a_\mathbf{s}, r, \mathbf{s}')^{(i)}, \theta\big), \quad (34)$$

where D' is the mini-batch size. The stochastic gradient descent algorithm then follows the estimated gradient downhill as in (35).

$$\theta \leftarrow \theta - \nu g, \quad (35)$$

where ν is the learning rate of the algorithm.

The target network parameters θ_i^- are only updated with the Q-network parameters θ_i every C steps and are remained fixed between individual updates. It is worth noting that the training process of the deep double Q-learning algorithm is different from the training process in supervised learning by updating the network parameters using previous experiences in an online manner.

B. Deep Dueling Network

Due to the overestimation of optimizers, the convergence rates of the deep Q-learning and deep double Q-learning learning algorithms are still limited, especially in large-scale systems [42]. Therefore, we propose the novel network slicing framework using the deep dueling algorithm [42], which is also originally developed by Google DeepMind in 2016, to further improve the system's convergence speed. The key idea

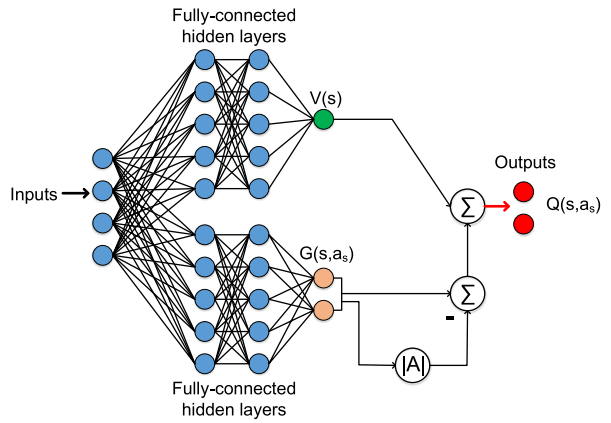


Fig. 5. Deep dueling model.

making the deep dueling superior to conventional approaches is its novel neural network architecture. In this neural network, instead of estimating the action-value function, i.e., Q-function, the values of states and advantages of actions[1] are separately estimated by two sequences, i.e., streams, of fully connected layers. The values and advantages are combined at the output layer as shown in Fig. 5. The reason behind this architecture is that in many states it is unnecessary to estimate the value of corresponding actions as the choice of these actions has no repercussion on what happens [42]. In this way, the deep dueling algorithm can achieve more robust estimates of state value, thereby significantly improving its convergence rate as well as stability.

Recall that given a stochastic policy π, the values of state-action pair $(\mathbf{s}, a_\mathbf{s})$ and state $\mathbf{s}$ are expressed as:

$$\mathcal{Q}^\pi(\mathbf{s}, a_\mathbf{s}) = \mathbb{E}\big[\mathcal{R}_t | \mathbf{s}_t = \mathbf{s}, a_{\mathbf{s}_t} = a_\mathbf{s}, \pi\big] \text{ and}$$
$$\mathcal{V}_\pi(\mathbf{s}) = \mathbb{E}_{a_\mathbf{s}\sim\pi(\mathbf{s})}\big[\mathcal{Q}^\pi(\mathbf{s}, a_\mathbf{s})\big]. \quad (36)$$

The advantage function of actions can be computed as:

$$\mathcal{G}^\pi(\mathbf{s}, a_\mathbf{s}) = \mathcal{Q}^\pi(\mathbf{s}, a_\mathbf{s}) - \mathcal{V}^\pi(\mathbf{s}, a_\mathbf{s}). \quad (37)$$

Specifically, the value function $\mathcal{V}$ corresponds to how good it is to be in a particular state $\mathbf{s}$. The state-action pair, i.e., Q-function, measures the value of selecting action $a_\mathbf{s}$ in state $\mathbf{s}$. The advantage function decouples the state value from Q-function to obtain a relative measure of the importance of each action. It is important to note that $\mathbb{E}_{a_\mathbf{s}\sim\pi(\mathbf{s})}\big[\mathcal{G}^\pi(\mathbf{s}, a_\mathbf{s})\big] = 0$. In addition, given a deterministic policy $a^*_\mathbf{s} = \text{argmax}_{a_\mathbf{s}\in\mathcal{A}}\, \mathcal{Q}(\mathbf{s}, a_\mathbf{s})$, we have $\mathcal{Q}(\mathbf{s}, a^*_\mathbf{s}) = \mathcal{V}(\mathbf{s})$, and hence $\mathcal{G}(\mathbf{s}, a^*_\mathbf{s}) = 0$.

To estimate values of $\mathcal{V}$ and $\mathcal{G}$ functions, we use a dueling neural network in which one stream of fully-connected layers outputs a scalar $\mathcal{V}(\mathbf{s}; \beta)$ and the other stream outputs an $|\mathcal{A}|$-dimensional vector $\mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha)$ with α and β are the parameters of fully-connected layers. These two streams are then combined to obtain the Q-function by (38).

$$\mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \alpha, \beta) = \mathcal{V}(\mathbf{s}; \beta) + \mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha). \quad (38)$$

[1]The value function represents how good it is for the system to be in a given state. The advantage function is used to measure the importance of a certain action compared with others [42].

