AI-Enabled Automated and Closed-Loop Optimization Algorithms for Delay-Aware Network

Da Xiao*, Wei Ni†, J. Andrew Zhang*, Renping Liu*, Shuo Chen+ and Yiwen Qu*
*The University of Technology Sydney, Global Big Data Technologies Centre (GBDTC), Australia
†Data61, Commonwealth Scientific and Industrial Research Organization (CSIRO), Sydney, Australia
+School of Computer Science and Engineering, Nanyang Technological University, Singapore
Email:{Andrew.Zhang; renping.liu}@uts.edu.au; {Da.Xiao; Yiwen.Qu-1}@student.uts.edu.au;
wei.ni@data61.csiro.au; schen@ntu.edu.sg

***Abstract*—Network slicing is one of the core techniques of the current 5G networks. To accommodate as many network slices as possible with limited hardware resources, service providers need to avoid over-provisioning of resources. In this paper, we first propose a Deep Q-Network (DQN) based network slicing algorithm to maximize the acceptance ratio and ensure prior placement of higher-priority requests for Ultra-Reliable Low-Latency Communication (URLLC) services. Specifically, we model the network slicing as a Markov Decision Process (MDP), where we consider Virtual Network Function (VNF) placements to be the actions of the MDP, and define a reward function based on service priority. For every service request, we use the DQN to choose an MDP action for performing the VNF placement. The placement results in an MDP reward that we can use to train the DQN. Once trained, the DQN approximates the optimal solution of the MDP. Considering the over-provisioning of resources, we then propose a Binary Search Assisted Transfer Learning algorithm (BSATL), in which the available hardware resources are scaled down/up and the knowledge learned from the source task is transferred to the target task in each iteration, to achieve automated and closed-loop optimization for the ever changing infrastructure, a scenario of 6G Event Defined uRLLC (EDuRLLC). Numerical evaluations show that our proposed scheme can significantly improve cost-utility while maintaining the optimal acceptance ratio.**

***Index Terms*—network slicing, over-provisioning of resources, VNF placement, 6G, EDuRLLC, transfer learning, Deep Q-Network.**

I. Introduction

The current-generation mobile network (5G) imposes stringent Quality-of-Service (QoS) requirements on payload traffic than its predecessor, the 4G system [1], for supporting Ultra-Reliable Low-Latency Communication (URLLC) applications such as remote surgery and self-driving vehicles [2]. Therefore, efficient and automatic placement of network services is one of the most important technologies for meeting such QoS requirements. Furthermore, within URLLC, traffic from different users may have different latency requirements. Hence, we need to create several network services, also known as sub-slices, to meet diverse requirements.

In 6G era, the network will be required to support an advanced version of URLLC service: Event Defined uRLLC (EDuRLLC). Unlike 5G network, in which redundant resources are reserved to offset uncertainties in URLLC application scenarios, 6G will need to support URLLC in emergency events with spatially and temporally changing device densities, traffic patterns, and spectrum as well as infrastructure availability. Therefore, the 6G architecture should be smart enough to learn the network dynamics and automatically re-orchestrate network services.

Many solutions have already been proposed for the placement of network services [3]. Some techniques were proposed to formulate and solve optimization problems [4]–[6]. They typically assume some theoretical traffic and service models. When these models do not match the practical ones, their performance might largely degrade. Some other authors proposed heuristic methods that are good for stationary systems while could have degraded performance for dynamic systems [7], [8]. Recently, techniques based on deep learning were developed [9]–[11]. For the latency issue, [9] is the first paper that proposes to extract latency in real time, making it very attractive. Adopting the real-time model can greatly improve the accuracy of the model, however, there exist several limits in [9]. First, they used Virtual Network Function Forwarding Graphs (VNF-FGs) requested by clients to symbolize traffic in the deep reinforcement learning state. It is important to take VNF-FGs into the state, but it is equally important to include the incoming traffic of each service, an indispensable feature of the environment. This is because for a given VNF-FG configuration, different traffic rates may result in different real latencies. Second, the authors assumed that the requested resources of VNFs are normalized and distributed uniformly, which could be impractical in some real cases. For instance, in OpenStack (an open source cloud computing infrastructure software project) supported cloud environment, for resources like VCPU, RAM and Disk, there are several 'flavors' [12] to choose from. Therefore, these required resources are discrete random variables rather than continuous random variables.

Of all the existing works, there are two main open issues. First, only a few of them focus on sub-slices issue, let alone latency-sensitive sub-slices. Second, most works deploy network slices in stable systems. However, in some cases where environmental characteristics such as infrastructure availability, service requests, and traffic patterns are prone to change, an automated and closed-loop optimization algorithm is needed to help re-orchestrate network slices.

