Efficient Network Slicing with Dynamic Resource Allocation

JI Hong[1], ZHANG Tianxiang[2], ZHANG Kai[1], WANG Wanyuan[1], WU Weiwei[1]

(1. School of Computer Science and Engineering, Southeast University, Nanjing 211189, China;
2. ZTE corporation, Shenzhen 518057, China)

Abstract: With the rapid development of wireless network technologies and the growing demand for a high quality of service (QoS), the effective management of network resources has attracted a lot of attention. For example, in a practical scenario, when a network shock occurs, a batch of affected flows needs to be rerouted to respond to the network shock to bring the entire network deployment back to the optimal state, and in the process of rerouting a batch of flows, the entire response time needs to be as short as possible. Specifically, we reduce the time consumed for routing by slicing, but the routing success rate after slicing is reduced compared with the unsliced case. In this context, we propose a two-stage dynamic network resource allocation framework that first makes decisions on the slices to which flows are assigned, and coordinates resources among slices to ensure a comparable routing success rate as in the unsliced case, while taking advantage of the time efficiency gains from slicing.

Keywords: network slicing; dynamic resource allocation; reinforcement learning

DOI: 10.12142/ZTECOM.202101003

https://kns.cnki.net/kcms/detail/34.1294.TN.20210208.1133.002.html, published online February 8, 2021

Manuscript received: 2020-12-09

Citation (IEEE Format): H. Ji, T. X. Zhang, K. Zhang, et al. "Efficient network slicing with dynamic resource allocation," *ZTE Communications*, vol. 19, no. 1, pp. 11 – 19, Mar. 2021. doi: 10.12142/ZTECOM. 202101003.

1 Introduction

With the fast growth of network technologies, such as 5G and data center networks, and ever-increasing demand for the high quality of service (QoS)[1], efficient employment of existing network infrastructure becomes a challenging task. Network slicing provides an effective method that can introduce flexibility and faster resource deployment in network resource management[2]. A slice is a horizontal subset of the entire network which is set to satisfy resource requests, for example, bandwidth for flows. A flow is a certain amount of bandwidth requirement on the passing links[3].

Many studies have been devoted to making full use of QoS and network resource utilization for traffic scheduling, for which queue management and scheduling are widely used. Generally, the flows in the queue may have different priorities, for example, the preceding flows may have a higher priority than other flows[4]. In floodlight-based software defined networks (SDN)[5], queue-based scheduling technology is widely used to implement QoS support. In the above research, the traffic to be transmitted is divided into QoS flows and optimal flows, and assigned to the queue according to their priorities[6].

However, the queuing nature of the above resource allocation or scheduling strategy has shortcomings. It is necessary to calculate the feasible route of the flow/request in real-time by checking the remaining available bandwidth on the network link. However, it is time-consuming to calculate the feasible route for the flows in the queue which need to be processed in

sequence. Inspired by network slicing, which has revealed an acceleration effect on routing[7], we introduce a network slicing model to parallel and speed up routing calculations. In the proposed network slicing model, different slices (e.g., horizontal slices) share the same topology, but have different bandwidth resources on the link. Flows are allocated to different slices, and routed in an independent manner in each slice. Finally, the routing process can be calculated in parallel, thus speeding up the routing process.

The main challenge of network slicing is to determine to which slice the request should be deployed. The objective of resource allocation is to ensure a high deployment success rate, for which some flows may not be deployed due to scarce resources. Moreover, using the checking mechanism to test whether the route is feasible or not will degrade the efficiency of parallel routing. Against this background, the first contribution of this paper is that we propose a two-stage resource allocation algorithm, which consists allocation to slice and the resource adjustment among slices. In the first stage of the flow allocation, given the current arrival flows, a reinforcement learning (RL) based mechanism is proposed to deploy these flows. In the second stage of resource adjustment, a dynamic and parallel network resource reallocation among slices is proposed. Another contribution is that we conduct an experimental evaluation in a hierarchical ring 5G network similar to a real large-scale network. Simulation results show that this method can still reduce the routing calculation time and maintain the deployment success rate when dealing with large-scale networks.

The rest of this paper is organized as follows. Section 2 reviews the related work on resource allocation of network technologies. Section 3 describes the problem and forms the model. We propose the resource allocation algorithm in Section 4. Section 5 presents the results of the experiments. Finally, we conclude this paper in Section 6.

2 Related Works

In recent years, network slicing and effective use of network resources have received a lot of attention. SDN is the candidate technology for implementing network slicing on common network infrastructure for deploying a number of services with different requirements. Deployed slices guarantee the isolation in terms of connectivity and performance[8].

Network resource virtualization (slicing) has been regarded as one of the main development trends of 5G cellular networks and data center networks, which can improve QoS, quality of experience (QoE), and network resource utilization[9]. Through network slicing, the whole network can easily accommodate the different needs of divergent service types, applications, and services in support of vertical industries[10]. A virtualized infrastructure is provided in Ref. [11], which is a network infrastructure in which some or all of the elements are virtualized by the data center network, such as servers, links, switches and topology. The cloud computing platform consists of single or multiple virtualized infrastructure, which relies on virtualization technology to divide available resources and share them among users.

For the limitation of network resources, a resource allocation plan can be implemented to improve communication reliability and network utilization. However, slicing may cause severe network performance degradation. ZHANG et al. [12] use a supply-demand model to quantify slicing interference. In order to maximize the total throughput of accepted requests, an adaptive interference awareness (AIA) heuristic method is proposed to automatically place slices in network slices customized for 5G services. CHEN et al. [13] also develop a dynamic network slice resource scheduling and management method based on SDN to meet the services' requirements with time-varying characteristics. A resource combination and allocation mechanism is proposed to maximize the total rate of the virtualized network based on its channel state information[14]. An algorithm based on iterative slice provision has been proposed, which adjusts the minimum slice requirement based on channel state information, but does not consider the global resource utilization rate of the network and slice priority. A centralized joint power and resource allocation scheme for priority multi-layer cellular networks has been proposed[15], which works by allowing users with higher priority to maximize the number of service users. Priority is only considered at the user level, and different priorities between slices are ignored.