However, $\mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \alpha, \beta)$ is only a parameterized estimate of the true Q-function. Moreover, given Q, we cannot obtain $\mathcal{V}$ and $\mathcal{G}$ uniquely. Therefore, (38) is unidentifiable resulting in poor performance. To address this issue, we let the combining module of the network implement the following mapping:

$$\mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \alpha, \beta) = \mathcal{V}(\mathbf{s}; \beta) + \big(\mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha) - \max_{a_\mathbf{s}\in\mathcal{A}} \mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha)\big). \quad (39)$$

By doing this, the advantage function estimator has zero advantage when choosing action. Intuitively, given $a^*_\mathbf{s} = \text{argmax}_{a_\mathbf{s}\in\mathcal{A}}\, \mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \alpha, \beta) = \text{argmax}_{a_\mathbf{s}\in\mathcal{A}}\, \mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha)$, we have $\mathcal{Q}(\mathbf{s}, a^*_\mathbf{s}; \alpha, \beta) = \mathcal{V}(\mathbf{s}; \beta)$. (39) can be transformed into a simple form by replacing the max operator with an average as in (40).

$$\mathcal{Q}(\mathbf{s}, a_\mathbf{s}; \alpha, \beta) = \mathcal{V}(\mathbf{s}; \beta) + \big(\mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a_\mathbf{s}} \mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha)\big). \quad (40)$$

Algorithm 2 Deep Dueling Network Based Resource Slicing Algorithm

1: Initialize replay memory to capacity D.
2: Initialize the primary network $\mathcal{Q}$ including two fully-connected layers with random weights α and β.
3: Initialize the target network $\hat{\mathcal{Q}}$ as a copy of the primary Q-network with weights $\alpha^- = \alpha$ and $\beta^- = \beta$.
4: **for** *episode=1 to T* **do**
5: Base on the epsilon-greedy algorithm, with probability ϵ select a random action $a_{\mathbf{s}_t}$, otherwise
6: select $a_{\mathbf{s}_t} = \text{argmax}\, \mathcal{Q}^*(\mathbf{s}_t, a_{\mathbf{s}_t}; \alpha, \beta)$
7: Perform action $a_{\mathbf{s}_t}$ and observe reward r_t and next state $\mathbf{s}_{t+1}$
8: Store transition $(\mathbf{s}_t, a_{\mathbf{s}_t}, r_t, \mathbf{s}_{t+1})$ in the replay memory
9: Sample random minibatch of transitions $(\mathbf{s}_j, a_{\mathbf{s}_j}, r_j, \mathbf{s}_{j+1})$ from the replay memory
10: Combine the value function and advantage functions as follows:

$$\mathcal{Q}(\mathbf{s}_j, a_{\mathbf{s}_j}; \alpha, \beta) = \mathcal{V}(\mathbf{s}_j; \beta) + \big(\mathcal{G}(\mathbf{s}_j, a_{\mathbf{s}_j}; \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a_{\mathbf{s}_j}} \mathcal{G}(\mathbf{s}_j, a_{\mathbf{s}_j}; \alpha)\big). \quad (41)$$

11: $y_j = r_j + \gamma \max_{a_{\mathbf{s}_{j+1}}} \hat{\mathcal{Q}}(s_{j+1}, a_{\mathbf{s}_{j+1}}; \alpha^-, \beta^-)$
12: Perform a gradient descent step on $(y_j - \mathcal{Q}(\mathbf{s}_j, a_{\mathbf{s}_j}; \alpha, \beta))^2$
13: Every C steps reset $\hat{\mathcal{Q}} = \mathcal{Q}$
14: **end for**

Based on (40) and the advantages of the deep reinforcement learning, the details of the deep dueling algorithm used in our proposed approach are shown in Algorithm 2. It is important to note that (40) is viewed and implemented as a part of the network and not as a separate algorithmic step [42]. In addition, $\mathcal{V}(\mathbf{s}; \beta)$ and $\mathcal{G}(\mathbf{s}, a_\mathbf{s}; \alpha)$ are estimated automatically without any extra supervision or modifications in the algorithm.

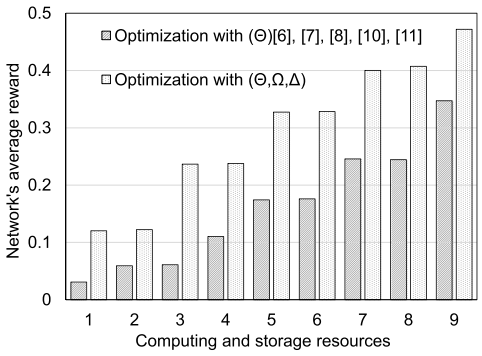


Fig. 6. The average reward when optimizing with one resource and three resources.

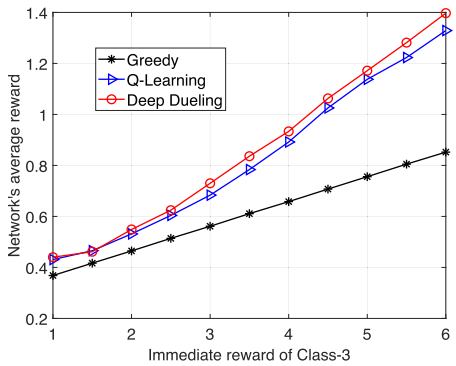


Fig. 7. The average reward of the system when the immediate reward of class-3 is varied.

