



Yan, M., Feng, G., Zhou, J., Sun, Y. and Liang, Y.-C. (2019) Intelligent resource scheduling for 5G radio access network slicing. *IEEE Transactions on Vehicular Technology*, 68(8), pp. 7691-7703. (doi: [10.1109/TVT.2019.2922668](https://doi.org/10.1109/TVT.2019.2922668))

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

<http://eprints.gla.ac.uk/212845/>

Deposited on: 24 April 2020

Enlighten – Research publications by members of the University of Glasgow  
<http://eprints.gla.ac.uk>

# Intelligent Resource Scheduling for 5G Radio Access Network Slicing

Mu Yan, Gang Feng, *Senior Member, IEEE*, JianHong Zhou, *Member, IEEE*, Yao Sun, and Ying-Chang Liang, *Fellow, IEEE*

**Abstract**—It is widely acknowledged that network slicing can tackle the diverse use cases and connectivity services of the forthcoming next generation mobile networks (5G). Resource scheduling is of vital importance for improving resource-multiplexing gain among slices while meeting specific service requirements for Radio Access Network (RAN) slicing. Unfortunately, due to the performance isolation, diversified service requirements and network dynamics (including user mobility and channel states, etc.), resource scheduling in RAN slicing is very challenging. In this paper, we propose an intelligent resource scheduling strategy (iRSS) for 5G RAN slicing. The main idea of iRSS is to exploit a collaborative learning framework which consists of deep learning (DL) in conjunction with Reinforcement Learning (RL). Specifically, DL is used to perform large time-scale resource allocation, while RL is used to perform on-line resource scheduling for tackling small time-scale network dynamics, including inaccurate prediction and unexpected network states. Depending on the amount of available historical traffic data, iRSS can flexibly adjust the significance between the prediction and on-line decision modules for assisting RAN in making resource scheduling decisions. Numerical results show that the convergence of iRSS satisfies on-line resource scheduling requirement and can significantly improve resource utilization while guaranteeing performance isolation between slices, compared with other benchmark algorithms.

**Index Terms**—RAN slicing, Resource Scheduling, Deep Learning, Reinforcement Learning.

## I. INTRODUCTION

The forth-coming generation mobile network (5G) is expected to significantly improve the efficiency of mobile networks to meet diverse use cases and service requirements. Current one size fits all network architecture is no more efficient for the multi-service oriented 5G. Virtualizing the 5G mobile network to make it programmable in a flexible way is of paramount importance for a cost-effective solution to address this issue. In this regard, network slicing is an emerging and valid solution to realize the service-oriented 5G vision. In network slicing, physical infrastructure is sliced into multiple isolated logical networks, with aim of supporting a wide range of verticals and use cases with a diverse set of performance and service

M. Yan, G. Feng, J.H. Zhou, Y. Sun and Y.-C. Liang are with the National Key Laboratory of Science and Technology on Communications, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the Center for Intelligent Networking and Communications (CINC), University of Electronic Science and Technology of China, Chengdu 611731, China. J.H. Zhou is also with School of Computer and Software Engineering, Xihua University, Chengdu 610039, China. G. Feng is the corresponding author (email: fenggang@uestc.edu.cn). This work was supported by the National Science Foundation of China under Grant number 61631005, and the Research and Development Program in Key Areas of Guangdong Province under Grant number 2018B010114001.

requirements. Software-defined network (SDN) and network function virtualization (NFV) are used to achieve creation of slices [1], such that 5G networks gradually evolve to a flexible and programmable network architecture [2].

While the virtualization of 5G core networks has been widely studied, the realization of Radio Access Network (RAN) slicing is at its infancy [3]. One key issue in RAN slicing is resource scheduling which appropriately allocates limited resources to individual users with diverse quality of service (QoS) requirements according to the traffic variations and network state dynamics. Compared with that in core network slicing [4], resource scheduling in RAN slicing is much more challenging due to the radio channel and user mobility involved.

The main idea of resource scheduling in traditional RANs enables a rigid way to exploit resources among users, so as to achieve a high spectrum multiplexing gain with spectrum sharing [5] [6] [7]. As RAN slicing provides a flexible and controllable architecture, the main design objective of a resource scheduling strategy (RSS) for RAN slicing is to flexibly and adaptively share RAN resources among slice owners (or tenants), so that the RAN infrastructure can be efficiently utilized. In the meantime, it is necessary to maintain a certain degree of slice independence (i.e., performance isolation and functional isolation), so that the tenants can maintain full control of their slices to be tailored to meet their service requirements. Without appropriate slice isolation, service interruptions may happen, leading to poor performance in the multi-service RAN slicing environment. Obviously, the RSS for RAN slicing is much more complicated compared with that in traditional RANs, and the existing RSSs for conventional RANs cannot be applicable to RAN slicing. Therefore, it is imperative to develop new RSSs dedicated to RAN slicing, with aim to maximize resource utilization subject to slice isolation requirements. Fortunately, under such a very complex and dynamic network environment, recent emerging machine learning tools that interact with surrounding environment can provide an effective way to address this challenging problem.

In this paper, we propose an intelligent resource scheduling strategy (iRSS) for RAN slicing, which is embedded in a collaborative learning framework. Both deep learning (DL) and reinforcement learning (RL) are incorporated and work in a collaborative way, to deal with both large and small time-scales network and traffic dynamics. In detail, long short-term memory (LSTM) [8], is used to explore the regularity of data traffic, and perform large time-scale resource allocation of RAN slices. In addition, for coping with inaccurate prediction

and unexpected network states in small time-scale, distributed architecture based asynchronous advantage actor-critic (A3C) algorithm [9] is exploited for performing on-line resource scheduling of RAN slices. It is known that the prediction accuracy and granularity of DL mainly depend on the volume of available effective data [8]. In comparison, on-line RL does not heavily rely on available data volume, and can quickly response to dynamic environment. Nevertheless, on-line RL generally cannot provide satisfactory performance at the beginning of a learning process [10]. In this regard, we propose to exploit both technologies for RAN slicing RSS in a collaborative way. Specifically, the time is divided into prediction windows (PWs), and DL is used in each PW to predict traffic volume for the next PW. In the meantime, inside each PW, RL is used for performing on-line resource scheduling. The significance between the prediction by DL and the real-time resource scheduling by RL can be dynamically adjusted, resulting in a total programmable iRSS. In detail, if the historical traffic data collected for training the deep neural network is insufficient, the decision-making can depend more on on-line RL solution through reducing the confidence level of prediction. With the progress of learning process, more data can be collected and thus the significance of prediction can be increased. Our contribution of this paper can be summarized as follows.

- To the best knowledge of the authors, this is the first work to exploit DL and RL in a collaborative way for addressing RAN resource scheduling. The significance between prediction and on-line decision modules can be dynamically adjusted for assisting RAN in making accurate decisions.
- We propose an A3C algorithm for the small timescale resource scheduling. In A3C, the *Actor* provides resource scheduling strategies for slices in advance through policy network, and the *Critic* uses *TD-error* to update the value network to make the decisions more appropriately. We have demonstrated through simulations that the collaboration of *Actor* and *Critic* in A3C can meet the real-time scheduling requirement, and can significantly improve the performance when compared with other benchmark learning algorithms.
- We implement parallel computing for individual slices in the A3C algorithm, such that the two target networks (*i.e.*, the Actor and Critic) can be maintained independently for different slices in parallel. By this way, A3C can well capture the regularities of service requests of individual slices, and thus can help them make appropriate decisions in the RSS.
- We transfer the control and responsibility from the mobile virtual network operator to individual slices. This stimulates the slice abilities of self-learning and self-control with reduced signaling interactions. Furthermore, to address the network information exchange among slices, we introduce the sharing *account book* in the *Critic* process, guiding the decisions to be made along the feasible way.

The rest of the paper is organized as follows. The network

model is presented in Section III. In Section IV, we formulate the problem of resource scheduling for RAN slicing as optimization problems from perspectives of large and small timescales respectively, and analyze the computational complexity. Then in Section V, we present the periodic traffic prediction by using LSTM method, and propose a distributed architecture for on-line resource scheduling. In Section VI, we model the problem of resource scheduling of small-timescale as a continuous Markov Decision Process (MDP), which is solved by using the parallel computing based asynchronous advantage actor-critic (A3C) method. In Section VII, we present the numerical results as well as discussions, and finally conclude the paper in Section VIII.

## II. RELATED WORK

In recent years, many researchers have investigated RAN sharing in 5G networks. 3GPP Rel. 15 also specifies two RAN sharing schemes, *i.e.*, multi-operator core networks (MOCN) and multi-operator RAN (MORAN) [11], and they have been widely referred in related research work [3] [12] [13].

Although separated core networks are implemented in both MOCN and MORANs for each operator, MOCN fully shares the spectrum resources among multiple operators while MORAN allocates dedicated spectrum for each operator. Thus, the resource allocation in traditional RAN indeed belongs to MOCN scheme while RAN slicing belongs to MORAN scheme. In the following, we review the major related work on RAN RSS for traditional RAN and RAN slicing respectively, followed by recent emerging machine learning based RSS.

### A. RSSs for Traditional RAN

Generally, RSSs in traditional RAN simply pool the spectrum resources and share them by catering various services with diverse requirements. As the RSSs do not consider to pre-reserve resources for future service requests, the performance isolation cannot be guaranteed. The major concern of the existing strategies of [5] [6] focuses on the design of efficient sharing of the radio resources among different users while guaranteeing the requirements of services. Thus, a high multiplexing gain can usually be obtained by exploiting the full spectrum sharing in traditional RANs. Unfortunately, in the dynamic multi-service environment of RAN slicing, service interruption may happen without guaranteeing the performance isolation if these RSSs are used. Therefore, it is necessary to develop new RSS which can make resource reservation for individual services, to guarantee a certain level of performance isolation for RAN slicing.