In this paper, we formulate the network slicing model, which enables flow requests to be dispatched among different slices, thereby speeding up routing computations. Moreover, the proposed algorithm is validated to have the advantage of maximizing the success rate of flow deployment.

3 Problem Formulation

3.1 Network Formulation

We model the wide area network (WAN) as an undirected graph $G = \{V, E, A\}$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of nodes which can represent switches or routers in the WAN, and $E = \{(v_s, v_d)_i | i = 1,2,...,m\}$ is a set of edges in the graph which can represent link connections between WAN nodes. Let $A = \{a_1, a_2, \ldots, a_m\}$ represent the available bandwidth per link. We use the abbreviation e_{sd} or abbreviation of edge number $e_i, i = 1, 2, ..., m$, to denote an edge between v_s and v_d. We use a_{sd} or a_i to denote the available bandwidth of a specific link. The notations to be used in this section can be found in **Table 1**.

Define a flow f_j as users' requests, indicating that at each time period, a user requests certain channels of a pre-defined transmission rate (bandwidth demand). Let $F = \{(v_s, v_d)_j | j = 1,2,...,h\}$ denote a flow set, and d_j denote the pre-defined

transmission rate of a flow. Each flow can be represented by a path that consists of a series of links, which is generated by the shortest path (SP) routing scheme. Note that a feasible SP route scheme depends on the current network state especially each link's available bandwidth. Let the path calculated by a certain routing scheme for f_j as P_j. After determining a routing scheme, we can get all links in P_j, so P_j also can be represented as a set of links. These links should also satisfy the constraints described above: $d_j \leqslant a_i, \forall e_i \in P_j$.

3.2 Network Slicing

Assuming we have a real network and several flows, after allocating some bandwidth resources to a flow, the network state will be changed, so the next flow's routing scheme must depend on the changed network state. If they are planned together, some conflicts may happen, for the reason that, to deploy a bunch of flows, the shortest path should be calculated sequentially to avoid deployment failures of following flows. For example, there are two flows, namely f_1 and f_2, established on a network in **Fig. 1**, where $f_1 = \{v_1, v_3, 80\}$ and $f_2 = \{v_1, v_3, 60\}$ if their shortest paths are calculated at the same time. Both of them will have the same path $v_1 \rightarrow v_2 \rightarrow v_3$, but the link between v_1 and v_2 cannot hold these two flows, so we need to calculate paths sequentially.

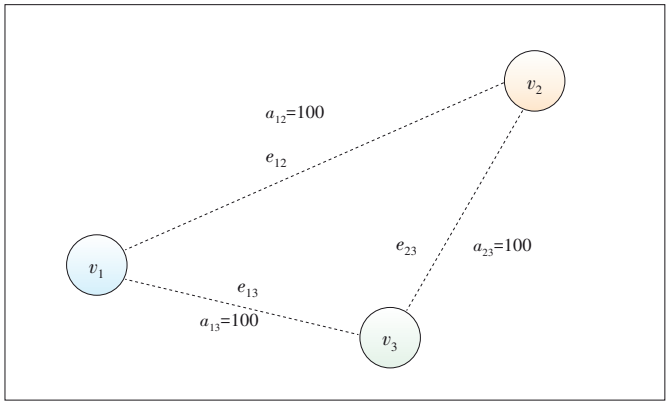
Due to this, the total time for planning all flows is the sum of time planning for each flow. In the real production environment, we always expect a minimum deployment time. To achieve this goal, we need to establish some resource isolation, so that paralleling computing can be applied to the separate resource part. This is a method called slicing which is what we are going to introduce in the following. First, we introduce the slice set $S = \{s_1, ..., s_k, ..., s_K\}$, which represents a set of slices. Each slice has the same topology as G, and each links' capacity is part of the original network. We call the process of dividing bandwidth into parts slicing. Assuming that the original network is divided into K slices, in this scenario, the total time of calculating route paths is the maximum time consumed among all slices. If we have K slices, the time consumed can be approximately reduced to $1/K$ of original time-consuming.

All slices can be seen as virtual networks based on the same physical network. In detail, we can also formulate a slice as an undirected graph $s_k = \{V, E, A_k\}$, which has the same topology as the original network. Additionally, the delay of an edge is the same across slices.

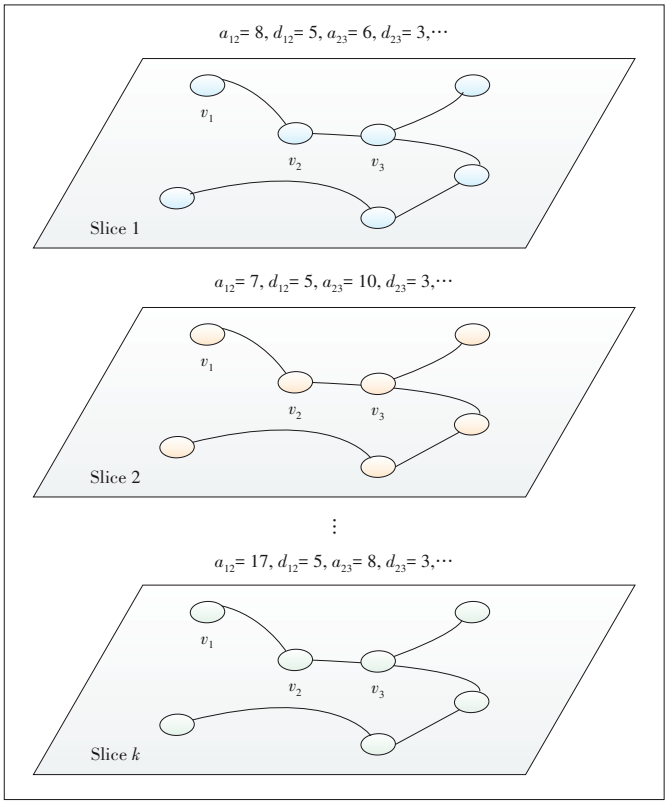
Different slices have the same network topology and delay, while their bandwidths and available bandwidths can be different, which depends on the slicing method. **Fig. 2** illustrates