VI. Performance Evaluation

A. Parameter Setting

We perform the simulations using TensorFlow [43] to evaluate the performance of the proposed solutions under different parameter settings. We consider three common classes of slices, i.e., utilities (class-1), automotive (class-2), and manufacturing (class-3). Unless otherwise stated, the arrival rates λ_c of requests from class-1, class-2, and class-3 are set at 12 requests/hour, 8 requests/hour, and 10 requests/hour, respectively. The completion rates μ_c of requests from class-1, class-2, and class-3 are set at 3 requests/hour. The immediate reward r_c for each accepted request from class-1, class-2, and class-3 are 1, 2, and 4, respectively. These parameters will be varied later to evaluate the impacts of the immediate reward on the decisions of the RMO. Each slice request requires 1 GB of storage resources, 2 CPUs for computing, and 100 Mbps of radio resources [44]. Importantly, the architecture of the deep neural network requires thoughtful design as it greatly affects the performance of the algorithm. Intuitively, increasing the number of hidden layers will increase the complexity of the algorithm. However, when the number of hidden layers is very small, the algorithm may not converge to the optimal policy. Similarly, when the size of hidden layers and mini-batch size are large, the algorithm will need more time to estimate the Q-function. In our experiment, we choose these parameters based on common settings in the literature [35], [42]. In particular, for the deep Q-learning and deep double Q-learning algorithms, two fully-connected hidden layers are implemented together with input and output layers. For the deep dueling algorithm, the neural network is divided into two streams [42]. Each stream consists of a hidden layer connected to the input and output layers. The size of the hidden layers is 64. The mini-batch size is set at 64. Both the Q-learning algorithm and the deep reinforcement learning algorithms use ϵ-greedy algorithm with the initial value of ϵ is 1, and its final value is 0.1 [39], [45]. The maximum size of the experience replay buffer is 10,000, and the target Q-network is updated every 1,000 iterations [35], [46].

B. Simulation Results

1) Performance Evaluation:

a) Comparison to Existing Network Slicing Solutions: As mentioned, most existing works, e.g., [6], [7], [13]–[15], optimized slicing for only the radio resource. In practice, besides the radio resources both computing and storage resources should also be accounted for while orchestrating slices. This makes existing solutions sub-optimal. In this section, we set the maximum radio resources at 500 Mbps. Each request requires 50 Mbps for radio access, 2 CPUs for computing, and 2 GB of storage resources. The computing and storage resources are then varied from 1 CPU to 9 CPUs and 1 GB to 9 GB, respectively. Fig. 6 shows the average reward of the system obtained by the Q-learning algorithm for the case with three resources are taken into account (as in our considered system model) and for the case with only radio resources as considered in [6], [7], and [13]–[15]. As can be observed, when the computing and storage resources increase, the average reward is increased as more slice requests are accepted. However, the average reward of our approach (taking all radio, computing, and storage resources into account) is significantly higher than those of other solutions in the literature, especially when the amount of computing and storage resources are small. This is due to the fact that slices not only request radio resources to ensure the bandwidth for connections but also computing and storage resources to fulfill the requirements of different services.

b) Average Reward and Network Performance: Next, we compare the performance of the proposed solution, i.e., deep dueling algorithm, with other methods, i.e., Q-learning [14] and greedy algorithms [15], [47], in terms of average reward and the number of requests running in the system. For a small-size system (the maximum radio, computing, and storage resources are set at 400 Mbps, 8 CPUs, and 4 GB, respectively), Fig. 7 shows the average reward of the system obtained by three algorithms while varying the reward of slices from class-3 from 1 to 6. As can be seen, with the increasing of the reward of slices from class-3, the average reward of the system is increased. However, the average reward obtained by the reinforcement learning algorithms, i.e., deep dueling and Q-learning, is significantly higher than that of the greedy algorithm. This is due to the fact that the proposed reinforcement learning approaches reserve resources for coming requests that may have high rewards, while the greedy algorithm accepts slices based on the available resource of the system as shown in Fig. 8. It is worth noting that the achieved reward of the Q-learning algorithm is not as good as the reward obtained by the deep dueling algorithm even

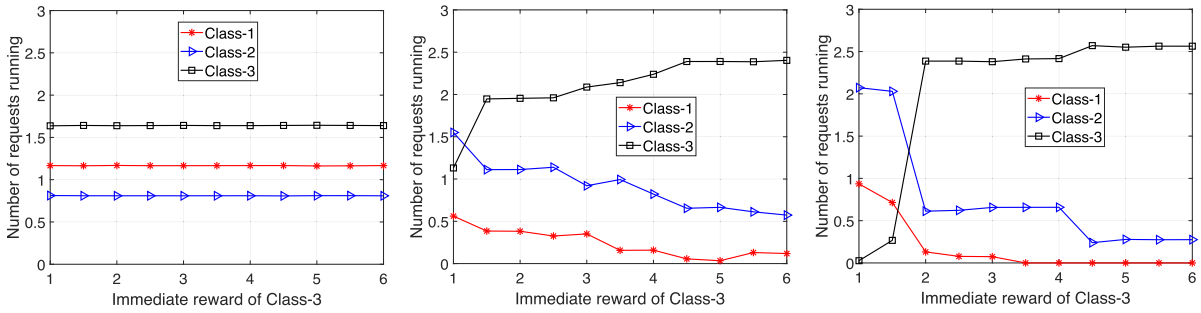


Fig. 8. The number of request running in the system of (a) greedy algorithm, (b) Q-learning algorithm, and (c) deep dueling algorithm when the immediate reward of class-3 is varied.

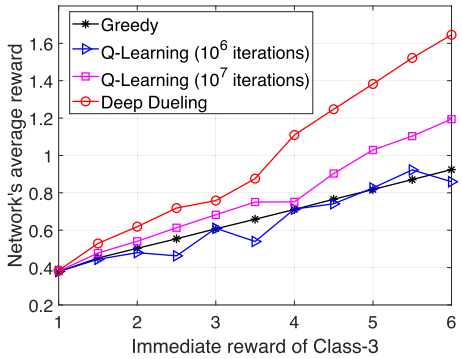


Fig. 9. The average reward of the system when the immediate reward of class-3 is varied.

with small-size scenarios. This is because that the Q-learning algorithm has a slow convergence rate due to the curse-of-dimensionality problem. This observation is more pronounced when we later increase the size of the system.

