Post-print: DRL-Based RAN Slicing Resource Allocation in Multi-Base Station Scenarios

Authors: Ma Yinghong, Jiang Lingyun

Date: 2022-05-10T11:22:58Z

Abstract

In fifth-generation mobile communications, network slicing is employed to provide an optimal network for diverse services. In the context of RAN slicing scenarios across multiple base stations, conventional resource allocation methods fail to meet slice requirements when the number of slices varies and are only applicable to specific scenarios. To address this issue, we propose a method that achieves optimal resource allocation independent of the number of slices. The method first utilizes the Ape-X method (a Deep Reinforcement Learning method) to allocate resources to slices, followed by slice-to-base station resource mapping and user resource allocation to satisfy user demands. Simulation results demonstrate that the proposed method can allocate resources according to slice status and requirements, distributing the necessary number of RBs to meet slice demands while remaining unaffected by variations in slice quantity. Furthermore, the method exhibits high generality and scalability.

Full Text

Preamble

RAN Slice Resource Allocation in Multi-Base Stations Based on DRL

Ma Yinghong, Jiang Lingyun†

(College of Telecommunications & Information Engineering, Nanjing University of Posts & Telecommunication, Nanjing 210003, China)

Abstract: In fifth-generation mobile communications, network slicing is used to provide an optimal network for various services. For RAN slicing scenarios under multiple base stations, previous resource allocation methods cannot meet slice demands when the number of slices changes and are only suitable for specific scenarios. To address this problem, this paper proposes a method to achieve optimal resource allocation independent of the number of slices. This method

first uses the Ape-X method (a DRL method) to allocate resources to slices, then satisfies user demands through slice-to-base station resource mapping and user resource allocation. Simulation results show that the proposed method can allocate resources according to slice states and demands, assigning the necessary number of RBs to meet slice requirements without being affected by changes in the number of slices. The method also demonstrates high general performance and scalability.

Keywords: multi base station; network slice; deep reinforcement learning; radio access network; resource allocation

0 Introduction

Fifth-generation (5G) mobile networks have attracted attention as a solution to the growing demand for mobile data communications. 5G improves upon several areas not adequately addressed in fourth-generation (4G) networks, such as higher data rates, lower end-to-end (E2E) latency, higher reliability, and massive device connectivity [1]. Moreover, service types in 5G are becoming increasingly diverse, including virtual reality (VR) requiring high data rates and low latency, and factory automation requiring massive device connectivity and low latency [2]. Traditional communication networks primarily served single mobile broadband services and cannot adapt to future diversified 5G service scenarios. Therefore, to simultaneously support multiple service scenarios with different performance requirements on the same physical network infrastructure and meet diverse service demands, network slicing technology has emerged. 5G employs network slicing technology to provide networks suitable for various services in slice units [3].

Slices establish requirements for throughput, latency, and reliability. To meet these demands, network resources are allocated to slices. Network slicing typically includes access network slicing (encompassing both wireless and fixed access) and core network slicing [4]. Among these, the Radio Access Network (RAN) must confront resource scarcity issues. Furthermore, when allocating wireless resources in practice, slice states continuously change, such as the number of users in a slice, service arrival rates, and user distribution. Consequently, a method capable of efficiently allocating wireless resources according to slice states while meeting slice requirements is needed [5,6].

Literature [7-12] presents several wireless resource allocation methods in single base station environments. The method in [7] meets slice demands by allocating resources from slices without requirements, but does not evaluate resource allocation utilization, creating the possibility of over-allocating resources to slices. Literature [8] proposes a method considering both slice demands and resource utilization, but cannot achieve resource isolation for each slice and is affected by the load of other slices. Literature [9] adopts an Earliest Deadline First (EDF) scheduling strategy for wireless resource allocation, which can meet slice

latency requirements under high load but severely impacts the performance of slices with throughput demands. Literature [10] proposes an online learning-based network slicing virtual resource allocation algorithm in C-RAN scenarios, aiming to maximize the average network slice sum rate while considering average network slice constraints and network average backhaul link bandwidth consumption constraints, but does not consider slice resource utilization. The method in [11] utilizes deep reinforcement learning, considers both slice satisfaction and resource utilization, and achieves resource isolation among slices, but only evaluates specific scenarios. Since 5G assumes various service scenarios, the method in [11] may not be applicable. Literature [12] proposes a network slicing resource scheduling mechanism based on online double auction, which can serve high-priority slices while guaranteeing the QoS demands of low-priority slice users, but does not consider resource isolation among slices.

The above literature studies wireless resource allocation methods in single base station environments, where allocation differs in multi-base station environments. Literature [13-15] presents several resource allocation methods in multi-base station environments. Literature [13] investigates dynamic network slicing strategies for mixed services in RAN, considering user QoS demands for delay and rate, but the slice quantity setting is not flexible enough. The algorithm in [14] considers base station backhaul capacity and satisfies the delay and data rate demands of different slice users well, but resource utilization is low under low load. Literature [15] designs and implements a two-layer Network Virtualization Substrate (NVS) algorithm that achieves slice resource scheduling based on slice priority and achievable rate. In this method, slices are allocated resources uniformly across base stations, which may lead to over-allocation at some base stations and under-allocation at others, resulting in suboptimal user satisfaction and resource utilization.