### B. RSSs for RAN Slicing

While the RSSs in traditional RAN have been widely studied, RSS for RAN slicing has just become one of the research focuses in 5G network research recently. The state-of-the-art on RAN slicing aims at scheduling resources to customized services for guaranteeing the isolation among slices (*e.g.*, functional isolation, performance isolation). In [13] [14], full isolation is considered to accommodate the need

for slice customizability. By this way, the resource Docker is cut into small compartments, resulting in a strict isolation among slices. Obviously, the spectrum multiplexing efficiency is low due to physical isolation of resources in this scheme. As an improvement, the authors of [3] propose a flexible multi-service mobile network architecture, aiming to find a balancing between providing functional isolation among slices and facilitating efficient sharing of RAN resources. However, under a very complex and dynamic environment of RAN slicing, it is necessary for the service-orientation RSS to adopt an intelligent solution for providing an on-line solution with a certain degree of generalization.

### C. Machine Learning based RSSs

The recent rapid development of machine learning technologies provide a novel way for designing RSS for RANs. At the edge of wireless networks, artificial intelligence (AI) (e.g., machine learning) has been applied to provide prediction and fast decision making in uncertain network environments. Deep learning such as recurrent neural network (RNN) can dig out the intrinsic correlation between data, so as to make prediction or classification, which could be an essential component of RSS. The authors of [15] exploit the RNN for traffic forecasting, and thus achieve performance improvement. On the other hand, reinforcement learning (RL) learns to control a system so as to maximize a numerical performance measure that expresses a long-term objective based on a Markov Decision Process (MDP), by dynamically interacting with the environment. AlphaGo and AlphaGo Zero, *i.e.*, [16] [17] have leveraged RL for addressing the Go game which is modelled as an MDP problem, and have notable achievements in the Go games of Machine vs Man. In [6], we use the multi-agent RL to address the multi-RAT access, where RL performs better in a time-varying network environment when compared with other benchmark solutions. However, RL algorithms such as conventional Q-learning and Monte Carlo Tree Search (MCTS) may be inefficient when the action (or decision) space is too large or the action space is continuous, since the computational complexity of these RL algorithms exponentially grows with the size of action space. In this regard, the authors of [18] and [19] model the RSS problem as MDPs, and respectively adopt discrete-action based Q-learning and continuous-action based Actor-Critic (AC) to address this problem. In addition, the AC algorithm has demonstrated a strong potential in solving the continuous action space based MDP problem. In general, by exploiting historical traffic data, deep learning can provide prediction results, assisting RAN in allocating resources to users in advance. By interacting with the network environment, RL provides an effective approach for RAN to adapt to network dynamics.

## III. NETWORK MODEL

### A. Network Topology

In this paper, we consider multiple network slices that are deployed on a substrate mobile network enabled by Software Defined Network (SDN) / Network Function Virtualization (NFV). Fig. 1 shows an exemplary deployment scenario of

network slices, which consists of the RANs and core network (CN), where the physical network infrastructure is logically split into multiple virtual networks (slices) to support diverse mobile services. Let RAN be composed by a set of  $\mathcal{N}$  base stations (BS) deployed in a geographic area, where the spectrum resources are aggregated to form a resource pool. We denote by  $\mathcal{M}(\mathcal{M} = \{1, \dots, m, \dots, M\})$  the set of slices sharing the RAN, and by  $\mathcal{U}_m, m \in \mathcal{M}$  the subset of the users belonging to slice  $m$ . A slice may cover multiple BSs, and the corresponding transmission resources of the BSs are allocated to individual slices. Let there be some distributed data collector units (DCUs) integrated into nearby data centers (DC), which are able to record the aggregated traffic information of individual slices. Let there be a sharing *account book* used to record and share some necessary information among slices (e.g., states of network, slices, user behaviors), and each slice has the authority to modify and maintain this account book. This decentralization mechanism transfers the control and responsibility from the mobile virtual network operator (MVNO) to individual slices, with the aim of stimulating the slice abilities of self-learning and self-control with reduced signaling interactions.

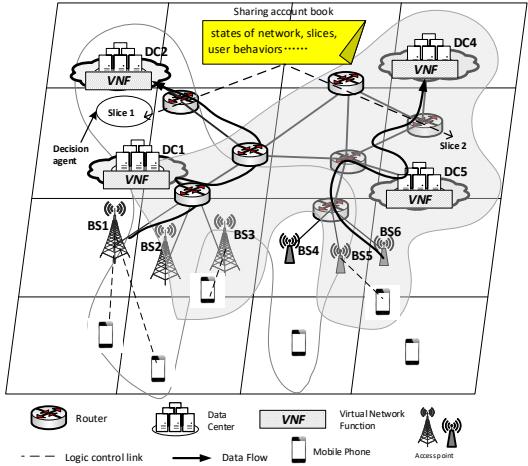


Fig. 1: Network topology

### B. Traffic Model

Two types of resource provisioning have been widely used [20]: (1) RAN slices request spectrum resources based on the amount of spectrum resources, and (2) resource provisioning is based on transmission rate. Specifically, resource-based provisioning defines the resource scheduling for a slice in terms of a fraction of the total resources. Rate-based provisioning defines resource scheduling in terms of the aggregate throughput. Let  $\mathcal{M}^I \subset \mathcal{M}$  denote the subset of the slices (or tenants) requiring resource-based provisioning, and  $\mathcal{M}^{II} \subset \mathcal{M}$  denote the subset of the slices requiring rate-based slices provisioning. Let the quality of service (QoS) requirement set of slice  $m$  be  $\mathcal{G}_m(t)$  which is represented by a three-tuple  $\{r_m(t), h_m(t), \tau_m(t)\}, m \in \mathcal{M}$  at time  $t$ , where  $r_m$ ,  $h_m$  and  $\tau_m$  represent the assigned amount of spectrum resources, the threshold of resource requirement, and the

required transmission time interval (TTI) length, respectively. Note that the length of TTI can be dynamically set according to the specific requirement of slices of the mobile networks, such as 5G new radio (NR) framing [21].

### C. RAN Slicing Model

To realize network slicing, certain degree of resource isolation between slices must be enforced, so that the offered quality of service of a slice will not be influenced by the traffic load variation of other slices [14]. In this work, we focus on the performance isolation, and assume that the functional isolation among slices is already guaranteed as that in Orion [3].

**Definition 1.** Let  $P_m$  be the performance metric (e.g., QoS requirements, throughput, etc.) of slice  $m$  at a given time. Assuming that the resource requirements of slices change from  $\{r_1, \dots, r_m, \dots, r_M\}$  to  $\{r_1^+, \dots, r_m^+, \dots, r_M^+\}$  from  $t$  to  $t+1$ . We define the strict performance isolation for slice  $m$  as that there is no performance deterioration of slice  $m$  because of any changes on the resources requirements of other slices. This strict performance isolation is expressed by

$$P_m^t(r_1, \dots, r_m, \dots, r_M) \leq P_m^{t+1}(r_1^+, \dots, r_m^+, \dots, r_M^+).$$

Because of the possible unknown changes of other slices requirements, it is necessary for RAN to make reservations for the aggregated resources of slices in advance, so as to support a certain degree of performance isolation across slices. On this basis, we partition the total spectrum resources  $\Theta$  ( $\Theta = 1$ ) into the fraction of dedicated resources  $\Theta_m^r, m \in M$  and the fraction of shared resources  $\Theta^s = \Theta - \sum_{m \in M} \Theta_m^r$ . We define the performance isolation degree (PID) of a slice as the ratio of the time duration in which the strict performance isolation can be guaranteed over a given time period. It is obvious that PID increases with the amount of dedicated resources. In addition, when considering RAN slicing in a single-cell scenario, we simply assign orthogonal resources to different users to avoid co-channel interferences, and thus to guarantee the physical isolation of inter-slice. We illustrate the relationship between dedicated and shared resources in Fig. 2. Finally, for improving the clarity, we summarize the notations and variables used in this paper in Table I.

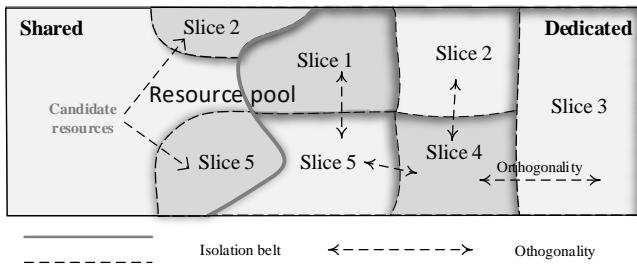


Fig. 2: Dedicated and shared resources in RAN slicing

TABLE I: Main Parameters and Variables

Symbol	Description
$\Theta$	the fraction of all of the resources
$\Theta_m^r$	the fraction of reserved resources for slice $m$
$\Theta^s$	the fraction of shared resources among slices
$M = \{1, \dots, m, \dots, M\}$	set of slices
$\mathcal{U}$	set of users sharing the network
$\mathcal{U}_m, m \in M$	subsets of users belonging to each slice
$\mathcal{M}^I \subset \mathcal{M}$	slices which require resource-based slices provisioning
$\mathcal{M}^{II} \subset \mathcal{M}$	slices which require rate-based slices provisioning
$\mathcal{G}(t), m \in M$	the requirement set of slice $m$ at time slot $t$
$T_\Delta$	Prediction Window (PW)
$\mathcal{D}_m(t)$	the function of the difference between before and after resource reconfiguration
$\mathbb{B}(\mathcal{D}_m(t))$	Bonus incurred for that reconfiguration is conducted for slice $m$ .
$r_m(t), h_m(t)$	assigned spectrum resources, threshold of resource requirement
$\tau_m$	the required transmission time interval (TTI) length

### IV. PROBLEM FORMULATION OF RSS

In the following, we formulate the problems of resource scheduling of large and small time-scales respectively. Let a prediction window (PW) have  $T_\Delta$  decision time intervals (DTIs), and each DTI have one or several TTIs. Specifically, we respectively perform resource allocation of large timescale in the next PW, and the resource scheduling of small timescale in every DTI. For periodic traffic prediction in large time-scale, the shapes of traffic volume can be derived by minimizing the mean-square-error (MSE) between the predicted value  $\tilde{r}_m(t)$  and the actual traffic volume  $r_m(t)$ , which is formulated as follows.