▼**Table 1. Notation overview**

Notation	Description
$V = \{v_1, ..., v_n\}$	the set of nodes
$E = \{(v_s, v_d)_i \mid i = 1, 2, ..., m\}$	the set of edges
e_i or e_{sd}	a certain link
$A = \{a_1, ..., a_m\}$	available bandwidth per link
$G = \{V, E, A\}$	the set of a network
$F = \{(v_s, v_d)_j \mid j = 1, ..., h\}$	the set of flows
f_j or f_{sd}	a certain flow
d_j	demand of a flow
P_j	a path the flow j takes
$S = \{s_1, ..., s_k, ..., s_K\}$	set of slices
$f_j^k \in F^k$	set of flows to be deployed in slice k



▲**Figure 1. A network with 3 nodes and 3 edges**



▲**Figure 2. Network slices in our slicing model**

the slicing method in our slicing model.

3.3 Resources Allocation

Given the network slicing, by randomly allocating users' flow requests to slices, these flows in different slices can be routed by SP in parallel, which in turn can reduce total routing time. However, compared with routing on the original network, this network slicing-based routing mechanism will lead to a higher failure rate, since the network resources might be sliced in unavailable fragments. Therefore, this method of directly slicing reduces the flexibility of routing compared with routing on the original network without slicing. In other words, this method improves the calculation time performance at the expense of the success rate of routing.

To improve the flow deployment success rate, we design an appropriate method to determine which slice the flow should be deployed in. We use f_j^k to denote that f_j will be deployed in s_k, $F^k = \{f_1^k, \ldots, f_h^k\}$ represent the set of flows deployed in s_k, and we also use F^k to represent the set of flows F_{succ}^k successfully deployed in s_k.

Our objective is to design a flow allocation algorithm to maximize the success rate of flow deployment on slices, which can be formulated as $\max \sum_{i=1}^{K} |F_{succ}^i|$. Particularly, we hope that the success rate after slicing can be close to the success rate of deployment on the original topology.

4 Algorithm

In this section, we propose a two-stage network resource allocation algorithm. In the first stage, once the flow requests are received, we propose a RL-based mechanism to allocate these flows to slices in sequence. In the second stage, aware of the imbalance of resources among slices, we propose a real-time resource adjustment mechanism to balance the resources dynamically.

4.1 RL Based Flow Allocation

We use reinforcement learning to train a general policy for specific topology and flow, which is used to determine on which slice each flow is deployed to maximize the success rate.

RL is a field of machine learning that emphasizes how to act based on the environment to maximize the expected benefits[16]. The basic reinforcement learning model is defined by the tuple $< S, A, R, P >$, where S is the set of states, A is the set of actions, R is the reward function and P is the state transition probability. We formulate the flow deployment with network slicing problem as a Markov decision process (MDP), shown as follows:

• Observation state: The observation obtained by the agent from the simulation environment is composed of two parts. One is the current state of all slices, and the other is the information of the flow to be deployed. We use *obs* and *obf* to represent these two parts. So, the state $s \in S$ can be denoted by $s = \{ob_s, ob_f\}$. More specifically, ob_s is represented by the adjacency matrix of the available bandwidth of each slice. ob_f includes the source node and the target node, and the size of the current flow f_j.

• Action: The action space is a discrete space of slice index. Specifically, our RL agent selects the slice number to deploy for each flow, so we use $A = \{1,2,\ldots,K\}$ to represent the action space, where K is the number of slices.

• Reward: Reward is numerical feedback obtained by the agent after taking an action according to the current state and applying it in the environment. The magnitude of the value can reflect the quality of the action, and the reinforcement learning design is used to maximize the reward value in the long-term range. In our situation, the specific form of the reward is shown in Eq. (1) below.

$$R(f_j, a) = s_j^a \times \frac{K}{\sum_{k=1}^{K} \mathbb{I}\left(s_j^a = s_j^k\right)}, \tag{1}$$

in which $s_j^k \in \{-1, 1\}$ is a binary variable, when a flow f_j to be deployed is successfully deployed on the the slice s_k. The value of s_j^k is 1, otherwise the value is -1. The $\mathbb{I}(\cdot)$ is an indicator function, and the condition is 1, otherwise, it is 0.

The reward function designed in this way can properly reflect the correctness of the decision made by the RL agent. When most of the slices can correctly deploy the flow, it is easy to make a decision, so a small reward will be generated. Moreover, if most of the slices cannot deploy the flow correctly, and the RL agent makes a correct decision, it will produce a large positive reward. Conversely, the wrong decision will always produce a negative reward. The absolute value of the reward is related to the difficulty of making a wrong decision.

• Transition probability: The MDP transition probability $p(s_t + 1 | s_t, a_t)$ is deterministic, which depends on the way that the flow routing method and the resource adaptation between slices after the flow is deployed. It will be introduced in the next section.

We want to train an agent to decide which slice to assign based on the state of each slice. The process that the RL policy interacts with the environment and the specific elements of the learning environment can be seen in **Fig. 3**.