To address the problems in the above literature, this paper proposes a method using DRL to allocate wireless resources that meet slice requirements in multi-base station environments. Moreover, even when the number of slices changes, the proposed method can allocate resources to each slice to meet user QoS demands.

1 System Model

1.1 Network Model

This paper considers 5G base stations k, $k \in K$, where K is a set of base stations. Base stations provide services for slices s, $s \in S$, where S is the slice set. To meet user Quality of Service (QoS) demands, wireless resources must be allocated among slices. In 5G systems, wireless resources are represented by Resource Blocks (RBs). RBs are resource units divided based on time and frequency domains. One RB consists of 12 subcarriers, each with a spacing of 15 kHz. The time domain uses Transmission Time Interval (TTI) as the unit, with a TTI length of 1 ms. TTI is the minimum scheduling unit, and RBs are

allocated to users in each TTI.

Base station k allocates RBs to users to meet their demands. The achievable rate of user u on one RB is calculated as: $\langle MATH_0 \rangle$. Here, B represents the bandwidth of one RB, i.e., 15 kHz $\times$ 12 = 180 kHz. p_k denotes the transmission power of base station k, $\langle MATH_1 \rangle$ represents the channel gain of user u on one RB of base station k, and N_0 denotes the noise power spectral density. The achievable rate of user u on RBs differs across different base stations. Therefore, users establish a ranking based on the achievable rate on RBs from different base stations, from highest to lowest. The rank of user u for base station k is denoted as $\langle MATH_2 \rangle$, which can represent the importance of base station k to user u.

Users in different slices have different QoS demands. In this paper, demands are divided into two types: throughput demand and delay demand. For users requiring high throughput, RBs are allocated in each TTI; for users with delay demands, RBs are allocated when data packets arrive. Assuming user u' s throughput demand is R_u, i.e., the data rate demand. User u' s delay demand is T_u, and user u' s data packet size is p_u. Therefore, user u' s data rate demand is $\langle MATH_3 \rangle$. TTI is the minimum scheduling time unit, so user u' s data rate demand R_u needs to be converted to a rate demand over 1 ms. Additionally, this paper sets slice demands as user QoS demands, where users within the same slice have identical QoS demands.

This paper defines two metrics: Network Slice Demand Satisfaction (NSDS) and RB Usage Ratio (RBUR). The first metric, NSDS, measures whether the network meets the service demands. NSDS is expressed as: $\langle MATH_4 \rangle$. Here, k_s represents the number of users in slice s, and u_k indicates whether user k meets the slice demand.

The closer NSDS is to 1, the better the QoS demands of users in the slice are satisfied, enabling the provision of more suitable slices for services. RBUR is a metric measuring resource utilization, expressed as $\langle MATH_5 \rangle$, where R_{RB} represents the number of consumed RBs and ARB represents the number of RBs allocated to the slice. The closer RBUR is to 1, the higher the RB utilization and overall resource utilization.

When NSDS is low while RBUR is high, all RBs allocated to the slice are consumed, yet some users in the slice do not meet slice demands. Therefore, allocating more RBs to the slice can improve NSDS. By maximizing both NSDS and RBUR, slice demands can be met with the minimum number of RBs.

1.2 Problem Model

The RB allocation problem can be divided into two steps: inter-slice resource allocation and intra-slice resource allocation. The inter-slice resource allocation problem includes allocating RBs to slices and mapping slice resources to base stations. The inter-slice resource allocation problem is modeled as follows:

The objective is to maximize the product of slice resource utilization RBUR and slice satisfaction NSDS, meeting slice demands with minimal RB allocation. The first constraint in Equation (2) indicates that the total number of RBs allocated to all slices cannot exceed the total number of RBs owned by the base station. The second constraint indicates that under each base station, the total number of RBs obtained by all slices cannot exceed the number of RBs owned by that base station.

After determining the resource distribution of slices on each base station, the network slice controller executes intra-slice resource allocation. Users in a slice prioritize accessing the base station with the highest RB rank to maximize satisfaction of their QoS demands. Under base station k, let the set of RBs allocated to slice s be $M_{s,k}$. The intra-slice resource allocation problem is modeled as follows:

In Equation (3), A is the allocation matrix. If the n-th RB is allocated to the u-th user, then element $a(u,n)$ is 1; otherwise, it is 0. $U_{s,k}$ represents the utility of the s-th slice [16] on the k-th base station, as shown in Equation (4). This paper considers proportional fairness and selects $\alpha = 1$. In Equation (3), the first constraint indicates that under base station k, the total number of RBs allocated to users within each slice cannot exceed the resources available to that slice. The third constraint ensures that one RB can only be allocated to one user.