#### Problem 1 Periodic Traffic Prediction for Slices

$$\arg \min_{\tilde{r}_m(t)} \frac{1}{T_\Delta} \sum_{t \in T_\Delta} |r_m(t) - \tilde{r}_m(t)|^2, \quad (1)$$

$$\text{s.t. } \sum_{m \in \mathcal{M}} \tilde{r}_m(t) = \Theta^r \leq \Theta. \quad (1.1)$$

We next formulate the on-line resource scheduling problem. Our design objective is to minimize the overall resource consumption of slices in RANs while guaranteeing the required performance isolation degree. There are two possible cases in the real-time resource scheduling: (1) The amount of the required resources  $r_m(t)$  exceeds the predicted value  $\tilde{r}_m(t)$  derived from Problem 1, i.e.,  $r_m(t) > \tilde{r}_m(t)$ . Thus slice  $m$  may need to request more resources from the shared resource pool. In addition, we assume that in order not to compromise the QoS of the majority of ongoing traffic sessions, new arrival flows need to wait until more resources are available. (2)  $r_m(t) \leq \tilde{r}_m(t)$  and in this case the pre-assigned resources in slices remain unchanged (i.e., released) for a long-term performance isolation.

Frequent resource requesting based on the instantaneous resource demand may cause service interruption of other slices and a certain reconfiguration overhead. The robustness of resource configuration for maintaining a certain degree of isolation and thus reducing the reconfiguration overhead

require a prospective on-line resource scheduling strategy. Let  $\mathcal{D}_m(t) = r_m(t) - r_m(t - \tau_m)$  be the difference between the amount of resources before and after resource reconfiguration for slice  $m$  at every decision time interval  $\tau_m$ , and  $\Omega_m(\mathcal{D}_m = 0)$  be a counter used to record the length of isolation time. Thus, we formulate the on-line resource scheduling of small timescale as follows.

### Problem 2 On-line Resource Scheduling

$$\text{Min}_{\mathcal{M}, t_\Delta} \sum_{m \in M} \sum_{t \in T_\Delta} r_m(k), \quad (2)$$

$$\text{s.t.} \quad \sum_{m \in M} r_m(t) \leq \Theta, \quad (2.2)$$

$$r_m(t) > h_m(t), \forall m \in \mathcal{M}, \quad (2.1)$$

$$\Omega_m(\mathcal{D}_m = 0) \geq T_m^{th}, \quad (2.3)$$

where  $T_m^{th}$  represents the threshold of the length of isolation duration. Specifically, for slices belonging to set  $\mathcal{M}^{II}$ ,  $h_m(t)$  varies with the signal-to-interference plus noise ratio (SINR) which is translated from the required transmission rate  $R_m^{th}$ . Therefore, for  $m \in \mathcal{M}^{II}$ ,  $h_m(t) = R_m^{th}/\log(1 + p_m g_m^2/N_0)$ , where  $g_m$  represents the channel gain,  $p_m$  is the transmission power which is fixed in this work, and  $N_0$  is the variance of white Gaussian channel noise.

This RSS problem in RAN slicing is a variant of the Multiple Choice dimension Knapsack Problem, which is known to be equivalent to an NP-hard problem [22]. More importantly, we target at a long-term optimal solution, and thus it is infeasible to use static optimization technique to solve the prediction and on-line resource scheduling problems. We thus resort to machine learning technique to address the RSS problem.

## V. COLLABORATIVE LEARNING FRAMEWORK OF iRSS

Our proposed iRSS for RAN slicing is embedded in a collaborative learning framework. Both DL and RL are incorporated in the framework and work in a collaborative way, so as to tackle both large and small time-scales network dynamics. Fig. 3 illustrates the collaboration of these two algorithms, where we use the confidence level  $\chi(\chi \in (0, 1))$  for intuitively describing the respective significance of DL and RL. In other words, we can adjust  $\chi$  according to the contribution from the DL and RL respectively to the decision-making process. The intuition is as follows. At the beginning of the learning process, there is no sufficient data in the DCU for training the LSTM, resulting in inaccurate prediction results. Accordingly  $\chi$  can be set smaller such that the decisions is mainly dependent on RL for performing real-time resource scheduling. With the progress of leaning process, more data is collected by the DCU, and  $\chi$  can thus be set bigger as the prediction results by DL become more credible. This dynamic adjustment process enables the network to well leverage the advantages of both DL and RL for making better scheduling strategies for individual slices. In extreme case, if the performance given by DL is too poor, iRSS can make decision according to the results of RL only( $\chi = 0$ ).

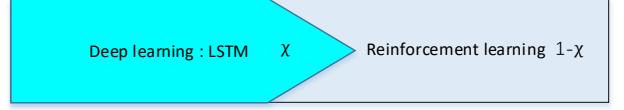


Fig. 3: Collaboration of DL and RL in CoLF

### A. Using LSTM Recurrent Neural Network for Periodic Traffic Prediction

Prediction leverages the regularity of historical traffic data to allocate resources for individual slices in advance. We aim to find a value of  $\tilde{r}_m(t)$  in (1) that can minimize the MSE for predicting the traffic volume of the next PW by using the traffic records collected in current and previous PWs.

Usually a huge amount of traffic data (e.g., data generated in one month) can be used for an accurate prediction. However, due to the *gradient explosion* or *gradient vanish* phenomenon in the process of back propagation through time (BPTT), standard recurrent neural networks (RNNs) fail in learning when the time lags are greater than 5-10 discrete time steps [23]. Long Short-Term Memory (LSTM) is a variant of RNN, capable of finding the embedded characteristics and leveraging the long-time dependency in the sequence [2]. As shown in Fig. 4, under the iRSS decision framework, we employ LSTM algorithm to make traffic volume prediction for performing the large time scale resource allocation in the next PW. The LSTM architecture consists of a set of recurrently connected subnets, known as memory blocks. These blocks can be deemed as a differentiable version of the memory chips in a digital computer. Each block contains one or more self-connected memory cells and three multiplicative units-the input, output and forgets gates-that provide continuous analogs of write, read and reset operations for the cells.

Specifically, we leverage the traffic records collected by DCU in dozens of previous DTIs and the current DTI for prediction. In the design of prediction module, several LSTM modules are connected in series, and the states (*i.e.*, the value of weight coefficients and the MSE results) are transferred between adjacent LSTM modules. Note that the input of the raw data is attached with three tags which are respectively the time steps (*i.e.*, the PW  $T_\Delta$ ), traffic volume per time step and instances (*i.e.*, the slices). Specifically, for slice  $m$ , within a PW  $T_\Delta$ , the raw data  $\mathbb{D}_r = \{d_{t-n}, d_{t-n+1}, \dots, d_t\}$  is input to the first LSTM block, and the predicted results are obtained by the manipulations for minimizing objective (1), and then used for resource allocation for individual slices. We use *Tensorflow* framework for performing LSTM.

**Remark 1.** It has been proved by Godfrey et al. [24] that LSTM outperforms ARIMA and SVR on time-series forecasting through a large number of experiments.

As mentioned in Section IV.A, we use the confidence level  $\chi(\chi \in (0, 1))$  for intuitively describing the respective significance of the prediction results. In other words, we can relax strict performance isolation according to the accuracy of prediction (based on the amount of data) and the ser-

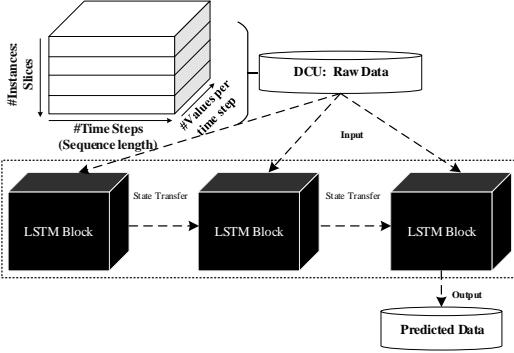


Fig. 4: Prediction by LSTM under the iRSS decision Framework

vice types. For instance, best-effort traffic requests without stringent requirements can tolerate an imprecise prediction results. Let the predicted traffic volume demand be  $\mathbb{D}_p = \{d_{t+1}, d_{t+2}, \dots, d_{t+n}\}$ . The sample mean of the elements of  $\mathbb{D}_p$  is  $\bar{d} = \frac{1}{n} \sum_{k=1}^n d_{t+k}$ , and the sample standard deviation is  $\sigma(d) = (\frac{1}{n} \sum_{k=1}^n (d_{t+k} - \bar{d})^2)^{\frac{1}{2}}$ . We denote by  $\chi$  the confidence level which can be dynamically adjusted according to the service types, and the confidence interval is given by

$$\Pr\{r_m(t) \in \bar{d} \pm z_{(1-\chi)/2} \cdot \frac{\sigma(d)}{\sqrt{n}}\} = \chi, t \in [0, T_\Delta]. \quad (3)$$

In other words, the probability that the future traffic requests (the predicted volume) lie within  $[\bar{d} \pm z_{(1-\chi)/2} \cdot \frac{\sigma(d)}{\sqrt{n}}]$  is  $\chi$ . Due to the penalties imposed by traffic service level agreements (SLAs), we suppose to assign resources based on the upper bound of the prediction interval as it provides the *worst-case* of a forecasted traffic level. Let  $r_m^U$  be the upper bound of the confidence interval for slice  $m \in \mathcal{M}$ . Therefore, for the next PW  $T_\Delta$ , the predicted amount of spectrum resources allocated to tenants are  $\sum_{m \in \mathcal{M}} r_m^U(T_\Delta) = \Theta^r$ .