As observed in Fig. 8, the number of requests running in the system under the greedy algorithm remains the same when the immediate reward of slices from class-3 is varied. The reason is that the greedy algorithm does not consider the immediate reward of slice requests into account. In other words, upon receiving a slice request, the greedy algorithm will accept this request if the available resources of the infrastructure satisfy the slice service demands. In contrast, for the reinforcement learning algorithms, the immediate reward is also an essential factor to make optimal decisions. In particular, when the immediate reward of slice requests from class-3 increases, the algorithms are likely to reject the slice requests from classes which have lower immediate rewards, i.e., slice requests from class-1. For example, when the immediate reward of slice request from class-3 is 6, the number of requests from class-1, whose immediate reward is 1, approaches 0.

To observe the performance of the proposed solutions when the state space of the system is large, we increase the radio, computing, and storage resources to 2 Gbps, 40 CPUs, and 20 GB, respectively. The arrival rate of requests from class-1 is 48 requests/hour, from class-2 is 32 requests/hour, and from class-3 is 40 requests/hour. The completion rates from all classes are set at 2 requests/hour. Fig. 9 shows that the average reward obtained by the deep dueling algorithm is much higher than those of the greedy and Q-learning algorithms. This is because of the slow convergence of the Q-learning algorithm to optimality. Specifically, with 10^6 iterations, the performance of the Q-learning algorithm is just the same as that of the greedy algorithm. The performance of the Q-learning algorithm is improved with 10^7 iterations, but it is still way inferior to that of the deep dueling algorithm. For this large system scenario with over 74,000 state-action pairs, on a laptop with Intel Core i7-7600U and 16GB RAM, the deep dueling algorithm just takes about 2 hours to finish 15,000 iterations and obtain the optimal policy. This is a very practical number compared with the Q-learning algorithm that cannot obtain the optimal policy within 10^7 iterations (more than 15 hours). In practice, with specialized hardware and much more powerful computing resource (compared with our laptop) at the network provider (e.g., GPU cards from NVIDIA), the deep dueling algorithm should take much shorter than 2 hours to finish 15,000 iterations [48]. These results confirm that the Q-learning algorithm, despite its optimality, requires a much longer time to converge, compared with the deep dueling algorithm.

Similar to the case in Fig. 8, as shown in Fig. 10, the deep dueling and Q-learning algorithms reserve resources for slices from classes which have high immediate rewards. However, the deep dueling algorithm achieves better performance compared to the Q-learning algorithm. For example, when the immediate reward of slices from class-3 is 6, the number of requests running in the systems is about 16 requests and 11 requests for the deep dueling and the Q-learning algorithms, respectively.

In summary, in all the cases, the deep dueling algorithm always achieves the best performance in terms of the average reward and network performance.

c) Optimal Policy: In Fig. 11, we examine the optimal policy of the deep dueling and Q-learning algorithms. Specifically, we set the maximum resources of the system at 4 times, 10 times, and 20 times of resources requested by a slice and evaluate the policy of the algorithms with different available resources in the system as shown in Fig. 11(a), Fig. 11(b), and Fig. 11(c), respectively. Note that the lines Q-learning{1,2,3} and Deep Dueling{1,2,3} represent the probabilities of accepting requests from class-{1,2,3} by using the Q-learning and deep dueling algorithms, respectively.

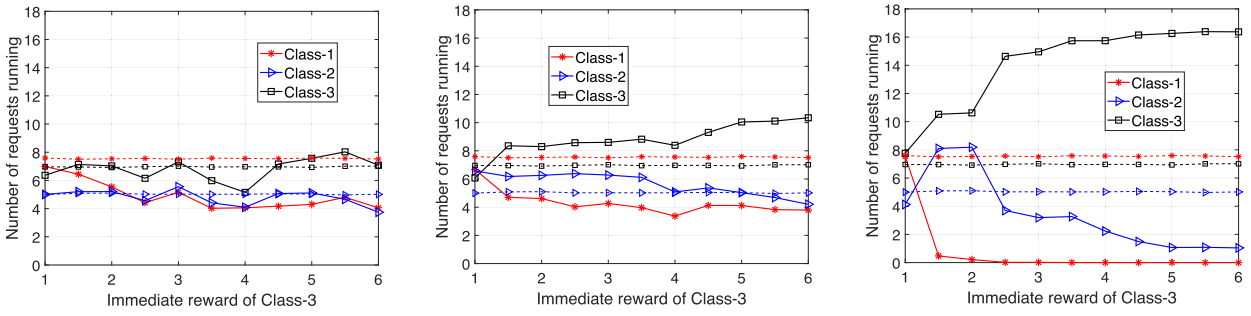


Fig. 10. The number of request running in the system of (a) Q-learning algorithm (10^6 iterations), (b) Q-learning algorithm (10^7 iterations), and (c) deep dueling algorithm (20,000 iterations) when the immediate reward of class-3 is varied. The dash lines are results of the greedy algorithm.