1) RL training mechanism: We use an asynchronous proximal policy optimization (APPO) algorithm[17] based on the importance weighted actor-learner architecture (IMPALA)[18] to train our agent. Compared with synchronous proximal policy optimization (PPO), APPO is more efficient in wall-clock time due to its use of asynchronous sampling. Using a clipped loss also allows for multiple stochastic gradient descent (SGD) passes, and therefore the potential for better sample efficiency compared with IMPALA. Meantime, V-trace can also be en-

abled to correct for off-policy samples. The PPO-based flow allocation algorithm is described in Ref. [17]. It repeatedly uses the data obtained by sampling to update the strategy parameters with gradient ascent until the strategy is no longer updated. The specific proof of convergence can be seen in the original paper[17].

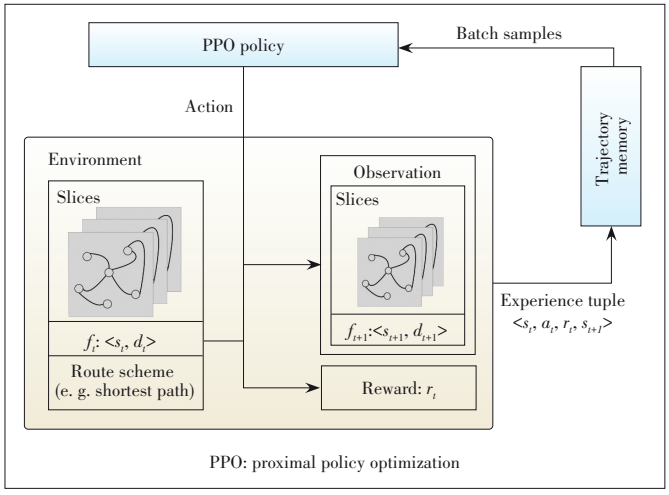
The PPO is more stable than the Q-learning algorithm learning process. This training framework can use distributed learning to solve actual large-scale network topology scenarios.

2) RL Training Architecture: We use a framework called ray[20–22] for our reinforcement learning training, which can provide simple primitives for building and running distributed applications. With the help of ray, we can build a distributed framework to accelerate the training process of reinforcement learning. The specific framework is shown in **Fig. 4**. Each rollout worker contains a simulation environment and the latest policy, where each policy exchanges weights with the central SGD learning process at regular intervals to obtain the latest learned policy.

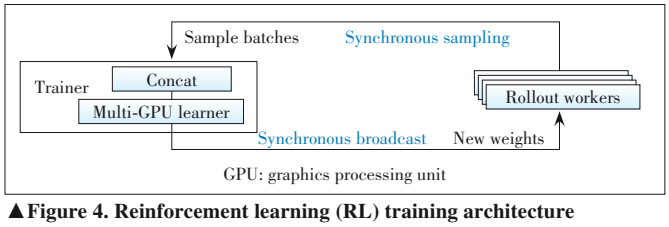
In this distributed reinforcement learning architecture, the central learner runs SGD in a tight loop, while asynchronously extracting sample batches from many participant processes, and also supports multiple GPU learners and experience replay.

4.2 Dynamic Resource Adjustment Among Slices

We first introduce the traditional resource allocation in the slice scenario. In a network that already has some flows, each link in the network has different remaining bandwidths. At this time, we have to start the slicing operation. The common method is to evenly allocate the remaining bandwidth of each link to each slice. This relatively fair initialization strategy is intuitive; however, the resources of slices are unchanged during deployment. Therefore, the flexibility of flow routing on each slice is greatly reduced compared with when it is not sliced.



▲**Figure 3. Interaction between policy and environment**



▲**Figure 4. Reinforcement learning (RL) training architecture**

We now propose the dynamic resource adjustment strategy to increase the flexibility of resource allocation method. Compared with the traditional (static) method, our proposed method dynamically adjusts resources among slices in a real-time manner, aiming at balancing the type of flow deployed in the slice.

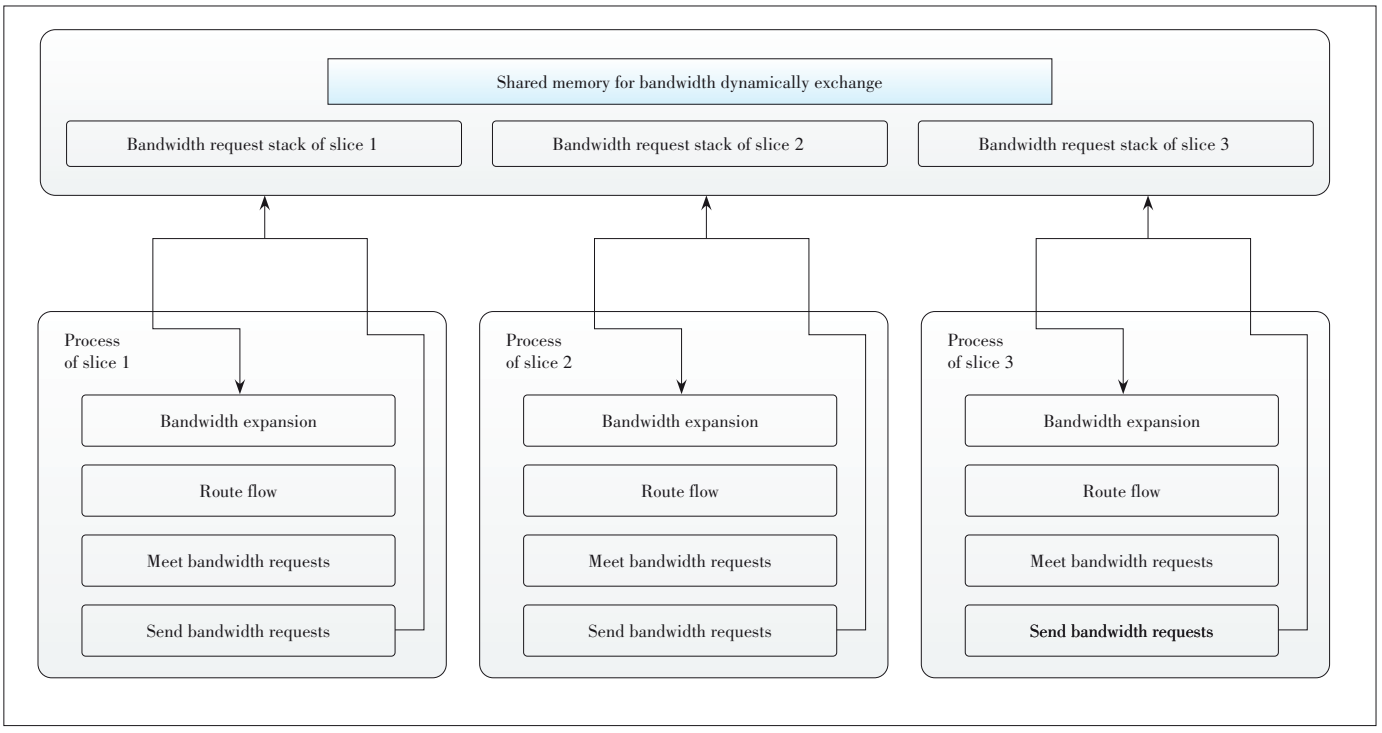
The specific algorithm description can be seen in **Algorithm 1**, and the deployment in each slice can be processed in separate processes. In the beginning, in each slice process, we expand the link bandwidth of the slice in step 3, and this step is to apply resources transferred from other slices to the current slice before the flow deployment. Then in step 4, we calculate the path by the specified routing strategy. In the algorithm, we use the simple strategy of the SP as an example. In fact, it can be any routing strategy. After this, in step 5 we check the bandwidth request from another slice. If the available bandwidth is greater than twice the maximum flow size, we then transfer the required bandwidth and reduce it in the link. At last, in step 6, we send requests to other slices to increase the bandwidth resources of each link in P_n.