### B. Distributed Architecture for On-line Resource Scheduling of RAN Slicing

In conjunction with large timescale resource allocation based on periodic traffic prediction, we employ an on-line (or real-time) algorithm for small timescale resource scheduling in iRSS framework. After carefully investigating the characteristics of on-line resource scheduling problem for RAN slicing and the system model, we propose a distributed on-line resource scheduling architecture as shown in Fig. 5. This architecture consists of five components: 1) a distributed store of experience replay memory [9]; 2) parallel actors (*i.e.*, slices) that generate new actions; 3) parallel learners (*i.e.*, slices) that are trained by stored experience; 4) a distributed neural network (NN) to approximate the Q-value and policy; and 5) a distributed account book to store and exchange information among slices. It is obvious that individual slices have different requirements on resources scheduling, and thus the resource allocation strategy for different slices cannot be optimized in the same way. In other words, joint optimization of resource allocation for all slices is infeasible. We thus resort to the parallel computing for performing resource allocation

for individual slices, where the slices learn from the stored experience and generate new actions in a parallel way. In addition, the algorithm of A3C is employed in this distributed architecture as an asynchronous updating on-line solution to the RSS problem, which is a sub-filed of reinforcement learning under a c-MDP model.

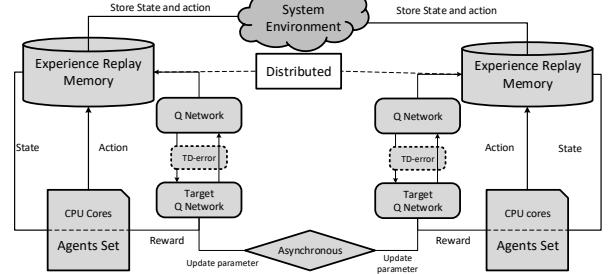


Fig. 5: Architecture of distributed on-line resource scheduling of RAN slicing

## VI. CONTINUOUS MARKOV DECISION PROCESS FOR ON-LINE RESOURCE SCHEDULING

We model the decision process of on-line scheduling as a continuous-time Markov decision process (c-MDP), and adopt RL to resolve this problem. In conventional reinforcement learning algorithms *i.e.*, Q-learning algorithm, value iteration, Monte Carlo Tree Search, etc., the decision agents update their Q-value by using *bootstrap* method [25], which cannot well estimate the Q-value of the subsequent state that is not traversed. Therefore, these algorithms cannot be used when the state or action space is infinite. Therefore, we exploit the asynchronous advantage actor-critic (A3C) (Mnih *et al.*, [10]) for providing an on-line solution to the c-MDP based scheduling problem. A3C accelerates the convergence when compared with traditional AC, by employing an asynchronous update strategies in a distributed architecture. Moreover, A3C has achieved the state-of-the-art results on many gaming tasks including Atari 2600 [26].

### A. c-MDP Model

The c-MDP is modeled as a five-tuple  $\langle \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{P}, \gamma \rangle$ , where  $\mathcal{S}$  denotes the state space,  $\mathcal{A}$  denotes the action space, and let  $\mathcal{F} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  be a given cost function used to measure the quality of the decision. We consider a set of randomized stationary policies  $\Pr = \{\pi_\theta; \theta \in \mathbb{R}^n\}$ , parameterized in terms of a vector  $\theta = \{\theta_t, \theta_{t+1}, \dots\}$ . For example, for each pair  $(s, a) \in \mathcal{S} \times \mathcal{A}$ ,  $\pi_\theta(s, a) = \Pr\{s'|s, \theta\}$  denotes the probability of taking action  $a$  at state  $s$ , under the policy based on  $\theta$ . Moreover, we assume that  $\pi$  is differentiable with respect to its parameter, *i.e.*,  $\frac{\partial \pi_\theta(s, a)}{\partial \theta}$  exists.  $\gamma \in [0, 1]$  is the discount factor. Note that in this MDP model, the transition probability from one state to another is determined once an action is adopted, which can be expressed as

$$\Pr\{s'|s, a\} = \begin{cases} 1 & \text{if } s' = s_{t+1} \\ 0 & \text{otherwise} \end{cases}$$

The *State* in the model describes the network environment. We define that at arbitrary time  $t$ , the sharing (or cognitive) state space is represented by a set of  $\{\mathcal{U}_{\mathcal{M}^I}, \mathcal{U}_{\mathcal{M}^{II}}, h_{\mathcal{M}^{II}}, r_{\mathcal{M}}\}$ . In this work, the network will decide the proportion of resources allocated to slices. Note that in this decentralized network environment, actions between slices are unobservable, such that each slice makes decisions independently. Therefore, for a given slice  $m$ , the *Action* is defined as  $A_m(s(t)) = a_m(t)$  under the deterministic state  $s \in \mathcal{S}$ , where  $a_m(t)$  is the fraction of resources allocated to slice  $m$  at time  $t$ .

The *Reward* reflects our design objectives. As shown in Problem 2, the objective is minimizing the resource consumption (or improving the resource efficiency) while guaranteeing a certain degree of performance isolation. The reward function should be designed to guide the slices to make decisions towards the direction of the optimization objective of Problem 2. The logarithm function is widely used as the elastic utility function for allocating resources (or determining transmission rate) in related studies [27] [28]. In our model, we make some modifications to the logarithm function based on our design objective. In more detail, we define the elasticity demand on assigned resources related to  $r_m(t)$  and  $h_m(t)$  as an absolute value of the logarithm function, i.e.,  $e(r_m(t), h_m(t)) = |\ln(r_m(t) - h_m(t) + 1)|$ . The definition domain of  $e(\cdot)$  is within  $[h_m(t) - 1, h_m(t)] \cup [h_m(t), +\infty)$ . Therefore, when  $r_m(t) < h_m(t)$ , we transfer  $r_m(t)$  into  $\frac{r_m(t)}{h_m(t)} + h_m(t) - 1$  whose value range is within  $[h_m(t) - 1, h_m(t)]$  that satisfies the definition domain of  $e(\cdot)$ .

In addition, by considering the constraints in Problem 2, we formulate the reward function for slices as follows

$$c_m(t) = e(r_m(t), h_m(t)) + \mathbb{B}(\mathcal{D}_m(t)), \quad (4)$$

where  $\mathbb{B}(\cdot)$  is defined as bonus incurred by the reconfiguration conducted for slice  $m$ . Specifically,  $\mathbb{B}(\mathcal{D}_m(t)) = w \cdot \mathbf{1}_{\{\mathcal{D}_m(t) \neq 0\}}$ , where  $w > 0$ , and  $\mathbf{1}_{\{\mathcal{V}\}}$  is the indicator function that equals to 1 if condition  $\mathcal{V}$  is satisfied and 0 otherwise. The value of  $w$  in the bonus function makes the length of  $\Omega_m(\mathcal{D}_m(t)) = 0$  (in constraint (2.3)) controllable.

As described in Problem 2, meeting the slices' requirements with minimal resources is the primary objective. This is equivalent to minimizing the value of  $c_m(t)$ . However, due to the existence of bonus,  $h_m(t)$  may be not the optimal solution. The cost function has the following characteristics: (1) when  $r_m(t) \geq h_m(t)$ , with the increase of  $r_m(t)$ , the cost will gradually increase towards saturation since the partial derivative  $\nabla_{r_m} e(r_m(t), h_m(t))$  gradually decreases to zero; (2) when  $r_m(t) \leq h_m(t)$ , with the decrease of  $r_m(t)$ , the cost rapidly increases towards infinity, indicating the degree of resource shortage in the slice; (3) the gradient of the cost function on the left side of  $h_m(t)$  is steeper than that on the right side. As we pursue the long-term system performance optimization, resource scheduling strategies that cannot satisfy the QoS requirement (i.e.,  $r_m(t) < h_m(t)$ ) at this decision time cannot be just simply dropped. We preserve these strategies in case that it can provide good performance for subsequent states, and thus can improve the long-term system performance.

The objective of our MDP modeling is to find a policy  $\pi_\theta(\ell)$  which can optimize reward  $r(t)$  from time  $t$  to  $t + T_\Delta$ .

Assuming that the MDP starts from an initial state  $s_0 \in \mathcal{S}$ , and makes decisions according to  $\pi_\theta$ , resulting in a trajectory  $\ell \sim \{s_0, a_0, s_1, a_1, \dots, s_{T_\Delta}, a_{T_\Delta}\}$ . Then, we formulate the following objective function  $O(\pi_\theta)$  by considering the discounted return:

$$\text{Min } O(\pi_\theta) = E_{\ell \sim \pi_\theta(\ell)} [c_m(\ell)], \quad (5)$$

where  $r(\ell) = \sum_{k=0}^{T_\Delta} \gamma^k r_{t+k}$  is the discounted cumulative reward starting from time  $t$  and increasingly discounted at subsequent steps by factor  $\gamma \in (0, 1]$ .

Note that our MDP problem has continuous state and action space as the wireless channel state and the amount of assigned resources are continuous variables, and thus it is infeasible to compute and save all value functions for every particular state-action pair. With respect to continuous or infinite state and action problems, the objective of (2) can be rewritten as

$$O(\pi_\theta) = E_{\ell \sim \pi_\theta(\ell)} [c_m(\ell)] = \int_{\ell \sim \pi_\theta(\ell)} \pi_\theta(\ell) c_m(\ell) d\ell. \quad (6)$$

### B. Actor and Critic in the c-MDP for the Slice RSS

The *Actor* works with a family of parameterized policies. The gradient of the performance (5), with respect to the actor parameters, is directly estimated by simulation, and the parameters are updated in the direction of improving the objective (5). Specifically, the gradient of the feedback function with respect to the vector of  $\theta$  is expressed by  $\nabla_\theta O(\pi_\theta)$ . The reinforcement method updates  $\theta$  by using the gradient

$$\nabla_\theta O(\pi_\theta) = \nabla_\theta \log \pi_\theta(\ell) c(\ell). \quad (7)$$

As we use *1-step* update in this work, such that (7) is an unbiased estimation of  $E_{\ell \sim \pi_\theta(\ell)} [c(\ell)]$  [29].