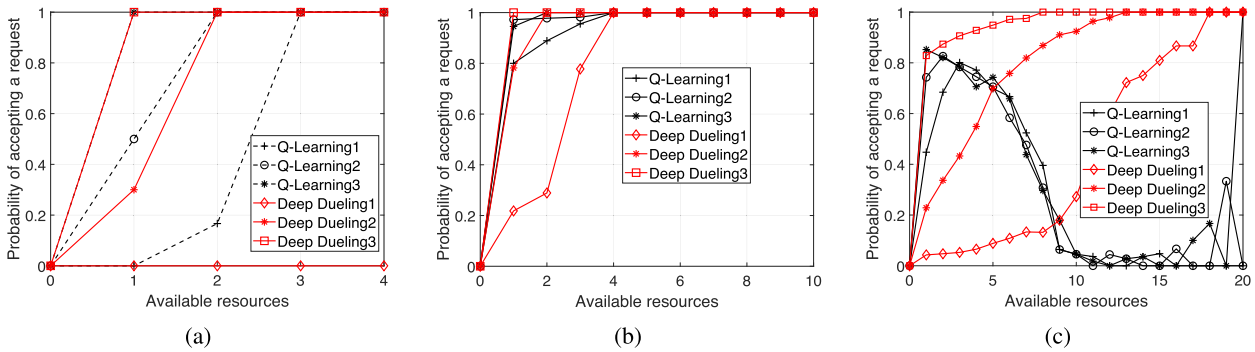


Fig. 11. The probabilities of accepting a request from classes when the maximum available resources of the system is (a) 4 times, (b) 10 times, and (c) 20 times of resources requested by a slice.

Clearly, in three cases, the deep dueling algorithm always obtains the best policy. In particular, it will reject almost requests from class-1 (lowest immediate reward) when there are few available resources in the system. When the available resources in the system increase, the probability of accepting a request from class-1 is also increased. Note that, when the maximum system resource capacity is large, i.e., 20 times of resources requested by a slice, the performance of the Q-learning is fluctuated as it cannot converge to the optimal policy event with 10^7 iterations.

2) Convergence of Deep Reinforcement Learning Approaches: Next, we show the learning process and the convergence of the deep reinforcement learning approaches, i.e., deep Q-learning, deep double Q-learning, and deep dueling, in different scenarios. As shown in Fig. 12(a), when the maximum radio, computing, storage resources are 400 Mbps, 8 CPUs, and 4 GB, respectively, the convergence rates of the three deep Q-learning algorithms are considerably higher than that of the Q-learning algorithm. Specifically, while the deep reinforcement learning approaches converge to the optimal value within 10,000 iterations, the Q-learning algorithm needs more than 10^6 iterations to obtain the optimal policy. This is stemmed from the fact that in the system under consideration, the state space is dimensional and the system dynamically changes over time. In Fig. 12(b), we show the convergence of the Q-learning and deep dueling algorithms in the first 20,000 iterations to clearly verify this observation. On the contrary, by implementing the neural network with

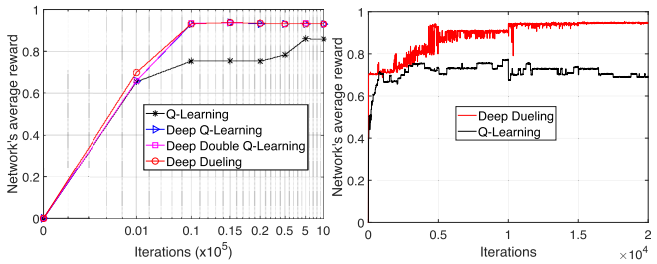


Fig. 12. The convergence of reinforcement learning algorithms when the radio, computing, storage resources are 400 Mbps, 8 CPUs, and 4 GB, respectively with (a) 10^6 iteration and (b) 20,000 iterations.

fully-connected layers, the deep reinforcement algorithms can efficiently reduce the curse of dimensionality, thereby improving the convergence rate.

We continue to increase the radio, storage, computing resources to 1 Gbps, 10 GB, and 20 CPUs, respectively. The arrival rates of classes are increased by 4 times, i.e., $\lambda_1 = 48$, $\lambda_2 = 32$, and $\lambda_3 = 40$ requests/hour, while the completion rates are equal to 2 requests/hour for all classes. As shown in Fig. 13(a), the performance of the deep reinforcement algorithms is significantly higher than that of the Q-learning algorithm. It is important to note that as the state space now is more complicated than in the previous case, the deep dueling algorithm obtains the optimal policy within 15,000 iterations,

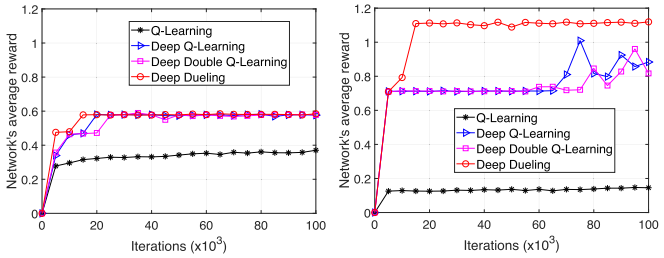


Fig. 13. The convergence of reinforcement learning algorithms when (a) the radio, computing, storage resources are 1 Gbps, 20 CPUs, and 10 GB, respectively and (b) the radio, computing, storage resources are 2 Gbps, 20 GB, and 40 CPUs, respectively.

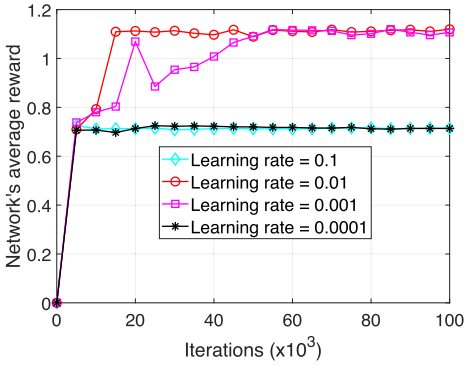


Fig. 14. The performance of deep dueling algorithm with different learning rates.

while the other two deep reinforcement learning approaches require more time to converge to the optimal policy.