2 Methodology

2.1 Method Overview

In RAN slicing, a method is needed that meets slice demands with minimal RB allocation and is independent of the number of slices. Therefore, this paper proposes a flexible RB allocation method using Ape-X [17]. Since Ape-X is used, the model learned by the learner includes various experiences collected by each actor. Thus, when the number of slices changes, RB allocation can be performed without retraining the model. In existing methods, the number of slices controlled by an agent is fixed, so if the number of slices differs between training and evaluation, the model must be retrained. In the proposed method, one agent allocates RBs to one slice, and when multiple slices exist, the agent is called multiple times. This design enables RB allocation independent of the number of slices. After each RB allocation, the network slice controller updates the resource configuration of slices on each base station so that BSs can adapt to system states in each scheduling period.

Furthermore, the agent learns to allocate the minimum required number of RBs to meet demands, thereby maximizing the number of slices whose demands are satisfied while improving RB utilization efficiency. In this paper, one slice is defined for each service type. A slice is generated when the number of users in the slice becomes one or more, and terminated when the number of users becomes zero. The flowchart of the proposed algorithm is shown in Fig. 1.

2.2 RB Allocation Using Ape-X

The proposed method adopts the Ape-X approach, applying distributed learning to DRL. Here, one actor controls one slice. When the number of slices changes, the number of actors changes accordingly, with no limit on the number of slices, making RB allocation independent of slice quantity. The proposed method can flexibly set the number of slices and allocate RBs. The learner learns a policy that meets slice demands with the minimum number of RBs. The architecture of the allocation method is shown in Fig. 2. Since there is no limit on the number of slices, there are 1 to N slices, each containing multiple users. In addition to managing slices, the network slice controller bridges base stations and actors. Actors are Ape-X agents, with the same number as slices. Since actors use the policy trained by the learner, all actors have the same control strategy. In each resource scheduling period, RB allocation operations are executed. Base stations collect state information for each slice, including whether users in the slice meet QoS demands and slice resource utilization. The base station then notifies the network slice controller of the slice state. The network slice controller generates states and rewards based on slice states and passes them to the actor corresponding to the slice. The actor generates actions according to the policy and outputs them to the network slice controller. When the network slice controller receives actions from each slice, it calculates the number of RBs allocated to them.

During learning, rewards, states, and actions are passed to replay memory as experiences. After slice-level resource updates, resources need to be mapped to all base stations. Section 2.3 introduces the base station resource update algorithm flow. After base station-level resource updates, the network slice controller notifies each base station of the resource distribution of each slice on it. Each base station allocates RBs to users in each slice to meet user throughput and delay demands.

State is a crucial factor for the agent to determine actions. Better learning outcomes are achieved when states are designed to eliminate uncertain elements as much as possible. Based on this, this paper divides the state for learning RB allocation into three types given in Table 1. These three types are NSDS-related, RBUR-related, and slice state. First, NSDS-related information is important for the agent to identify slice demands. Second, RBUR-related information helps the agent identify RB allocation status for slices. The third type addresses state ambiguity.

Action is the control executed by the agent on the environment. The proposed method allocates RBs to each slice. The action output by the actor is denoted as a, which can be negative, zero, or positive. A negative a indicates a decrease in the number of RBs allocated to the slice, $a = 0$ indicates no change, and a positive a indicates an increase. The action a takes values from $\langle MATH_6 \rangle$, making it nine-dimensional. At time $t + 1$, the number of RBs allocated to the slice can be $\langle MATH_7 \rangle$. The calculated ARB does not consider RBs allocated

to other slices. However, since base stations have a limited number of RBs, the total number of RBs allocated to slices may exceed the base station' s total RBs. Therefore, the network slice controller adjusts the number of RBs allocated to each slice according to Algorithm 1. The product of the number of RBs allocated to each slice and NSDS at time t is calculated and sorted in ascending order. Resources are allocated to slices following this order. This prioritizes slices requiring fewer RBs or with smaller NSDS at time $t+1$, preventing slices requiring more RBs from occupying excess resources and improving resource utilization. On the other hand, this improves slice NSDS.

Reward indicates to the agent whether an action is good or bad for a state. In the proposed algorithm, the objective is to meet slice demands with minimal RB allocation, and maximizing NSDS and RBUR is the learning goal. The reward r is designed as $\langle MATH_8 \rangle$, where ARB is the number of RBs allocated to the slice and $Buff$ is the number of data packets stored in the buffer. When the buffer is empty, no RB allocation is needed, so if ARB is 0, r is 1; if ARB is not 0, r is 0. When data packets are stored in the buffer but ARB is 0 (i.e., the slice has demands but is not allocated RBs), r is 0. When data packets are stored in the buffer and ARB is not 0, NSDS and RBUR change with the allocated ARB. Therefore, r is calculated based on NSDS and RBUR.

The DQN framework is shown in Fig. 3, containing two neural networks: $\langle MATH_9 \rangle$ represents the predictive Q-network, where θ denotes the parameters of the predictive neural network used to evaluate the value of current state-action pairs; $\langle MATH_10 \rangle$ represents the target Q-network used to calculate target values, with θ^* denoting the target neural network parameters. The loss function $L(\theta)$ is the Temporal Difference (TD) error $\langle MATH_11 \rangle$. The gradient is calculated using the loss function, with the computation expression being $\langle MATH_12 \rangle$. This paper uses deep reinforcement learning to solve the inter-slice resource allocation problem and make optimal decisions. The components of the reinforcement learning architecture are detailed below.