We use the known Gaussian probability distribution to derive a stochastic policy for selecting actions, which can be presented as

$$\pi_\theta(a|s) \sim \mathcal{N}(\mu(s), \sigma^2), \quad (8)$$

where  $\mu(s)$  is the mean and  $\sigma$  is the standard deviation.  $\mu(s)$  is indeed the action that has the largest probability to be chosen at state  $s$ , and  $\sigma$  indicates the extent of exploration over all actions at state  $s$ . By exploiting Gaussian probability distribution, *exploration* (searching for better strategies) and *exploitation* (exploiting the previous best strategies) can be well balanced in the action selection process.

We use the network state  $\mathcal{S}$  as the feature vector  $\Phi(s_t) = (\phi_1, \phi_2, \dots, \phi_f)^\top$  with  $f$  elements which need to be normalized for better performance of linear approximation. Moreover, a linear feature-based function is used to approximate  $\mu(s_t)$  which is represented as

$$\mu_\theta(s_t) = \theta^\top \cdot \Phi(s_t), \quad (9)$$

where  $\theta^\top = (\theta_1, \theta_2, \dots, \theta_f)$  is updated by the policy gradient method. Specifically, the policy gradient update formulation for parameters  $\theta$  is given by  $\Delta\theta = \alpha_{a,t} \cdot \nabla_\theta O_m(\pi_\theta)$ , where  $\alpha_{a,t} > 0$  is an appropriately small step-size for the policy update.

The *Critic* relies exclusively on the value function approximation and aims at learning an approximate solution to

the Bellman equation, which will then hopefully prescribe a near-optimal policy [30]. We use function approximation to estimate the value function and update the parameters by using some samples. The approximation state-action value and state value, denoted by  $V_\vartheta(s_t) \approx V(s_t)$ , can be parameterized by the vector  $\vartheta^T = (\vartheta_1, \vartheta_2, \dots, \vartheta_f)$ . We choose the linear feature-based function to approximate  $V_\vartheta(s)$  since this solution is effective for our scenario with a low complexity and fast convergence. Thus,

$$V_\vartheta(s_t) = \vartheta^T \cdot \Phi(s_t). \quad (10)$$

**Remark 2.** *Linear approximation function can be deemed as a neural network with 1 neural cell and the activation function is a linear equation. Since linear feature-based function performs well in function approximation and has demonstrated high efficiency in gradient decent, it has been successfully used in some on-line game applications [26] [30], such as cart-pole, acrobot and Atari, etc.*

Next, we need to update the parameter vector  $\vartheta$  in the critic process. We introduce  $\varpi_t$  as the temporal difference (TD) error between the approximated value and the real value at a state when taking an action, i.e.  $\varpi_t = V^\pi(s_t) - V_\vartheta(s_t)$ , where  $V^\pi(s_t) = c_{t+1} + \gamma V_\vartheta(s_{t+1})$ . Note that the objective of this *critic* process is to minimize the TD-error  $\varpi_t$ . Therefore, the objective function of the *critic* is designed as  $\varepsilon_\vartheta^\pi = \arg \min_\vartheta \frac{1}{2}(\varpi_t)^2$ , and the gradient of this quadratic error with respect to  $\varpi_t$  can be derived as  $\nabla_\vartheta \varepsilon_\vartheta^\pi = |\varpi_t| \cdot \nabla_\vartheta V_\vartheta(s_t)$ .

We use the gradient descent method to update the approximation towards the gradient, and thus the parameter vector  $\vartheta$  can be updated by  $\Delta\vartheta = \alpha_{c,t} \cdot |\bar{w}_t| \cdot |\nabla_\vartheta V_\vartheta(s_t)|$ , where  $\alpha_{c,t}$  is the learning rate for the value function. As  $V_\vartheta(s_t)$  is approximated by the linear function, we have

$$\nabla_\vartheta V_\vartheta(s_t) = \Phi(s_t), \quad (11)$$

There are two methods to compute  $\text{TD}(\lambda)$ : forward and backward estimation methods respectively. Specifically, the forward method combines the future steps for joint optimization. However, as shown in Fig. 6, the states in the future two or more steps may be observed in tens of TTIs in this scenario, which is infeasible since we focus on online-solution. On the other hand, eligibility trace can be used as a backward method to evaluate  $\text{TD}(\lambda)$ , which provides a better way of assigning credit to state-action pairs which have been visited several steps earlier [30]. It is known that  $\text{TD}(0)$  method considers one-step backup, while the reward is the result of a series of steps. The extensive use of the eligibility traces turns  $\text{TD}(0)$  into the backward method of  $\text{TD}(\lambda)$ , which can speed up the learning process considerably. The eligibility trace vector for all features at time  $t$  is denoted by  $z_t$  and its update equation is  $z_t = \varsigma \cdot \gamma \cdot z_{t-1} + \Phi(s_t)$ , where  $\varsigma \in [0, 1]$  is the trace-decay factor.  $\varsigma \cdot \gamma$  makes the recently used features more eligible for receiving credit. With the use of eligibility traces, the update interval becomes  $\nabla\vartheta = \alpha_{c,t} \cdot |\varpi_t| \cdot z_t$ .

#### C. Parallel Computing and Information Exchange of Network Slices in an AC unit

The slices execute the A3C algorithm in multiple cores, where the slices learn from the stored experience and generate

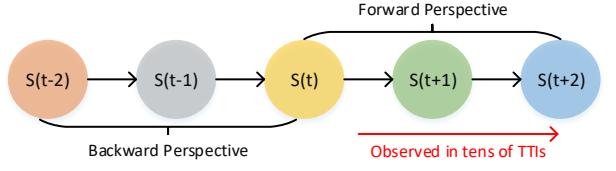


Fig. 6: Backward and forward perspective of  $\text{TD}(\lambda)$

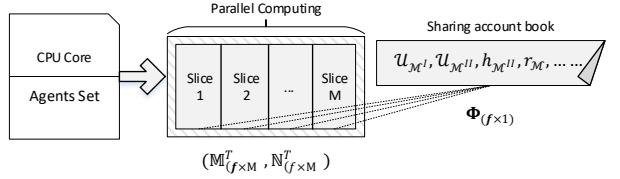


Fig. 7: Illustration of parallel computing and distributed account book for network slices

new actions in a parallel way, as shown in Fig. 7. However, with the increase of the number of slices, this parallel computing will consume a lot of computing resources. Therefore, we implement the parallel computing for slices by vectorizing the corresponding parameters in A3C framework, for effectively improving the computational efficiency. In addition, in order not to violate the different TTI requirements of individual slices, the DTI is set as the minimum TTI of the slices, i.e.,  $t_\Delta = \arg \min_{m \in \mathcal{M}} \tau_m(t)$ . Specifically, let there be  $M$  slices, and the dimension of the parameter vectors  $\theta$  and  $\vartheta$  is  $f \times 1$ . We form the parameter vectors of slices, such that the dimension of the vectorized parameter vectors represented by  $\mathbb{M}$  and  $\mathbb{N}$  is now  $f \times M$ . Therefore, (9) and (10) can be respectively rewritten as  $\mathbb{U}_{1 \times M} = (\mathbb{M}_{f \times M}^T \cdot \Phi_{f \times 1})^T$  and  $\mathbb{V}_{1 \times M} = (\mathbb{N}_{f \times M}^T \cdot \Phi_{f \times 1})^T$ .

This parallel computing for multiple slices can be deemed as a multi-agent system, where information exchange is the key to designing a feasible policy. Indeed, necessary information exchange among slices (agents) should be performed before making decisions in such a multi-agent system. As shown in (9) and (10), the value  $V_\vartheta(s_t)$  and policy  $\mu(s_t)$  are linear with the feature vector  $\Phi(s_t)$  which is deemed as an environment information carrier shared among slices. In addition, based on the decentralization model as shown in Fig. 1, we let  $\Phi(s_t)$  be the distributed account book used to record the information among slices, and each slice has the authority to modify and maintain this account book. In the resource scheduling for slices, a key problem is to guarantee that the total amount of required resources does not exceed the overall amount of system resources, represented by Constraint (2.2). However, when the resources become scarce among slices, the information in the account book cannot adequately solve this problem. In this regard, we adopt an additional indicator vector in  $\Phi(s_t)$  to indicate if there are sufficient resources to be shared among slices, and guide the strategies of slices towards decreasing the required resources to make feasible

resource allocation decisions.

**Definition 2.** An Dirac impulse function in terms of the remaining shared resources  $\Theta^r$  is used as the indicator vector

$$\delta(\Theta^r) = \begin{cases} \infty, & \text{if } \Theta^r = 0 \\ 0, & \text{otherwise} \end{cases}$$

Thus, the feature vector  $\Phi(s_t)$  can be rewritten as  $(\phi_1, \phi_2, \dots, \phi_f, -\delta(\Theta^r))^T$ .

**Theorem 1.** If  $\vartheta_{f+1}$  and  $\theta_{f+1}$  are initialized as a negative constant  $\iota < 0$ ,  $-\delta(\Theta^r)$  in the feature vector can well guide the strategies towards the direction that the resources will not overflow.