We keep increasing the radio, storage, computing resources to 2 Gbps, 20 GB, and 40 CPUs, respectively and observe the convergence rates of the deep reinforcement algorithms as shown in Fig. 13(b). Clearly, as now the system is very complicated, the deep dueling can achieve the optimal policy within 20,000 iterations while the deep Q-learning and deep double Q-learning algorithms cannot converge to the optimal policy after 100,000 iterations. This is due to the fact that by decoupling the neural network into two streams, the deep dueling algorithm can significantly reduce the overestimation of the optimizer, i.e., stochastic gradient descent.

Next, we show the effects of the learning rate on the performance of the deep dueling algorithm. The learning rate is the most critical hyper-parameters to tune for training deep neural networks. If the learning rate is too slow, the training process is more reliable but requires a long time to converge to the optimal policy. In contrast, if the learning rate is too high, the algorithm may not converge to the optimal policy or even diverge. This is stemmed from the fact that the deep dueling algorithm uses the gradient descent method. If the learning rate is too large, gradient descent may overshoot the optimal point, and thus resulting in poor performance. This observation is proved by simulation results as shown in Fig. 14. Specifically, with the learning rate is 0.01, the deep dueling algorithm achieves the best performance in terms of the average reward and the convergence rate compared to other learning rates.

VII. Conclusion

In this paper, we have developed the optimal and fast network resource management framework which allows the network provider to jointly allocate multiple combinatorial resources (i.e., computing, storage, and radio) to different slice requests in a real-time manner. To deal with the dynamic and uncertainty of slice requests, we have adopted the semi-Markov decision process. Then, the reinforcement learning algorithms, i.e., Q-learning, deep Q-learning, deep double Q-learning, and deep dueling, have been employed to maximize the long-term average reward for the network provider. The key idea of the deep dueling algorithm is using two streams of fully connected hidden layers to concurrently train the value and advantage functions, thereby improving the training process and achieving the outstanding performance for the system. Extensive simulations have shown that the proposed framework using deep dueling can yield up to 40% higher long-term average reward with few thousand times faster compared with those of other network slicing approaches. Future works comprise considering the connectivity resources and the existence of multiple data centers in complex network slicing models by accommodating more states to the system state space. The performance of the proposed solution will be evaluated in terms of complexity and scalability. Moreover, the convergence rate and stability of the deep dueling algorithm will be improved by using the state-of-the-art deep neural networks.

Appendix A
The Proof of Theorem 1

For any $t \geq 0$, define the matrix $\mathbf{P}(t)$ by $\mathbf{P}(t) = (p_{\mathbf{s},\mathbf{s}'}(t)), \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}$. Denote by $\mathbf{Q}$, the matrix $\mathbf{Q} = q_{\mathbf{s},\mathbf{s}'}, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}$, where the diagonal elements $q_{\mathbf{s},\mathbf{s}}$ are defined by:

$$q_{\mathbf{s},\mathbf{s}} = -z_{\mathbf{s}}. \tag{42}$$

After that Kolmogoroff's forward differential equations can be written as $\mathbf{P}'(t) = \mathbf{P}(t)\mathbf{Q}$ for any $t \geq 0$. Hence, the solution of this system of differential equations is given by:

$$\mathbf{P}(t) = e^{t\mathbf{Q}} = \sum_{n=0}^{\infty} \frac{t^n}{n!}\mathbf{Q}^n, \quad t \geq 0. \tag{43}$$

The matrix $\overline{\mathbf{P}} = \overline{p}_{\mathbf{s},\mathbf{s}'}, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}$ can be reformulated as $\overline{\mathbf{P}} = \mathbf{Q}/z + \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Therefore, we have

$$\begin{aligned}\mathbf{P}(t) &= e^{t\mathbf{Q}} = e^{zt(\overline{\mathbf{P}}-\mathbf{I})} = e^{zt\overline{\mathbf{P}}}e^{-zt\mathbf{I}} = e^{-zt}e^{zt\overline{\mathbf{P}}} \\ &= \sum_{n=0}^{\infty} e^{-zt}\frac{(zt)^n}{n!}\overline{\mathbf{P}}^n.\end{aligned} \tag{44}$$

Based on conditioning on the number of Poisson events up to time t in the $\{\overline{X}(t)\}$ process, we have

$$P\{\overline{X}(t) = \mathbf{s}'|\overline{X}(0) = \mathbf{s}\} = \sum_{n=0}^{\infty} e^{-zt}\frac{(zt)^n}{n!}\overline{p}_{\mathbf{s},\mathbf{s}'}^{(n)}, \tag{45}$$

where $\overline{p}_{\mathbf{s},\mathbf{s}'}^{(n)}$ is the n-step transition probability of the discrete-time Markov chain $\overline{X}_n$. By recalling the Corollary 1, the proof is completed.

APPENDIX B
THE PROOF OF LEMMA 1

Let $\{A_n : n \geq 0\}$ be a sequence of matrices. We have $\lim_{n\to\infty} A_n = A$ if $\lim_{n\to\infty} A_n(\mathbf{s}'|\mathbf{s}) = (\mathbf{s}'|\mathbf{s})$ for each $(\mathbf{s}, \mathbf{s}') \in \mathcal{S} \times \mathcal{S}$. We now consider the Cesaro limit which is defined as follows. We say that A is the Cesaro limit (of order one) of $\{A_n : n \geq 0\}$ if

$$\lim_{n\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} A_n = A, \tag{46}$$

and write

$$C - \lim_{N\to\infty} A_N = A \tag{47}$$

to distinguish this as a Cesaro limit. We then define the limiting matrix $\overline{P}$ by

$$\overline{P} = C - \lim_{N\to\infty} P^N. \tag{48}$$

In component notation, where $\overline{p}(\mathbf{s}'|\mathbf{s})$ denotes the $(\mathbf{s}'|\mathbf{s})$-th element of $\overline{P}$, this means that, for each $\mathbf{s}$ and $\mathbf{s}'$, we have

$$\overline{p}(\mathbf{s}'|\mathbf{s}) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} p^{n-1}(\mathbf{s}'|\mathbf{s}), \tag{49}$$

where p^{n-1} denotes a component of P^{n-1} and $p^0(s'|s)$ is a component of an $\mathcal{S} \times \mathcal{S}$ identity matrix. As P is aperiodic, $\lim_{N\to\infty}$ exists and equals to $\overline{P}$.