This paper uses the RMSProp optimization algorithm to update network parameters. Every M steps, the predictive network parameters θ are copied to the target network parameters θ^*.

Algorithm 1: Slice RB Allocation

$\langle MATH_13 \rangle$ is the number of RBs calculated for allocation to slice s at time $t+1$, $\langle MATH_14 \rangle$ is the number of RBs allocated to slice s at time t, $\langle MATH_15 \rangle$ is the network slice demand satisfaction of slice s, S is the slice set, and $AllRB$ is the total resources owned by all base stations. $\langle MATH_16 \rangle$ is the number of RBs allocated to slice s at time $t+1$.

1) Begin

2) For $s \in S$

3) $W[s] = \langle MATH_1 7 \rangle$

4) End for

5) $remainRB = AllRB$

6) For $s \in S$ # Extract s in ascending order of W

7) If $remainRB > 0$

8) `$\langle MATH_18 \rangle$`

9) `$remainRB = remainRB - \langle MATH_19 \rangle$`

10) End if

11) End for

12) End

Since dueling networks [20] are adopted, $\langle MATH_2 0 \rangle$ is divided into the state value function $\langle MATH_2 1 \rangle$ and advantage function $\langle MATH_2 2 \rangle$. The state value function depends only on state s_t and is independent of action a_t. The advantage function depends on both state s_t and action a_t. The Q-value is expressed as: $\langle MATH_2 3 \rangle$.

The actor interacts with the environment $\langle MATH_2 4 \rangle$. This paper adopts the ϵ-greedy policy, where the actor selects the action with the highest action value with probability ϵ and a random action with probability $1 - \epsilon$. The actor calculates the TD error of experiences based on its network parameters, sets the priority p_k of experiences according to the TD error, and stores them in experience replay. The predictive and target networks use prioritized sampling instead of random sampling to extract experiences from experience replay. The sampling probability of experiences is $\langle MATH_2 5 \rangle$, with *batch* samples extracted each time, where k is the experience index. After updating network parameters, the TD errors and priorities of the *batch* samples are calculated, and the priorities of these samples in experience replay are updated. As learning progresses, action values are updated, the accuracy of old experiences decreases, and the priorities of old experiences stored in experience replay are updated to low values. When experience replay is full, old experiences are deleted.

Every N steps, the actor copies trained parameters from the predictive network to update its parameters to the latest version. This improves learning efficiency by prioritizing experiences with large TD errors while performing distributed learning to accelerate the process.

Fig. 3 DQN Framework

Fig. 4 shows the neural network structure of the proposed method. The network has an input layer, several hidden layers, and an output layer. Since the state dimension is designed as the input dimension, the input layer is eight-dimensional. The output layer is nine-dimensional, matching the action dimension. Hidden layers have 128 neurons. The fourth hidden layer branches into state value and advantage functions. Training was conducted for 2×10^6 steps using these specifications, taking approximately 2 days.

2.3.1 Base Station Resource Update

After slice resource updates, resources must be mapped to all base stations. Algorithm 2 describes the base station resource update process. Base station resource update relies on the weight of specific slices at base stations. Users in a slice have different RB ranks $\langle MATH_26 \rangle$ across different base stations, yielding the rank of slice s at base station k:

$\langle MATH_27 \rangle$

Therefore, the weight of slice s at base station k can be expressed as: $\langle MATH_28 \rangle$, representing the importance of base station k to slice s. These weights are updated in each scheduling period to update base station resources. The resources occupied by slice s at base station k are calculated as: $\langle MATH_29 \rangle$. The calculated $\langle MATH_30 \rangle$ does not consider the capacity limit of each base station, and the total number of RBs allocated to all slices at base station k may exceed base station k' s capacity. Therefore, resources allocated to slices at each base station need adjustment. All base stations are traversed, and base stations with allocated resources exceeding capacity are added to the reallocation queue Q_{BS}.

For base station k belonging to queue Q_{BS}, users belonging to this base station k are first identified. Based on users' RB ranks, the user set of slice s at base station k is obtained $\langle MATH_31 \rangle$, where user u prioritizes accessing the base station with the highest RB rank. The rate demand of slice s at base station k is:

$\langle MATH_32 \rangle$

where $\langle MATH_33 \rangle$ represents user u' s demanded rate. Therefore, the weight of slice s at base station k can be calculated as: $\langle MATH_34 \rangle$, yielding the number of RBs allocated to slice s at base station k as: $\langle MATH_35 \rangle$, where L_k is the total number of RBs owned by base station k.

After reallocation of base station resources, the total resources allocated to some slices may fall below ARB, requiring allocation of remaining base station resources to slices. First, the remaining resource amount of each base station and slices with insufficient allocated resources are checked. The queue of base stations with remaining resources is set as Q_{reBS}, and the queue of slices with

insufficient allocated RBs is set as Q_{slice}. Traversing base station queue Q_{reBS}, the weights $\langle MATH_36\rangle$ of all slices in slice queue Q_{slice} at this base station are compared, and the base station' s remaining RB resources are allocated to slices in descending order of weight until the slice' s RB demand is met or the base station' s resources are fully allocated.