*Proof:* When there are sufficient resources for sharing among slices,  $-\delta(\Theta^r) = 0$ , and  $-\delta(\Theta^r) \cdot \iota = 0$ . In this case, this element in the feature vector will not affect the value of  $V_\vartheta(s_t)$  and the policy  $\mu(s_t)$ , and thus not make any sense on the decision-making of A3C algorithm. However, if there are no remaining shared resources in RAN, the element  $\Theta^r = 0$ , and  $-\delta(\Theta^r) = -\infty$ , which makes the value of  $V_\vartheta(s_t)$  equal to  $-\infty$ , and thus the TD-error  $|\varpi_t| = |V^\pi(s_t) - V_\vartheta(s_t)| = |V^\pi(s_t) + \infty| = +\infty$ . Therefore, the strategies of excessive use of the spectrum resources bring large TD-error. In this situation, the next stage of strategies will be in the opposite direction of trying to decrease the TD-error. In other words, the new strategy adopted next time is in the direction of avoiding consuming the entire spectrum resources and satisfying the objective of (6) in the meantime. ■

#### D. Asynchronous Actor-Critic (A3C) Algorithm for iRSS

A3C algorithm relies on parallel actor-learners and accumulative updates for improving training stability. Specifically, the Actor-Critic algorithm is the combination of Actor-only and Critic-only methods. However, as the variance of convergence in AC algorithm could be very large, we introduce the advantage function  $A(s_t)$  as the bias to greatly decrease the variance [31]. For further improving performance, we let  $A(s_t)$  be the TD-error  $\varpi_t$ . Then, (7) can be rewritten as

$$\nabla_\theta O(\pi_\theta) = \nabla_\theta \log \pi_\theta(a|s) A(s_t). \quad (12)$$

Obviously, as we intend to minimize the objective (2), we update the value of  $\theta$  by  $\theta \leftarrow \theta - \nabla_\theta O(\pi_\theta)$ .

As we conduct the asynchronous decision process by multi-threads, we use  $\theta^+$  and  $\vartheta^+$  as the global parameters for the target neural network, which are updated asynchronously by the distributed actor-critic units in different threads. During the update process, we set two timers  $I_{target}$  and  $I_{asynUpdate}$  in the parameter updating process. In more detail, for every time interval  $I_{target}$ ,  $\theta$  and  $\vartheta$  are updated by  $\nabla\theta$  and  $\nabla\vartheta$  with given step-size in an AC unit. At the end of this round of learning or for every time interval  $I_{asynUpdate}$ , the central controller performs asynchronous updates of  $\theta^+$  and  $\vartheta^+$  by using the accumulated gradient  $\nabla\theta$  and  $\nabla\vartheta$  respectively. Next,  $\theta^+$  and  $\vartheta^+$  are assigned to  $\theta$  and  $\vartheta$  for the next round of learning.

As linear approximation is used for the value  $V_\vartheta(s_t)$  and policy  $\mu_\theta(s_t)$ , A3C becomes an online solution with low

computational complexity. Note that the convergence condition of A3C is satisfied when the learning rate  $\alpha_{a,t}$  and  $\alpha_{c,t}$  respectively satisfy  $\sum_{t=0}^{\infty} \alpha_{a,t} = \infty$  and  $\sum_{t=0}^{\infty} \alpha_{c,t} = \infty$ , and meanwhile  $\sum_{t=0}^{\infty} \alpha_{a,t}^2 = \infty$  and  $\sum_{t=0}^{\infty} \alpha_{c,t}^2 = \infty$  [32]. Moreover, we empirically set the learning rate according to physical system consideration to ensure that the global optimal solution can be found by iterations. Specifically, if the learning rate is set very small, the optimal solution will be eventually found, but the convergence rate could slow. On the other hand, if the learning rate is set very big, the convergence rate could be fast, but the solution may be oscillated. In summary, we elaborate the algorithm of iRSS in Algorithm 1.

---

#### Algorithm 1 Intelligent Resource Scheduling Strategy (iRSS) for 5G RAN Slicing

---

```

1: //Given global shared parameters  $\chi$ ,  $\Phi(s)$ ,  $\theta^+$ ,  $\vartheta^+$ , counter  $T_c, t_c$ , and terminal time  $T_{max}$ 
2: Initialize thread step counter  $T_c, t_c \leftarrow 0$ 
3: Initialize global target network parameters  $\theta^+, \vartheta^+ \leftarrow 0$ 
4: Get initial state  $s$  and the value of  $r_m^U$ 
5: repeat
6:   Initialize the critic and policy gradients:  $\nabla\theta, \nabla\vartheta \leftarrow 0$ 
7:   Generate a policy trajectory by using  $\pi_\theta(\ell)$ , where  $\ell = \{s_0, a_0, c_0, s_1, a_1, c_1, \dots\}$ 
8:    $T_c \leftarrow T_c + 1$  and  $t_c \leftarrow t_c + 1$ 
9:   if  $T_c \bmod I_{target} == 0$  then
10:    Synchronize the local target network parameters:
      $\theta \leftarrow \theta^+, \vartheta \leftarrow \vartheta^+$ 
11:   end if
12:   if  $t_c \bmod I_{asynUpdate} == 0$  or  $s$  is the terminal state
     then
13:    Perform an asynchronous update of  $\theta^+$  and  $\vartheta^+$  by
     using the accumulated gradients of  $\nabla\theta$  and  $\nabla\vartheta$ .
14:   end if
15: until  $T_c > T_{max}$ 

```

---

#### E. Computational Complexity

The computational complexity of A3C is given by

$$\begin{aligned} \mathcal{O}((N_u \cdot \frac{1}{N_u}) \cdot T_c \cdot M \cdot (\sum_{i=0}^{L_a} u_a^{(i)} \cdot u_a^{(i+1)} + \sum_{j=0}^{L_c} u_c^{(j)} \cdot u_c^{(j+1)}) = \\ \mathcal{O}(T_c \cdot M \cdot (\sum_{i=0}^{L_a} u_a^{(i)} \cdot u_a^{(i+1)} + \sum_{j=0}^{L_c} u_c^{(j)} \cdot u_c^{(j+1)})), \end{aligned} \quad (13)$$

where  $N_u$  is the number of CPU threads used to train the AC algorithm,  $T_c$  is the training steps or termination steps,  $u^{(i)}$  ( $u^{(j)}$ ) is the number of units in the  $i^{th}$  ( $j^{th}$ ) layer of the neural network, and  $u_c$  ( $u_a$ ) denotes the number of units of the critic network (actor network).

We can see that the computational complexity is linear with the length of  $T_c$ , the depth of the hidden layer and the number of slices  $M$ . Moreover, we have simulated in Fig. 10 that the training steps used in the policy gradient is less than 100 steps in most cases, such that the convergence rate is adequate for meeting the timeliness requirement of on-line resource scheduling.

## VII. NUMERICAL RESULTS AND DISCUSSIONS

We evaluate the performance of our proposed iRSS by extensive simulations. At the beginning of the learning process of iRSS, DCU has not collected sufficient information of traffic data, and thus we use the widely used Gaussian distribution [33] [34] to simulate the traffic data. Note that we exploit the collaboration of DL and RL for improving the accuracy of decisions, thus if the simulated data cannot well fit the real data, the RL can be relied on for modifying the decisions. The number of slices  $M$  is set as 10 in the simulation experiments. As mentioned in the parallel computing method, increasing the number of slices will just increase the dimension of the parameter vectors  $\mathbb{M}_{f \times M}^T$  and  $\mathbb{N}_{f \times M}^T$ , which will not significantly increase the complexity of the algorithm. The step-size  $\alpha_{a,t}$  in the *actor* process and the step-size  $\alpha_{c,t}$  in the *critic* process are set very small as constants to ensure the convergence of the algorithm. Specifically, we set  $\alpha_{a,t} = 10^{-5}$ , and  $\alpha_{c,t} = 10^{-3}$  respectively. Other parameters used are listed in TABLE II.

TABLE II: Used simulation parameters

Parameter Description	Value
Number of Slices $M$	10
Step-size $\alpha_{a,t}$	$10^{-4}$
Step-size $\alpha_{c,t}$	$10^{-3}$
Discount factor $\gamma$	0.99
Trace-decay factor $\varsigma$	0.2
The Number of Radio Blocks	200
Prediction window	3600s

We first examine the performance of the proposed iRSS in the large timescale. In this simulation, we examine the prediction accuracy of the LSTM in our iRSS without RL. We conduct this simulation in the platform of Python by using tensorflow. Fig. 8 (a) shows the mean square error (MSE) as a function of the number of epochs. Fig. 8 (b) shows the error between the targets and outputs of training data, validation data and test data, respectively. In this simulation, we use 70%, 15% and 15% of the dataset for training, validation, and testing, respectively. From the simulation results, we can see that the MSE can eventually converge to a minimum value with around 18 epochs. Note that the dozens of epochs are much smaller than the length of PW. Therefore, the LSTM can converge at the end of a PW and provide an optimum prediction results. Moreover, at this MSE point, for most instances, the errors between the targets and outputs are concentrated on both sides of 0, showing that the LSTM algorithm in this work can be leveraged to well predict the traffic volume.

Next, we investigate the relationship between the performance isolation of the slices and the confidence level through simulation with the same settings of the first experiment. Fig. 9 shows that the PID monotonically increases with the confidence level. This indicates that the amount of pre-assigned resources for the slices is appropriately increased with the confidence level, and hence the isolation degree of slice is also improved. On the other hand, if we decrease the confidence level, the isolation degree is decreased and the service requirements are violated with the probability of  $1 - \chi$ . In this case, more resources will be released to the resource pool as the shared resources.