REFERENCES

[1] *5G Will Start Moving the Needle on Mobile Data Growth in 2020: CISCO VNI*. Accessed: Mar. 15, 2019. [Online]. Available: https://disruptive.asia/5g-mobile-data-growth-2020-cisco-vni/

[2] *NGMN 5G White Paper*, Next Gener. Mobile Netw., Frankfurt, Germany, 2015.

[3] A. Manzalini *et al.*, "Towards 5G software-defined ecosystems," IEEE SDN, White Paper, 2016.

[4] H. Zhang, N. Liu, X. Chu, K. Long, A. Aghvami, and V. C. M. Leung, "Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 138–145, Aug. 2017.

[5] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network slicing as a service: Enabling enterprises' own software-defined cellular networks," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 146–153, Jul. 2016.

[6] M. Jiang, M. Condoluci, and T. Mahmoodi, "Network slicing management & prioritization in 5G mobile systems," in *Proc. 22nd Eur. Wireless Conf.*, Oulu, Finland, May 2016, pp. 1–6.

[7] H. M. Soliman and A. Leon-Garcia, "QoS-aware frequency-space network slicing and admission control for virtual wireless networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[8] J. Zheng, P. Caballero, G. de Veciana, S. J. Baek, and A. Banchs, "Statistical multiplexing and traffic shaping games for network slicing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2528–2541, Dec. 2018.

[9] S. D'Oro, F. Restuccia, T. Melodia, and S. Palazzo, "Low-complexity distributed radio access network slicing: Algorithms and experimental results," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2815–2828, Dec. 2018.

[10] P. L. Vo, M. N. H. Nguyen, T. A. Le, and N. H. Tran, "Slicing the edge: Resource allocation for RAN network slicing," *IEEE Wireless Commun. Lett.*, vol. 7, no. 6, pp. 970–973, Dec. 2018.

[11] Y. Sun, M. Peng, S. Mao, and S. Yan, "Hierarchical radio resource allocation for network slicing in fog radio access networks," *IEEE Trans. Veh. Technol.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8630653

[12] J. Ni, X. Lin, and X. S. Shen, "Efficient and secure service-oriented authentication supporting network slicing for 5G-enabled IoT," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 644–657, Mar. 2018.

[13] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Atlanta, GA, USA, May 2017, pp. 1–9.

[14] 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. (INFOCOM)*, Atlanta, GA, USA, May 2017, pp. 1–9.

[15] A. Aijaz, "Hap–SliceR: A radio resource slicing framework for 5G networks with haptic communications," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2285–2296, Sep. 2017.

[16] A. Aijaz, "Radio resource slicing in a radio access network," U.S. Patent 15 441 564, Nov. 2, 2017.

[17] A. B. Koehler, R. D. Snyder, and J. K. Ord, "Forecasting models and prediction intervals for the multiplicative Holt–Winters method," *Int. J. Forecasting*, vol. 17, no. 2, pp. 269–286, Jun. 2001.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning—An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[19] An Introduction to Network Slicing. *The GSM Association*. Accessed: Mar. 15, 2019. [Online]. Available: https://www.gsma.com/futurenetworks/wp-content/uploads/2017/11/GSMA-An-Introduction-to-Network-Slicing.pdf

[20] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.

[21] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Hoboken, NJ, USA: Wiley, 1994.

[22] N. V. Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Real-time network slicing with uncertain demand: A deep learning approach," presented at the IEEE Int. Conf. Commun. (ICC), Shanghai, China, May 2019.

[23] ETSI. *Network Functions Virtualisation (NFV); Use Cases*. Accessed: Mar. 15, 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.02.01_60/gr_nfv001v010201p.pdf

[24] ETSI. *Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support With ETSI NFV Architecture Framework*. Accessed: Mar. 15, 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf

[25] ETSI. *Network Functions Virtualisation (NFV); NFV Performance & Portability Best Practises*. Accessed: Mar. 15, 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-PER/001_099/001/01.01.01_60/gs_nfv-per001v010101p.pdf

[26] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.

[27] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. CloudNet*, Niagara Falls, ON, Canada, Oct. 2015, pp. 171–177.

[28] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, "Joint virtual network function placement and routing of traffic in operator networks," in *NetSoft*, 2015, pp. 1–5. [Online]. Available: http://networks.cs.ucdavis.edu/abgupta/technical_report_vnf_placement_and_routing_abhishek_gupta.pdf

[29] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 533–546, Aug. 2016.

[30] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. Kozat, "A resource allocation framework for network slicing," in *Proc. IEEE INFOCOM*, Honolulu, HI, USA, Apr. 2018, pp. 15–19.

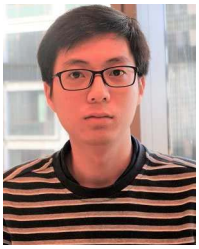
[31] B. Awerbuch, Y. Azar, and S. Plotkin, "Throughput-competitive on-line routing," in *Proc. 34th IEEE Ann. Found. Comput. Sci.*, Nov. 1993, pp. 32–40.