Algorithm 2: Base Station Resource Update

Input: Number of RBs allocated to slice s $\langle MATH_37\rangle$, rank of slice s at base station k $\langle MATH_38\rangle$, requested rate of user u R_u, slice set S, base station set K, number of RBs lacking for slice $reRB$, RB capacity of base station k L_k, remaining RB resources of base station k L_{k_remain}, slice queue awaiting reallocation Q_{slice}, base station queue with remaining resources Q_{reBS}, number of RBs allocated to slice s at time $t+1$ $\langle MATH_39\rangle$.

Output: $\langle MATH_40\rangle$ is the number of RBs allocated by base station k to slice s.

1) Begin: Initialize Q_{slice}, Q_{reBS} queues
2) Step 1: Initial resource mapping
3) For $s \in S$
4) For $k \in K$
5) `$\langle MATH_41 \rangle$`
6) End for
7) End for
8) For $k \in K$
9) $\langle MATH_42\rangle$
10) If $\langle MATH_43\rangle$
11) `Add $k$ to $Q_{BS}$`
12) `Obtain a user set $U_{s,k}$, $\langle MATH_44 \rangle$`
13) `For $s \in S$`
14) `$\langle MATH_45 \rangle$`

15) $\langle MATH_46 \rangle$

16) End for

17) End if

18) If $\langle MATH_47 \rangle$

19) $\langle MATH_48 \rangle$

20) Add k to Q_{reBS}

21) End if

22) End for

23) Step 2: Base station remaining resource allocation

24) For $s \in S$

25) $\langle MATH_49 \rangle$

26) If $\langle MATH_50 \rangle$

27) Add s to Q_{slice}

28) End if

29) End for

30) End for

31) For $k \in Q_{reBS}$

32) Sort slices s in descending order of weight $\langle MATH_51 \rangle$ at base station k to obtain $Rank$

33) If $\text{len}(Q_{slice}) == 0$

34) Break

35) End if

36) For $s \in Rank$

37) If $\langle MATH_52 \rangle$

38) $\langle MATH_53 \rangle$

39) $\langle MATH_54 \rangle$

40) $\langle MATH_55 \rangle$

41) End if

42) If $\langle MATH_56 \rangle$

43) Break

44) End if

45) End for

46) If $\langle MATH_57 \rangle$

47) Break

48) End if

49) End for

50) End

2.3.2 User Resource Allocation and Connection Control

To solve the intra-slice resource allocation problem, this paper proposes an iterative solution. The allocation matrix A starts empty, and RBs are allocated to users through iteration. A gain factor is defined here: $\langle MATH_58 \rangle$, where $\langle MATH_59 \rangle$ represents the rate already allocated to user u, $\langle MATH_60 \rangle$ represents the achievable rate of user u on $\langle MATH_61 \rangle$, and $\langle MATH_62 \rangle$ is the unallocated RBs. If user u has the largest gain factor and this user' s data rate demand is not yet satisfied, then $\langle MATH_63 \rangle$ is allocated to this user. The process then enters the next iteration, stopping when all RBs are allocated or all user demands are satisfied.

After the iterative allocation process is completed for all slices under all base stations, it is checked whether all users have met their demands and whether slice resources have been fully utilized. Because too many users may access one base station while allocated resources are insufficient, while few users access

other base stations with abundant allocated resources. Therefore, to fully utilize resources and meet user demands, resources of slices at other base stations are allocated to users whose demands are not met. For users with unmet demands, following the RB rank order, other base stations are checked for remaining RB resources. If available, users are connected to the next base station, and the base station' s remaining RB resources are iteratively allocated to the newly connected users according to proportional fairness until all users in the slice meet their demands or all allocated resources to the slice are utilized.

3 Simulation and Evaluation

3.1 Simulation Overview

The proposed method is evaluated based on whether it achieves minimal RB allocation to meet slice demands and is unaffected by changes in the number of slices. In the proposed method, the RB allocation model must first be trained. The trained model is then used to evaluate the proposed method. The evaluation consists of three types. The first demonstrates that the proposed method appropriately implements RB allocation in the created specific scenario. The second shows the general performance of the model evaluated based on multiple randomly generated scenarios. In the third type, this paper evaluates the relationship between the number of slices and performance and describes the scalability of the proposed method.

3.2 Training

The model in the proposed method learns RB allocation that maximizes NSDS and RBUR from slice states. In 5G, various service types are assumed. Therefore, simulation scenarios are randomly generated, and the model is trained using various service types. Table 2 provides the scenarios used for training. Table 3 provides common parameters for training and evaluation. A new scenario is generated after each simulation. The number of slices is fixed at three, but during simulation, the number of slices varies from 0 to 3 because slice start and end times differ. Additionally, the number of users in each slice, packet generation intervals, and packet sizes differ. Slice demands are throughput demand, delay demand, or both. According to Long Term Evolution (LTE) specifications, subcarrier spacing and TTI are set to 15 kHz and 1 ms, respectively. System bandwidth is set to 20 MHz, with a total of 100 RBs per TTI. Since RBs are grouped according to specifications, the number of RBs controlled by each base station is 25. The RB allocation control interval is 1 ms, the same as TTI.