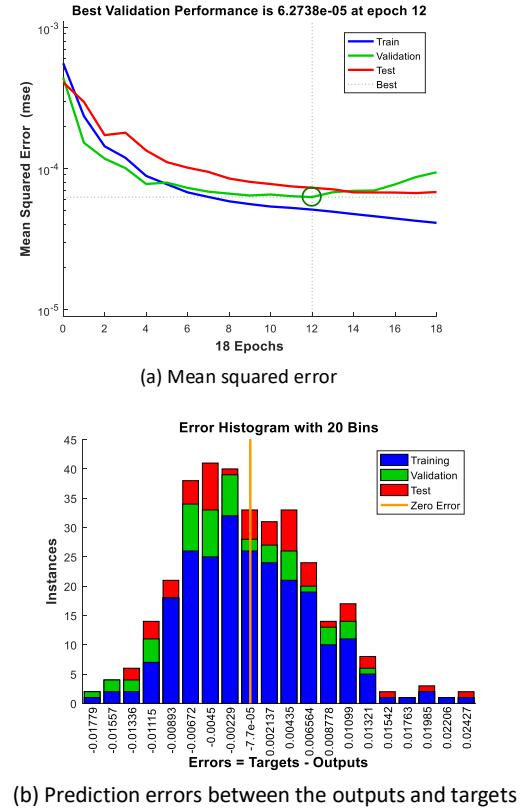


Fig. 8: Performance of LSTM in traffic prediction

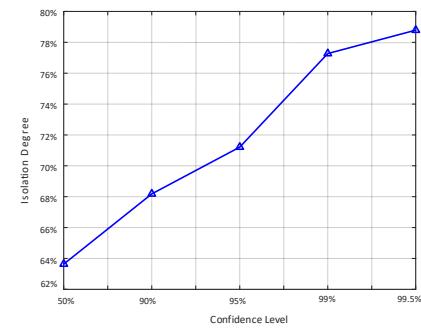


Fig. 9: The relationship between isolation degree and the confidence level

We then examine the convergence efficiency of iRSS in small-timescale without using LSTM for prediction. Note that the A3C algorithm is implemented in the iRSS for performing the small time-scale resource scheduling. In order to meet online requirement, we make an assumption that if  $|V_t - V_{t+1}| \leq \epsilon (\epsilon = 1e-3)$  within some policy gradient steps (*i.e.*, 100 steps as shown in Fig. 11), the A3C algorithm is assumed converged. Objective (5) is minimizing the cumulated reward, thus the smaller the value of the reward at each decision time, the better the strategy. Fig. 10 shows the reward vs. the number of learning steps in the A3C algorithm. We can see that the reward of the algorithm converges to the optimal decision rapidly with dozens of learning steps. Nevertheless, at the beginning of the simulation, the reward becomes very high

due to an inexperienced or a failed exploration. In addition, with the progress of the learning process, there exists certain degree of jitter due to the length of step-size in the gradient decent process. Therefore, when compared with the results of large timescale prediction, on-line learning may have unstable performance in the beginning. Fortunately, the computing time of dozens of learning steps is totally acceptable for the timeliness requirement of small timescale resource scheduling. In other words, iRSS can provide a feasible on-line solution of resource scheduling for slices within dozens of learning steps.

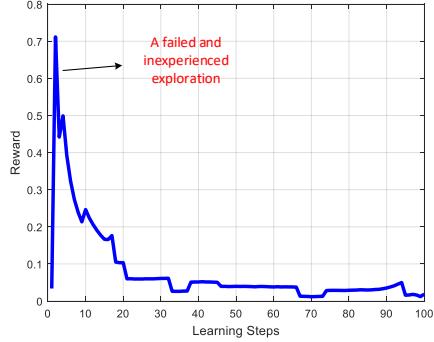


Fig. 10: Convergence of the A3C algorithm

Next, we extend the simulation to different network settings. As shown in Fig. 11, we compare the average network system rewards under different number of network slices and resource blocks respectively. In this simulation, the number of network slices and RBs varies from 1 to 20 and 1 to 200 respectively, to examine the system average cost. From this figure, we can see that the system average cost increases monotonically with the number of slices when the amount of RBs is fixed. This is because that with the increase of the number of slices, more resources are required to meet their requirements. Therefore, the system resources will be consumed off quickly, resulting in system cost increase. On the other hand, when the number of slices is fixed, the system average cost decreases to 0 when the amount of RBs increases. This implies that providing adequate resources for users will significantly improve network performance.

In the last experiment, we compare the performance our proposed iRSS with that of the state-of-the-art machine learning schemes including the traditional (or tabular) Q-learning and the classic AC algorithm, as well as a heuristic resource scheduling algorithm (HRSA). The performance metrics we use include the cumulated reward and resource utilization (RU). We define the RU of slice  $m$  of a period as  $RU = \sum_t h_m(t)/\sum_t r_m(t)$ . Note that Q-learning has poor performance in solving decision problems with continuous action space. For fair comparison, we discretize the action space first, and then choose actions by sampling for implementing Q-learning. Furthermore, due to the high computational complexity of Q-learning, we limit the learning time of Q-learning for a fairness comparison with the other two schemes. In this regard, let Q-learning sample 200 decision points by using Gaussian distribution, *i.e.*,  $\pi_t^Q(s) \sim N(\mu_t^Q, \sigma^2)$ , where the value of the mean point  $\mu_t^Q$  equals to the amount of

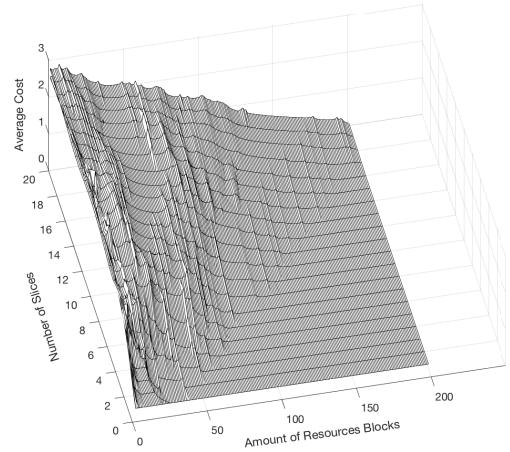


Fig. 11: System average costs with different number of slices and RBs

required resources in the last time, *i.e.*,  $\mu_t^Q = h_m(t - 1)$ . Other parameters in the three algorithms are listed in TABLE I and TABLE II. Note that in (5), minimizing the cumulated reward or Q-value is our optimization objective in this work. We design a heuristic resource scheduling algorithm (HRSA) for performance comparison. In HRSA, each slice pre-reserves a certain amount of resources for the performance isolation. Within every DTI, HRSA configures resources to each slice from the pre-reserved resources. In addition, we make an assumption that the amount of resources pre-reserved for each slice is equivalent to the requirements of slices of the last time.

Fig. 12 shows the cumulated cost of the three machine learning based algorithms over DTIs. We can see that the cumulated cost of iRSS is always significantly lower than that of the AC and Q-learning algorithms. In Fig. 13, we compare the resource utilization of the four algorithms when the required isolation degree is fixed. We compare the resource utilization (RU) of the four algorithms when the number of DTIs increases from 0 to 200, while the other configurations remain the same as in the previous experiment. We can see that the RU of iRSS is also always apparently higher than that of the Q-learning, classic AC and HRSA algorithms, respectively. We can also observe that when the number of DTIs is smaller than 40, the RU of Q-learning, AC, and HRSA fluctuates, while that of iRSS fluctuates mildly. This is because that the number of samples used for training is too small, resulting in poor generalization performance of the Q-learning and AC algorithms. Intuitively, HRSA lacks the ability of prediction and pre-reserves resources for slices empirically, such that it performs poorly on RU. Due to the asynchronous updating schemes used, A3C can adapt to the variation of environment much faster than the other two learning schemes. When the number of DTIs exceeds 40, the RU of the four algorithms gradually becomes stable, and the RU of iRSS is higher than that of Q-learning, AC algorithms and HRSA by approximately 16.3% ~ 19.8%, 8.3% ~ 13.7%, 30.5% ~ 34.7%, respectively. Since the policy iteration in the “action” process of A3C has much smaller search granularity and much lower

computational complexity when compared with the discrete action space based Q-learning, iRSS is able to find better strategies and achieve better performance in most cases. In addition, as aforementioned, the A3C algorithm is executed in a distributed architecture, and adopts asynchronous updating schemes, endowing iRSS algorithm the higher convergence efficiency and ability of generalization.

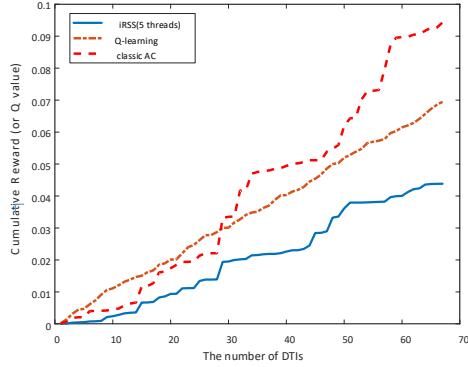


Fig. 12: Cumulative cost of the machine learning schemes

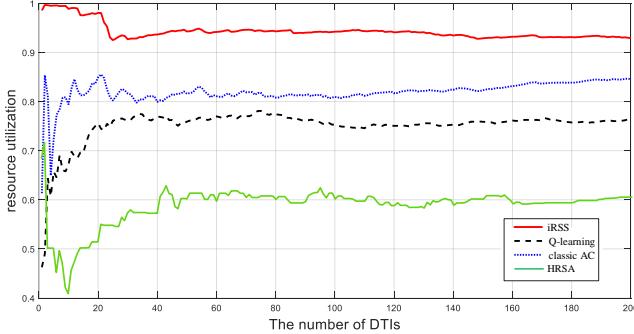


Fig. 13: Comparison of resource utilization

### VIII. CONCLUSION

In this paper, we have proposed an intelligent resource scheduling strategy (iRSS) for 5G RAN slicing. Our proposed iRSS for RAN slicing is embedded in a collaborative learning framework. Both DL and RL are incorporated in the framework and work in a collaborative way, so as to tackle both large and small time-scales network dynamics. The primary objective of iRSS is guaranteeing a certain degree of performance isolation among slices, and meanwhile improving the resources multiplexing gains. We have used the LSTM as the DL algorithm in large timescale to predict traffic volume for resource allocation and the parallel computing based A3C algorithm in the small timescale for performing the resource scheduling. Significant performance improvements in terms of the cumulated reward and resource utilization are achieved by iRSS when compared with other benchmark algorithms.