Algorithm 1 Dynamic resource adjustment

Input: F-flows, S-slices.
Output: Resource adjustment
1: **for all** s_k in S **do**:
2: **for** f_n in F^k **do**:
3: Expand link bandwidth in s_k
4: $P_n \leftarrow \mathrm{SP}f_n$
5: Meet the bandwidth transfer requirements of other slices
6: Request bandwidth resources of P_n from other slices
7: **end for**
8: **end for**

The specific structure of Algorithm 1 can be seen in **Fig. 5**, where we use a shared memory stack to asynchronously exchange bandwidth resources between different slice processes. This is an example of three slices, which can be adjusted to any number of slices. Each slice first obtains the bandwidth resources transferred by other slices, and then deploys the flow. After that, the slice determines whether the current bandwidth resources can meet the requests of other slices, and if so, transfers the bandwidth to the other slices. The slice finally sends bandwidth transfer requests to other slices according to the path taken by the current deployment flow.

In the flow classification process of RL training, the environment that RL agent learns from also contains a resource adaptation strategy between slices. With the cooperation of the two methods, RL can learn a policy to allocate flows that re-



▲Figure 5. Dynamic resource adjustment

peatedly pass through certain links to the same slice. One slice focuses on transmitting the same type of flows, and other infrequently used links' bandwidth resources are automatically adjusted to other more needed slices.

5 Experiment

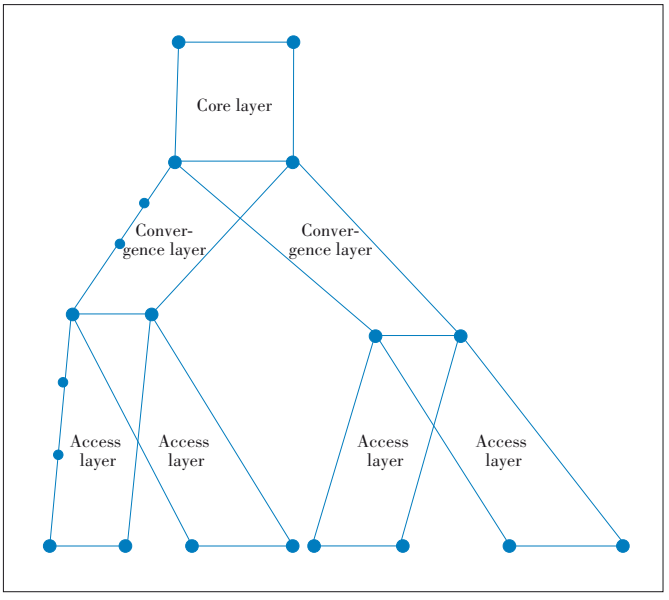
In this section, we conduct some experiments to explain the effectiveness of our algorithm.

5.1 Experiment Setup

We first introduce the information of network topology and flow.

Network topology: We use the ring network similar to a real 5G network as the experimental environment. This ring network consists of three kinds of rings, namely, the core layer, the convergence layer, and the access layer. The bandwidth and delay of links in three kinds of layers are different. The core layer, a ring network composed of the core equipment, has the largest bandwidth of the network, and it is the destination of most services in the network. The convergence layer, the bandwidth of which is smaller than that of the core layer, connects the core layer and the access layer, and aggregates each access layer network. The access layer has the smallest bandwidth of the network, which is composed of users and terminals. An example of the network topology of the ring network is shown in **Fig. 6**. In this experiment, we use a ring network with 6 245 nodes and 8 135 links.