Table 4 provides Ape-X parameter values, identical to those in [17]. The number of actors is a parameter set based on computer performance. The training computer configuration is AMD 3700x CPU, 32GB RAM, and RTX 2070 Super GPU. The simulation computer is 3700x CPU with 16GB RAM. This paper uses one computer for learning and six computers for simulation. Five simulations run on one computer, and one simulation has 4 actors, totaling 120 actors (6

computers × 5 simulations × 4 actors). The discount factor determines the weighting of future rewards. In RB allocation, slice states also change rapidly, so allocating RBs quickly according to state changes is important. Therefore, this paper sets the discount factor to 0.5 to maximize short-term rewards. For the same reason, n is set to 1.

3.3.1 RB Allocation Evaluation

This section evaluates whether the proposed method can allocate RBs to slices in scenarios with different numbers of slices. The considered mobile network scenario is based on 5G network standards, with parameters summarized in Tables 3 and 5. In a given 500m × 500m area, four base stations are uniformly distributed. The distance between every two adjacent BSs is fixed at 120m. The Path Loss (PL) model is defined as: $PL(dB) = 20\log_{10}(d) + 20\log_{10}(f) - 27.55$, where d (in meters) and f (in MHz) represent the user-base station distance and channel frequency, respectively. Based on 5G slice categories, this paper defines four slices: messaging service, application, audio, and video, with each slice matching one service type. Each slice has different numbers of users, packet lengths, and slice demands. Changes in the number of slices are simulated by setting slice start and end times. In the simulation, the number of slices varies from a minimum of two to a maximum of four. Specific parameters are shown in Table 5. The number of users in slices changes over time, as shown in Fig. 5.

Two methods are compared in this paper, described as follows:

a) **Hard-slicing**: The hard method divides all RBs equally among slices. The ARB for slice s can be calculated as: $\langle MATH_6 4\rangle$.

b) **NVS (Network Virtualization Substrate) method** [18]. Resources are allocated based on the weight of slices in the system. The weight of slice s is calculated as: $\langle MATH_6 5\rangle$, defined as the aggregated data rate request of all users in slice s. Therefore, the RBs allocated to slice s are calculated as: $\langle MATH_6 6\rangle$. In the NVS method, the amount of resources provided for each slice is equally distributed among base stations. The proposed method and the hard method adopt the resource mapping approach described in Section 2.3.

Figs. 6(a) to 6(c) show the relationship between NSDS, RBUR, ARB, and simulation time. Here, RBs are allocated to slices at one-millisecond intervals, but NSDS and RBUR are measured as one-second averages at one-millisecond intervals, and ARB is the total ARB per second (100 × 1000 RBs). The NSDS results show that the proposed method almost completely meets slice demands. In the proposed method, when the number of slices changes at 110s, 205s, and 400s, NSDS does not decrease. Based on these results, slice performance demands can be satisfied even when the number of slices changes.

In the hard method, RB allocation is related to the number of slices. Therefore, slices 1 and 3 with low data rate demands always meet slice demands. Slice

2 has many users, and between 320s and 400s, the number of users in slice 2 continuously increases, but the number of allocated RBs does not increase, resulting in insufficient resource allocation and decreased NSDS. Slice 4 requires high throughput, and between 350s and 400s, the number of users increases, increasing data rate demands, but ARB does not change, causing NSDS to decrease.

In Fig. 6(c), according to the NVS method, resources allocated to slices are related to slice data rate demands. The higher the data rate demand, the more resources obtained. Therefore, Fig. 6(c) shows that slice 1' s demands are met between approximately 90s and 160s. Between 160s and 200s, as the number of users in slice 2 continuously increases, slice 2' s data rate demand grows higher, and slice 2 obtains more resources, causing slice 1' s NSDS to decrease. Between 200s and 500s, slice 4 occupies excessive resources due to its high throughput demand. Around 400s, slice 4 has the maximum number of users and obtains the most RBs, while other slices' NSDS reaches the minimum. Slice 3 has low data rate demand and few users, thus obtains few RBs, resulting in very low NSDS and inability to meet slice delay demands.

Based on RBUR results, the proposed method' s RBUR is about 0.75 or higher, indicating RB over-allocation is less than 25%. In the hard method, since RBs are uniformly allocated to slices regardless of slice states and demands, RBUR performance is poor, with over-allocation in every slice, as shown in Fig. 6(b). For the NVS method, slices 2 and 4 have high data rate demands, resulting in severe over-allocation and poor RBUR performance. Slices 1 and 3 have low data rate demands, leading to insufficient RB allocation, but all allocated RBs are utilized, thus achieving relatively high resource utilization.

In summary, based on NSDS and RBUR results, the proposed method performs better than other methods in both metrics, can allocate resources according to slice states and demands, assigns the necessary number of RBs to meet slice demands, and is unaffected by changes in the number of slices.