[32] N. Bihannic, T. Lejkin, I. Finkler, and A. Frerejean, "Network slicing and blockchain to support the transformation of connectivity services in the manufacturing industry," *IEEE Softw.* Accessed: Mar. 15, 2019. [Online]. Available: https://sdn.ieee.org/newsletter/march-2018/network-slicing-and-blockchain-to-support-the-transformation-of-connectivity-services-in-the-manufacturing-industry.

[33] 5G Network Slicing for Vertical Industries. *Global Mobile Suppliers Association*. [Online]. Available: https://www.huawei.com/minisite/5g/img/5g-network-slicing-for-vertical-industries-en.pdf

[34] T. P. Lillicrap *et al.* (2016). "Continuous control with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1509.02971

[35] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
[36] R. G. Gallager, *Discrete Stochastic Processes*. London, U.K.: Kluwer, 1995.
[37] H. C. Tijms, *A First Course in Stochastic Models*. Hoboken, NJ, USA: Wiley, 2003.
[38] L. Kallenberg. *Markov Decision Process*. Accessed: Mar. 15, 2019. [Online]. Available: http://www.math.leidenuniv.nl/%7Ekallenberg/Lecture-notes-MDP.pdf
[39] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[40] A. Geramifard, T. J. Walsh, S. Tellex, G. Chowdhary, N. Roy, and J. P. How, "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Found. Trends Mach. Learn.*, vol. 6, no. 4, pp. 375–451, 2013.
[41] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double *Q*-learning," in *Proc. AAAI*, 2016, pp. 2094–2100. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389/11847
[42] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. (2016). "Dueling network architectures for deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1511.06581
[43] M. Abadi *et al.* (2016). "TensorFlow: Large-scale machine learning on heterogeneous distributed systems." [Online]. Available: https://arxiv.org/abs/1603.04467
[44] D. Sattar and A. Matrawy. (2018). "Optimal slice allocation in 5G core networks." [Online]. Available: https://arxiv.org/abs/1802.04655
[45] X. Chen *et al.* (2018). "Reinforcement learning based QoS/QoE-aware service function chaining in software-driven 5G slices." [Online]. Available: https://arxiv.org/abs/1804.02099
[46] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[47] B. Han, J. Lianghai, and H. D. Schotten, "Slice as an evolutionary service: Genetic optimization for inter-slice resource management in 5G networks," *IEEE Access*, vol. 6, pp. 33137–33147, 2018.
[48] F. B. Uz. *GPUs vs CPUs for Deployment of Deep Learning Models*. Accessed: Mar. 15, 2019. [Online]. Available: https://azure.microsoft.com/en-au/blog/gpus-vs-cpus-for-deployment-of-deep-learning-models/
[49] F. Kurtz, C. Bektas, N. Dorsch, and C. Wietfeld, "Network slicing for critical communications in shared 5G infrastructures—An empirical evaluation," in *Proc. IEEE NetSoft*, Montreal, QC, Canada, Jun. 2018, pp. 393–399.



Nguyen Van Huynh received the B.E. degree in electronics and telecommunications engineering from the Hanoi University of Science and Technology (HUST), Vietnam, in 2016. He is currently pursuing the Ph.D. degree with the University of Technology Sydney (UTS), Australia. From 2017 to 2018, he was a Research Assistant with Nanyang Technological University (NTU), Singapore. His research interests include wireless powered communications, green communications, and optimization problems for wireless communication networks.

Dinh Thai Hoang (M'16) received the Ph.D. degree in computer science and engineering from Nanyang Technological University, Singapore, in 2016. He is currently a Faculty Member with the School of Electrical and Data Engineering, University of Technology Sydney, Australia. His research interests include emerging topics in wireless communications and networking, such as ambient backscatter communications, vehicular communications, cybersecurity, the IoT, and 5G networks. He was an Exemplary Reviewer of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS in 2017 and 2018 and the IEEE TRANSACTIONS ON COMMUNICATIONS in 2018. He is also an Editor of the IEEE WIRELESS COMMUNICATIONS LETTERS and IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING.

Diep N. Nguyen received the M.E. degree in electrical and computer engineering from the University of California at San Diego (UCSD) and the Ph.D. degree in electrical and computer engineering from The University of Arizona (UA). He is currently a Faculty Member with the Faculty of Engineering and Information Technology, University of Technology Sydney (UTS). Before joining UTS, he was a DECRA Research Fellow with Macquarie University and a Member of Technical Staff at Broadcom, CA, USA, ARCON Corporation, Boston, MA, USA, consulting the Federal Administration of Aviation on turning detection of UAVs and aircraft, and the U.S. Air Force Research Lab on Anti-Jamming. His recent research interests are in the areas of computer networking, wireless communications, and machine learning application, with an emphasis on systems' performance and security/privacy. He received several awards from LG Electronics, UCSD, UA, the U.S. National Science Foundation, and the Australian Research Council.

Eryk Dutkiewicz received the B.E. degree in electrical and electronic engineering and the M.Sc. degree in applied mathematics from The University of Adelaide in 1988 and 1992, respectively, and the Ph.D. degree in telecommunications from the University of Wollongong in 1996. His industrial experience includes management of the Wireless Research Laboratory, Motorola, in early 2000s. He is currently the Head of the School of Electrical and Data Engineering, University of Technology Sydney, Australia. He also holds a professorial appointment at Hokkaido University, Japan. His current research interests cover 5G and the IoT networks. He is a Senior Member of the IEEE.