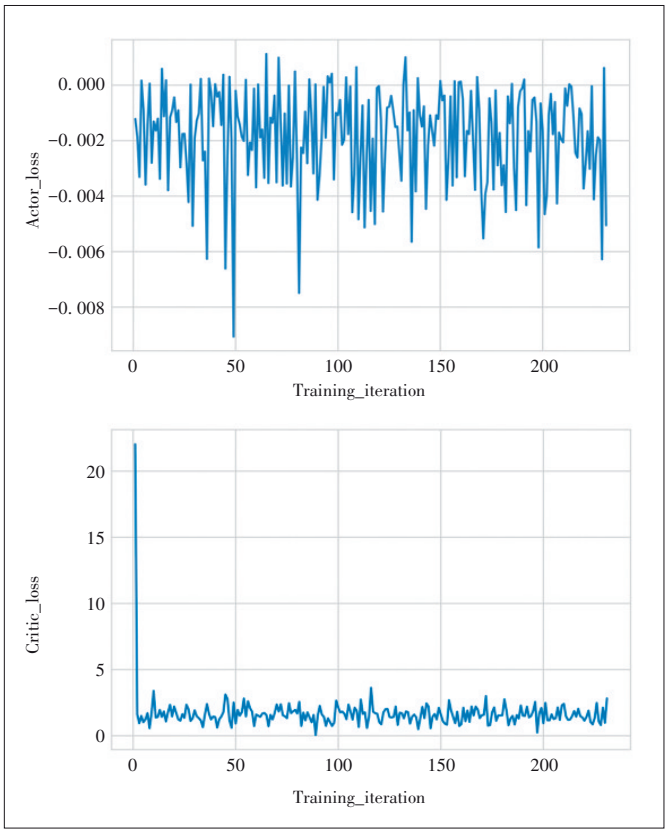
Flows: The source and end nodes of the flow are randomly generated, and the size of the flow is $U(0.1,0) * \max \text{bandwidth}_{\text{access}}$, where $U(0.1, 1)$ is a uniform distribution.



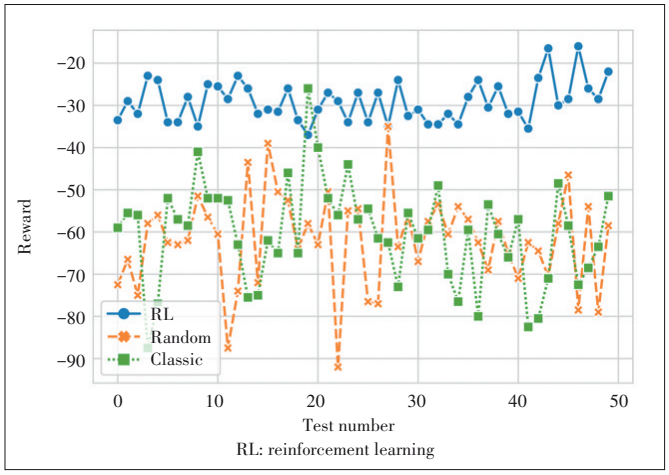
▲Figure 6. An example of network topology of the ring network

5.2 Result of RL Agent

In the training process, thanks to the ray framework and careful hyperparameter adjustment, our algorithm can tend to converge within five iterations. The specific training loss is shown in **Fig. 7**. The critic loss can reflect the accuracy of RL



▲Figure 7. Loss during training is based on a 2080 Ti graphics processing unit (GPU) and 120 sampling processes (2 Xeon CPUs)



▲Figure 8. Total reward of different methods

agent's estimation of observation, and the policy loss is close to zero, which indicates that the strategy is close to the optimal strategy.

In RL test, we use accumulated rewards to show whether the classification effect is good or not, and we also convert other classification into the same measurement indicators. From **Fig. 8,** we can see that the cumulative reward of the converged RL scheme fluctuates around −30 (the larger the better). The cumulative reward of the classic pooling method and random choice method used for comparison is around −70, so we can see that RL can effectively choose the correct slice for the flow.

The above is a unified measurement in the reinforcement learning environment, and what we actually care about is whether the success rate of the deployment can be improved with the help of RL, which relates to whether our reward design is reasonable.

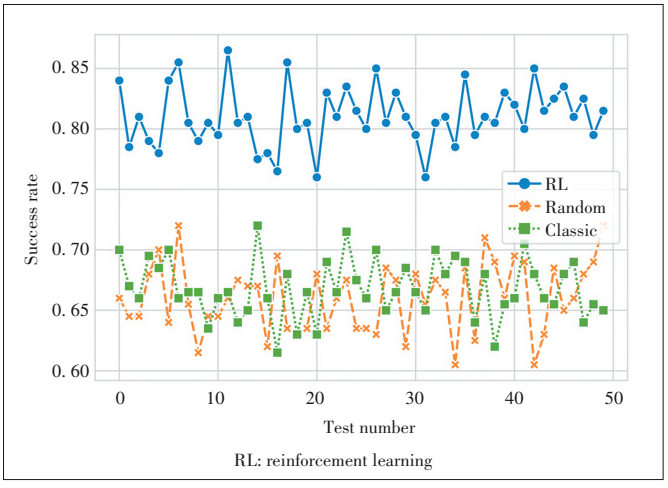
So we test the success rate of using the RL agent for flow classification, as well as the success rate of the classic method and random method as a comparison, and the results are shown in **Fig. 9**. We can see that the strategy learned using RL is significantly better than the classic method while maintaining the same trend as in Fig. 8, which shows that the reward we designed for RL is reasonable.

5.3 Result of Dynamic Resource Adjustment Strategy

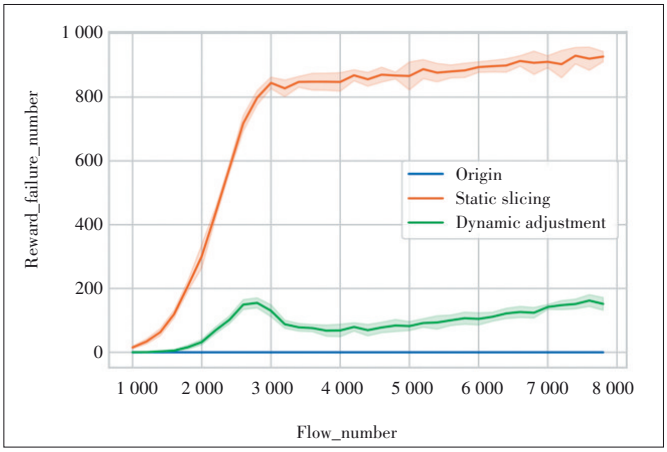
In the topology set above, we continuously increase the number of flows to compare the effects of different strategies by comparing the number of failed deployments. The first is the non-slicing case. After slicing, due to the decrease of routing flexibility, the failure rate will inevitably be greater than the case without slicing, so we regard the non-slicing case as a benchmark. Then we test the static slicing strategy and compare the dynamically adapted slicing strategy we proposed. The result can be seen in **Fig. 10**.