### REFERENCES

- [1] H. Zhang, N. Liu, X. Chu, K. Long, A. H. Aghvami, and V. C. Leung, "Network Slicing Based 5G and Future Mobile Networks: Mobility, Resource Management, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 8, 2017.
- [2] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, and H. Zhang, "Intelligent 5G: When Cellular Networks Meet Artificial Intelligence," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 175–183, 2017.
- [3] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: RAN Slicing for a Flexible and Cost-Effective Multi-Service Mobile Network Architecture," *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pp. 127–140, 2017.
- [4] P. Caballero, A. Banchs, G. De Veciana, and X. Costa-Perez, "Network slicing games: Enabling customization in multi-tenant networks," *Proceedings - IEEE INFOCOM*, 2017.
- [5] A. R. Elsherif, W.-P. Chen, A. Ito, and Z. Ding, "Resource Allocation and Inter-cell Interference Management for Dual-Access Small Cells," *IEEE Journal on Selected Areas in Communications*, vol. 8716, no. c, p. 1, 2015.
- [6] M. Yan, G. Feng, J. Zhou, and S. Qin, "Smart Multi-RAT Access based on Multi-Agent Reinforcement Learning," *IEEE Transactions on Vehicular Technology*, vol. PP, no. 99, p. 1, 2018.
- [7] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, "Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 56–62, 2019.
- [8] F. a. Gers and F. Cummins, "1 Introduction 2 Standard LSTM," *Neural Computation*, vol. 12, no. 10, October 2000, pp. 2451 – 2471, 1999.
- [9] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneerselvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively Parallel Methods for Deep Reinforcement Learning," *Computer Science*, 2015.
- [10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.
- [11] 3GPP TS 23.251 V15.0.0, "Network sharing; Architecture and functionalal description," 2018.
- [12] O. Sallent, J. Perez-Romero, R. Ferrus, and R. Agusti, "On radio access network slicing from a radio resource management perspective," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 166–174, 2017.
- [13] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "CellSlice: Cellular wireless resource slicing for active RAN sharing," *2013 5th International Conference on Communication Systems and Networks, COMSNETS 2013*, 2013.
- [14] M. Richart, J. Baliasian, J. Serrat, and J. L. Gorricho, "Resource Slicing in Virtual Wireless Networks: A Survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 462–476, 2016.
- [15] Z. Zhao, W. Chen, X. Wu, P. C. Chen, and J. Liu, "Lstm network: a deep learning approach for short-term traffic forecast," *IET Intelligent Transport Systems*, vol. 11, no. 2, pp. 68–75, 2017.
- [16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. Den, J. Schrittwieser, I. Antonoglou, V. Panneerselvam, M. Lanctot, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 1, pp. 484—489, 2016.
- [17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, and L. Sifre, "Article Mastering the game of Go without human knowledge," *Nature Publishing Group*, vol. 550, no. 7676, pp. 354–359, 2017.
- [18] Y. Wei, Z. Zhang, F. R. Yu, and Z. Han, "Power Allocation in HetNets with Hybrid Energy Supply Using Actor-Critic Reinforcement Learning," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–5, 2017.
- [19] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5g infrastructure markets: The business of network slicing," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.
- [20] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, "NVS: A substrate for virtualizing wireless resources in cellular networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1333–1346, 2012.
- [21] K. I. Pedersen, G. Berardinelli, F. Frederiksen, P. Mogensen, and A. Szufarska, "A flexible 5G frame structure design for frequency-division duplex cases," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 53–59, 2016.
- [22] R. D. Armstrong, D. S. Kung, P. Sinha, and A. A. Zoltners, "A computational study of a multiple-choice knapsack algorithm," *ACM Transactions on Mathematical Software (TOMS)*, vol. 9, no. 2, pp. 184—198, 1983.
- [23] F. Cummins, "Learning to Forget : Continual Prediction with 1 Introduction," no. October, 2000.

- [24] L. B. Godfrey and M. S. Gashler, "Neural decomposition of time-series data for effective generalization," *IEEE Transactions on Neural Networks & Learning Systems*, vol. PP, no. 99, pp. 1–13, 2017.
- [25] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [26] T. Chesebro and A. Kamko, "Learning Atari: An Exploration of the A3C Reinforcement Learning Method," 2016.
- [27] M. E. Helou, S. Lahoud, M. Ibrahim, K. Khawam, b. B. Cousin, D. Mezher, and B. Cousin, "A Hybrid Approach for Radio Access Technology Selection in Heterogeneous Wireless Networks," *Wireless Personal Communications*, vol. 86, no. 2, pp. 789–834, 2016.
- [28] B. Cao, S. Xia, J. Han, and Y. Li, "A distributed game methodology for crowdsensing in uncertain wireless scenario," *IEEE Transactions on Mobile Computing*, 2019.
- [29] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf, "Reinforcement Learning Through Asynchronous Advantage Actor-Critic on a GPU," *Iclr*, no. 1999, pp. 1–17, 2017.
- [30] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, no. 6, pp. 1291–1307, 2012.
- [31] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *Advances in Neural Information Processing Systems 12*, pp. 1057–1063, 1999.
- [32] K. Doya, "Reinforcement learning in continuous time and space." *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.
- [33] R. Li, Z. Zhao, J. Zheng, C. Mei, Y. Cai, and H. Zhang, "The Learning and Prediction of Application-Level Traffic Data in Cellular Networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3899–3912, 2017.
- [34] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, no. 671584, pp. 1–9, 2017.



**Mu Yan** received the B.S. degree in Electronic Engineering from the Beijing Jiaotong University, in 2014. He is now pursuing his Ph.D. degree in National Key Lab of Science and Technology on Communications, University of Electronic Science and Technology of China (UESTC). His research interests include AI-enabled wireless networking, next generation cellular networks, etc.



**Gang Feng** (M'01, SM'06) received his BEng and MEng degrees in Electronic Engineering from the University of Electronic Science and Technology of China (UESTC), in 1986 and 1989, respectively, and the Ph.D. degrees in Information Engineering from The Chinese University of Hong Kong in 1998. He joined the School of Electric and Electronic Engineering, Nanyang Technological University in December 2000 as an assistant professor and became an associate professor in October 2005. At present he is a professor with the National Laboratory of Communications, UESTC. Dr. Feng has extensive research experience and has published widely in wireless networking research. A number of his papers have been highly cited. He has received the IEEE ComSoc TAOS Best Paper Award and ICC best paper award in 2019. His research interests include next generation mobile networks, mobile cloud computing, AI-enabled wireless networking, etc. Dr. Feng is a senior member of IEEE.



**Jianhong Zhou** received the M.Eng. degree in Electronics and Electrical Engineering from the Nanyang Technological University (NTU), Singapore in 2008, and the Ph.D. degree in Computer Software and Theory from the University of Chinese Academy of Sciences in 2016. He is currently a Lecturer with the School of Computer and Software Engineering in Xihua University, China and postdoctor in National Key Lab of Science and Technology on Communications in University of Electronic Science and Technology of China (UESTC), respectively. Her research interests include next generation cellular networks, low latency ultra-reliable communication, etc.



**Yao Sun** received the B.S. degree in Mathematical Sciences from the University of Electronic Science and Technology of China (UESTC). He is currently working towards his Ph.D. degree at National Key Laboratory of Science and Technology on Communications, UESTC. His research and study interests include intelligent access control, handoff and resource management in mobile networks based on machine learning and other data analytics.



**Ying-Chang Liang** (F'11) is currently a Professor with the University of Electronic Science and Technology of China, China, where he leads the Center for Intelligent Networking and Communications and serves as the Deputy Director of the Artificial Intelligence Research Institute. He was a Professor with The University of Sydney, Australia, a Principal Scientist and Technical Advisor with the Institute for Infocomm Research, Singapore, and a Visiting Scholar with Stanford University, USA. His research interests include wireless networking and communications, cognitive radio, symbiotic radio, dynamic spectrum access, the Internet-of-Things, artificial intelligence, and machine learning techniques. Dr. Liang has been recognized by Thomson Reuters (now Clarivate Analytics) as a Highly Cited Researcher since 2014. He received the Prestigious Engineering Achievement Award from The Institution of Engineers, Singapore, in 2007, the Outstanding Contribution Appreciation Award from the IEEE Standards Association, in 2011, and the Recognition Award from the IEEE Communications Society Technical Committee on Cognitive Networks, in 2018. He is the recipient of numerous paper awards, including the IEEE Jack Neubauer Memorial Award, in 2014, and the IEEE Communications Society APB Outstanding Paper Award, in 2012. He was elected a Fellow of the IEEE for contributions to cognitive radio communications. He is the Founding Editor-in-Chief of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS: COGNITIVE RADIO SERIES, and the Key Founder and now the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING. He is also serving as an Associate Editor-in-Chief for China Communications. He served as a Guest/Associate Editor of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE JOURNAL OF SELECTED AREAS IN COMMUNICATIONS, the IEEE Signal Processing Magazine, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORK. He was also an Associate Editor-in-Chief of the World Scientific Journal on Random Matrices: Theory and Applications. He was a Distinguished Lecturer of the IEEE Communications Society and the IEEE Vehicular Technology Society. He was the Chair of the IEEE Communications Society Technical Committee on Cognitive Networks, and served as the TPC Chair and Executive Co-Chair of the IEEE Globecom'17.