3.3.2 General Performance Evaluation

This section evaluates the general performance of the proposed method using scenarios that simulate various services. As with general machine learning including DRL, the optimal solution for target data can be estimated by training only on specific data, but correct estimation cannot be made for other untrained data. This is called overfitting to the environment. When only targeting specific time zones, locations, or services, such a model is effective. However, if the model is only used for specific cases, the advantages of using network slicing are lost. This is because there are many service types in RAN, and network slicing is a technology that adapts networks to various services.

The general performance of the proposed method is evaluated using randomly generated scenarios from Section 3.2. Since this scenario randomly determines the number of users, packet generation intervals, and slice demands, it can

simulate various service types. Note that not all generated scenarios can be explained by existing services. This paper evaluates untrained scenarios using different seed values from training to show the model' s generalization capability. A total of 3000 scenarios were tested. Evaluation metrics are NSDS and RBUR, measured as one-second averages. Results are presented as cumulative distribution functions (CDF) and mean values of measured data. High NSDS and RBUR indicate high performance across various randomly generated scenarios.

Figs. 7(a) and 7(b) show NSDS evaluation results. The proposed method' s average NSDS is about 0.92, almost meeting slice demands. The average NSDS of comparison methods is below 0.7. Fig. 7(b) shows that the hard slicing method' s NSDS is distributed at about 28% at 0.0 (where no users meet demands) and at about 50% at 1.0 (where all users meet demands). The results indicate a 28% probability that one user in a slice cannot meet demands. For the NVS method, there is a 31% probability that one user in a slice cannot meet demands, and a 50% probability that all users' demands in a slice are fully satisfied.

The proposed method' s NSDS is distributed at less than 10% at 0.0 and about 82% at 1.0. The results indicate that the probability of one user in a slice being unable to meet demands is less than 10%, while the probability that all users' demands in a slice are fully satisfied reaches 82%. Based on these results, the proposed method achieves high-level general performance, almost meeting slice demands in various scenarios.

Figs. 8(a) and 8(b) show RBUR evaluation results. The proposed method' s average RBUR is about 0.77. The average RBUR of comparison methods is below 0.7, with resource over-allocation exceeding 30%. The hard method has the lowest average RBUR because it equally distributes all RBs to each slice, causing over-allocation in some slices. The NVS method allocates resources based on different slices' data rate requests, but some slices may request much higher rates than others, causing some slices to be allocated excessive resources and resulting in low RBUR.

In summary, for randomly generated scenarios, the proposed method can effectively allocate RBs to slices, almost meeting various slice demands, achieving high-level general performance while maintaining high resource utilization.

3.3.3 Scalability Evaluation

This paper evaluates the scalability of the proposed method regarding the number of slices by assessing the relationship between slice quantity and performance. This evaluation uses the randomly generated scenarios from Section 3.2, with the number of slices selected from 1 to 8 according to a uniform distribution for each scenario. The number of slices during evaluation is calculated not by the number of slices set during scenario creation but by the number of slices running simultaneously. For example, if the number of slices is set to 7 but only 3 slices run simultaneously, it is counted as 3 slices. Evaluation metrics

are NSDS and RBUR, measured as one-second averages.

Fig. 9 shows the relationship between the number of slices and NSDS. For all tested methods, NSDS decreases as the number of slices increases. When the number of slices increases, more RBs are needed. NSDS decreases because the required number of RBs for slices cannot be guaranteed. When the number of slices reaches 4 or more, the proposed method' s NSDS is more than 0.2 higher than other methods, achieving better performance. When the number of slices is less than 8, NSDS is above 0.8, meaning more than 80% of users in slices fully meet QoS demands. When the number of slices is 8, NSDS is about 0.78. Among comparison methods, the NVS method performs worst because, on one hand, some slices may have high data rate demands while others have low demands, causing slices with low data rate demands to obtain too few RBs and resulting in low NSDS; on the other hand, the NVS method equally distributes resources among base stations, potentially causing over-allocation at some base stations and under-allocation at others, leading to users accessing base stations with insufficient resources and reducing demand satisfaction. The hard method equally distributes resources to each slice, and when the number of slices increases, each slice receives fewer resources, causing some slices to have insufficient allocated resources and decreasing demand satisfaction.

Fig. 10 shows the relationship between the number of slices and RBUR. In the proposed method, when the number of slices exceeds 4, RBUR decreases as the number of slices increases, with RBUR at 0.7 when the number of slices is 8. RB over-allocation is intended to reliably meet slice demands when allocating RBs. The proposed method learns allocation that prioritizes NSDS over RBUR, related to reward design. When designing rewards, both resource utilization and slice demand satisfaction are considered. If all users cannot meet demands, NSDS is 0, but if some users use RBs, RBUR is greater than 0. The reward is NSDS multiplied by RBUR, so if either becomes 0, the reward also becomes 0. During learning, NSDS close to 0 is prioritized to improve slice demand satisfaction, leading to RB over-allocation.

In the NVS and hard methods, when the number of slices is 1, the RBUR of both hard and NVS methods is less than 0.3 because these methods allocate all resources to this slice, causing low resource utilization. When the number of slices increases, more slices require resources, improving slice resource utilization. However, due to limitations of these two methods, RBUR performance is worse than the proposed method, with severe over-allocation.