From Fig. 10, we can see that the dynamic adjustment strategy can provide a failure rate that is almost close to that of an unsliced case, and the effect is much better than the static solution.

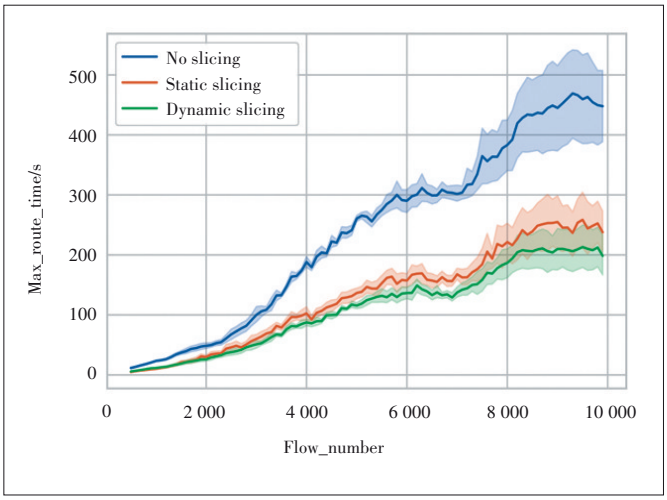
The purpose of slicing is to save the calculation time for parallel calculation, but the dynamic adjustment strategy will obviously increase some time consumption, so we test the average time of routing each flow, and the results are shown in **Fig. 11**. We can get that dynamic action almost bring no additional time consumption.



▲Figure 9. Success Rate of different methods



▲Figure 10. Relative failure number of dynamic resource adaptation vs. static slicing



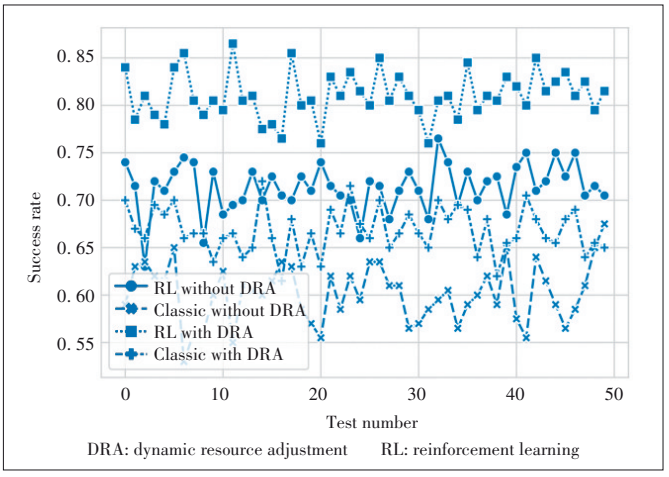
▲Figure 11. Comparison of calculation time of different methods

5.4 Comparison of Effectiveness of Combined Methods

Although we have confirmed the effectiveness of the dynamic resource adjustment strategy, we still have uncertainty about the combined effect of the two algorithms. Therefore, we conduct the following experiments to compare the performance of the RL agent flow classification and the classic method with and without the dynamic resource adjustment strategy. The result is shown in **Fig. 12**. It can be seen that no matter which method is used, the dynamic adjustment strategy can indeed further improve the success rate of flow deployment.

6 Conclusions

To solve the time-consuming problem of path calculation, a slicing operation is used to separate resources so that parallel path calculation can be performed to shorten the time. At the same time, after slicing, although the calculation time is shortened, the more serious problem occurs, which is the decline in the success rate of flow deployment. In response to this problem, we propose an efficient network slicing with dynamic resource allocation algorithm to let the success rate of flow deployment close to the level when not sliced. It combines the flow classification of reinforcement learning and resource adaptation between slices. The dynamic resource adaptive strategy between slices enables slice resources to be exchanged during flow deployment and gradually adapt to the characteristics of the flows to be deployed in each slice. In this way, the parallel calculation path can be shortened without sacrificing the success rate. Besides, we have also tested our method on a large-scale actual network structure, and the effect exceeds the previous classic methods.



▲Figure 12. Comparison of the effectiveness of combined methods

References

[1] JEFFREY G A, BUZZI S, CHOI W, et al. What will 5G be? [J]. IEEE Journal on selected areas in communications, 2014, 32(6): 1065. DOI: 10.1109/JSAC.2014.2328098
[2] EINSIEDLER H J, GAVRAS A, SELLSTEDT P, et al. System design for 5G converged networks [C]//2015 European Conference on Networks and Communications. Paris, France: IEEE, 2015: 391 – 396. DOI:10.1109/EuCNC.2015.7194105
[3] HAWILO H, SHAMI A, MIRAHMADI M, et al. NFV: state of the art, challenges, and implementation in next generation mobile networks [J]. IEEE network, 2014, 28(6): 18 – 26. DOI:10.1109/MNET.2014.6963800
[4] KARAKUS M, DURRESI A. Quality of service (QoS) in software defined networking (SDN): a survey [J]. Journal of network and computer applications, 2017, 80: 200 – 218. DOI: 10.1016/j.jnca.2016.12.019
[5] WALLNER R, CANNISTRA R. An SDN approach: quality of service using big switch's floodlight open-source controller [J]. Proceedings of the Asia-Pacific advanced network, 2013, 35: 14. DOI:10.7125/APAN.35.2
[6] XU C, CHEN B, QIAN H. Quality of service guaranteed resource management dynamically in software defined network [J]. Journal of communications, 2015: 843 – 850. DOI:10.12720/jcm.10.11.843-850
[7] XU C, GAMAGE S, LU H. vTurbo: accelerating virtual machine I/O processing

using designated turbo-sliced core [C]//2013 USENIX Annual Technical Conference. San Jose, USA: USENIX, 2013:243 – 254
[8] SCANO D, VALCARENGHI L, KONDEPU K, et al. Network slicing in SDN networks [C]//2020 22nd International Conference on Transparent Optical Networks (ICTON). Bari, Italy: IEEE, 2020: 1 – 4. DOI: 10.1109/ICTON51198.2020.9203184
[9] ZHU K, HOSSAIN E. Virtualization of 5G cellular networks as a hierarchical combinatorial auction [J]. IEEE transactions on mobile computing, 2016, 15 (10): 2640 – 2654. DOI:10.1109/TMC.2015.2506578
[10] KATSALIS K, NIKAEIN N, SCHILLER E, et al. Network slices toward 5G communications: Slicing the LTE network [J]. IEEE communications magazine, 2017, 55(8): 146 – 154. DOI:10.1109/MCOM.2017.1600936
[11] BARI M F, BOUTABA R, ESTEVES R, et al. Data center network virtualization: A survey [J]. IEEE communications surveys & tutorials, 2013, 15(2): 909 – 928.DOI:10.1109/SURV.2012.090512.00043
[12] ZHANG Q X, LIU F M, ZENG C B. Adaptive interference-aware VNF placement for service-customized 5G network slices [C]//IEEE INFOCOM 2019 IEEE Conference on Computer Communications. Paris, France: IEEE, 2019: 2449 – 2457. DOI:10.1109/INFOCOM.2019.8737660
[13] CHEN J J, TSAI M H, ZHAO L Q, et al. Realizing dynamic network slice resource management based on SDN networks [C]//2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA). Tainan, Taiwan, China: IEEE, 2019: 120 – 125. DOI:10.1109/ICEA.2019.8858288
[14] PARSAEEFARD S, JUMBA V, DERAKHSHANI M, et al. Joint resource provisioning and admission control in wireless virtualized networks [C]//2015 IEEE Wireless Communications and Networking Conference. New Orleans, USA: IEEE, 2015: 2020 – 2025. DOI:10.1109/WCNC.2015.7127778
[15] MONEMI M, RASTI M, HOSSAIN E. Low-complexity SINR feasibility checking and joint power and admission control in prioritized multitier cellular networks [J]. IEEE transactions on wireless communications, 2016, 15(3): 2421 – 2434. DOI:10.1109/TWC.2015.2504084
[16] SUTTON R S, BARTO A G. Reinforcement learning: an introduction [M]. Cambridge, UK: MIT Press, 2018
[17] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms [EB/OL]. (2017-08-28) [2021-12-09]. https://arxiv.org/abs/1707.06347
[18] ESPEHOLT L, SOYER H, MUNOS R, et al. IMPALA: scalable distributed deep-RL with importance weighted actor-learner architectures [EB/OL]. (2018-06-28) [2021-12-09]. https://arxiv.org/abs/1802.01561?context=cs
[19] STOOKE A, ABBEEL P. Accelerated methods for deep reinforcement learning [EB/OL]. (2019-12-09) [2021-01-18]. https://arxiv.org/abs/1803.02811
[20] MORITZ P, NISHIHARA R, WANG S, et al. Ray: A distributed framework for emerging AI applications [C]//13th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA: OSDI, 2018: 561 – 577
[21] LIANG E, LIAW R, NISHIHARA R, et al. RLLIB: abstractions for distributed reinforcement learning [EB/OL]. (2018-07-29) [2021-12-09]. https://arxiv.org/abs/1712.09381
[22] LIAW R, LIANG E, NISHIHARA R, et al. Tune: a research platform for distributed model selection and training [EB/OL]. (2018-07-13) [2021-12-09]. https://arxiv.org/abs/1807.05118

Biographies

JI Hong (hong_ji@seu.edu.cn) received the B.S. degree from School of Economics and Management from Xidian University, China in 2018. He is currently a master student at School of Cyber Science and Engineering, Southeast University, China. His research interests lie in network optimization and intelligent decision making.

ZHANG Tianxiang received the B.S. degree from Nanjing University of Aeronautics and Astronautics, China. He is currently an engineer with ZTE Corporation. His research interests include traffic scheduling and graph neural networks.

ZHANG Kai is currently a master student at School of Cyber Science and Engineering, Southeast University, China. His research interests include graph neural networks and resource allocation and reinforcement learning.

WANG Wanyuan is an assistant professor with the School of Computer Science and Engineering, Southeast University, China. He received his Ph. D. degree in computer science from Southeast University in 2016. He has published several articles in refereed journals and conference proceedings, such as the *IEEE Transactions on Mobile Computing*, *IEEE Journal on Selected Areas in Communications*, *IEEE Transactions on Cybernetics*, *AAAI*, and *AAMAS*. He won the best student paper award from ICTAI14. His main research interests include artificial intelligence, multiagent systems, and game theory.

WU Weiwei is a professor in the School of Computer Science and Engineering, Southeast University, China. He received his B.Sc. degree fron South China University of Technology, China and the Ph.D. degree from City University of Hong Kong (CityU), China and University of Science and Technology of China (USTC) in 2011, and went to Nanyang Technological University, Singapore for post-doctorial research in 2012. He has published over 50 peer-reviewed papers in international conferences/journals, and serves as TPCs and reviewers for several top international journals and conferences. His research interests include optimizations and algorithm analysis, wireless communications, crowdsourcing, cloud computing, reinforcement learning, game theory and network economics.