Based on NSDS and RBUR results, even when the number of slices changes, the proposed method can allocate RBs to meet slice demands for throughput and delay. Moreover, the proposed method handles changes in the number of slices by simply creating or terminating actors that perform RB allocation using the trained model. In summary, the proposed method has high scalability in terms of the number of slices.

4 Conclusion

To efficiently allocate wireless resources while meeting slice demands, this paper proposes an RB allocation method using Ape-X that is unaffected by the number of slices. Simulation results show that the proposed method can allocate resources according to slice states and demands, assigning the necessary number of RBs to meet slice demands without being affected by changes in the number of slices, while achieving high-level general performance for randomly generated scenarios.

The proposed method is designed based on TTI in LTE. In 5G, TTI is variable, and future research will continue in this direction.

References

[1] NGMN Alliance. 5G White Paper [EB/OL]. [2015-02-17]. https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf.
[2] Elayoub S E, Fallgren M, Spapis P, et al. 5G service requirements and operational use cases: Analysis and METIS II vision [C]//2016 European Conference on Networks and Communications (EuCNC). Athens: IEEE, 2016: 158-162.
[3] Afolabi I, Taleb T, Samdanis K, et al. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions [J]. IEEE Communications Surveys & Tutorials, 2018, 20(3): 2429-2453.
[4] NGMN Alliance. Description of network slicing concept [EB/OL]. [2018-02-20]. http://www.ngmn.org/fileadmin/user_upload/160113_Network_Slicing_v1_0.pdf.
[5] Foukas X, Patounas G, Elmokash A, et al. Network Slicing in 5G: Survey and Challenges [J]. IEEE Communications Magazine, 2017, 55(5): 94-100.
[6] Elayoubi S E, Jemaa S B, Altman Z, et al. 5G RAN Slicing for Verticals: Enablers and Challenges [J]. IEEE Communications Magazine, 2019, 57(1): 28-34.
[7] Shrivastava R, Samdanis K, Bakry A. On Policy Based RAN Slicing for Emerging 5G TDD Networks [C]//2018 IEEE Global Communications Conference (GLOBECOM). Emirates, United Arab Emirates: IEEE, 2018: 1-6.
[8] Chang C Y, Nikaein N, Spyropoulos T. Radio Access Network Resource Slicing for Flexible Service Execution [C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Honolulu, HI, USA: IEEE, 2018: 668-673.
[9] Guo T, Suarez A. Enabling 5G RAN Slicing with EDF Slice Scheduling [J]. IEEE Transactions on Vehicular Technology, 2019, 68(3): 2865-2877.
[10] Tang Lun, Wei Yannan, Ma Runlin, et al. Online Learning-based Virtual Resource Allocation for Network Slicing in Virtualized Cloud Radio Access Network [J]. Journal of Electronics & Information Technology, 2019, 41(7): 1533-1539.
[11] Abiko Y, Mochizuki D, Saito T D, et al. Proposal of Allocating Radio Resources to Multiple Slices in 5G Using Deep Reinforcement Learning [C]//2019

IEEE 8th Global Conference on Consumer Electronics (GCCE). Osaka: IEEE, 2019: 131-132.
[12] Chen Qianbin, Shi Yingjie, Yang Xixi, et al. Resource Scheduling Mechanism for Virtual Network Slice Based on Online Double Auction [J]. Journal of Electronics & Information Technology, 2018, 40(7): 1738-1744.
[13] Sun G L, Xiong K, Boateng G O, et al. Autonomous Resource Provisioning and Resource Customization for Mixed Traffics in Virtualized Radio Access Network [J]. IEEE Systems Journal, 2019, 13(3): 2454-2465.
[14] Xiong K, Adolphe S Xiang R, Boateng G O, et al. Dynamic Resource Provisioning and Resource Customization for Mixed Traffics in Virtualized Radio Access Network [J]. IEEE Access, 2019, 7: 115449-115458.
[15] Kokku R, Mahindra R, Zhang H, et al. NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks [J]. IEEE/ACM Transactions on Networking, 2012, 20(5): 1333-1346.
[16] Caballero P, Banchs A, Veciana G D, et al. Network slicing games: Enabling customization in multi-tenant networks [J]. IEEE/ACM Transactions on Networking, 2019, 27(2): 662-675.
[17] Horgan D, Quan J, Budden D, et al. Distributed prioritized experience replay [EB/OL]. [2018-03-02]. http://arxiv.org/abs/1803.00933.
[18] Hasselt H V, Guez A, Silver D. Deep reinforcement learning with double Q-learning [EB/OL]. [2015-12-08]. http://arxiv.org/abs/1509.06461.
[19] Sutton R S, Barto A G. Reinforcement Learning: An Introduction [J]. IEEE Transactions on Neural Networks, 1998, 9(5): 1054-1054.
[20] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for reinforcement learning [EB/OL]. [2016-04-5]. http://arxiv.org/abs/1511.06581.

Note: Figure translations are in progress. See original paper for figures.