In this paper, to tackle the sub-slice issue, we propose a

Deep Q-Network (DQN) based network slicing algorithm for deploying several URLLC sub-slices with different latency requirements. Specifically, we model the VNF placement for all requests as a Markov Decision Process (MDP) and represent the MDP action space as the possible VNF placements of a single request. We further prioritize those requests based on their latency requirements and define the reward function of the MDP based on priority. For every incoming request, the DQN chooses an MDP action to determine the VNF placement. In response to the VNF placement, an MDP reward is returned from the environment to train our DQN. Once trained, the DQN approximates the optimal solution of the MDP that maximizes the acceptance ratio and ensures prior placement of higher-priority services.

Then, to cope with the over-provisioning of resources, we propose a Binary Search Assisted Transfer Learning algorithm (BSATL), in which the available hardware resources are scaled down/up and the knowledge learned from the source task is transferred to the target task in each iteration, to achieve automated and closed-loop optimization for a 6G EDuRLLC scenario.

The main contributions of this paper can be summarized as follows:

1) We formulate the VNF placement problem as an MDP with appropriate state and action, by including the real incoming traffic into state, rather than by using VNF capacity (CPU, memory, ...) to symbolize incoming traffic as in [9]. This is important because, in some cases, VNF capacity could not accurately reflect real network traffic, which is critical to latency-sensitive services.
2) We prioritize service requests based on latency requirements (the lower latency threshold a service requests, the higher priority it gets) and propose a DQN framework to maximize the service acceptance ratio while ensuring prior placement of higher-priority requests.
3) We design an automated and closed-loop optimization algorithm (BSATL) for a 6G EDuRLLC scenario.

II. System Model and Problem Formulation

A. System Architecture

In the NFV architecture, Network Slice Management Function (NSMF) receives slice requests. Accordingly, it interfaces with Management and Orchestration (MANO) to plan for VNFs placement. In our work, the DQN-agent, which takes the role of MANO, is responsible for choosing location for VNFs. As for the infrastructure, we adopt a widely used nonblocking fat-tree topology, depicted in Fig 1. Let $\mathbb{D}$ denote the set of edge clouds and $\mathbb{E}_i$ denote the set of servers in the i^{th} edge cloud. For simplification, we assume that the number of servers in all edge clouds are the same. Hence, the number of edge clouds is $|\mathbb{D}|$ and the number of servers in each edge cloud is $|\mathbb{E}|$. Without loss of generality, we consider the following example. When DQN-agent handles a service request VNF1→VNF2→VNF3, we suppose that it chooses server1 to place VNF1 and VNF2, and server5 to place VNF3. After the incoming traffic travels

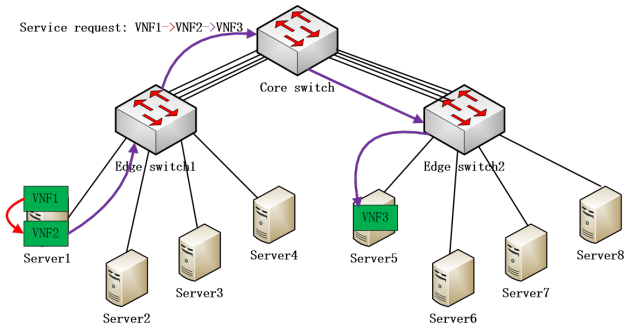

Fig. 1: Network infrastructure considered in this paper.

through VNF1→VNF2→edge switch1→core switch→edge switch2→VNF3, the agent extracts the real latency to see whether the corresponding latency requirement is satisfied.

B. Characteristics of Service Requests

Let K represent the number of requested services. Also, the number of chained VNFs for the URLLC service is assumed to be V. Unlike the authors in [9], who assume that all services have the same latency requirement, we assume that, within the URLLC category, each service has its distinctive latency requirement. Hence, we represent the latency threshold of the i^{th} service with L_i.

C. Problem Formulation

In this part, we formulate the objective as an optimization problem. Our purpose is to maximize the acceptance ratio while ensuring prior placement of higher-priority services, under the infrastructure resource constraints. Thus, the optimization problem can be written as

$$(P1): \max_{\boldsymbol{A}} C(\boldsymbol{A}) = \sum_{i=1}^{K} \rho_i \times f(L_i - l_i)$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{K}\sum_{j=1}^{V} x_{i,j}^{n} \times c_{i,j} \le C_{\text{tot}}, \ n = 1,2,...,|\mathbb{D}| \times |\mathbb{E}| & \text{(1a)} \\ \sum_{i=1}^{K}\sum_{j=1}^{V} x_{i,j}^{n} \times w_{i,j} \le W_{\text{tot}}, \ n = 1,2,...,|\mathbb{D}| \times |\mathbb{E}|. & \text{(1b)} \end{cases}$$

In the objective function above,

$$f(x) = \begin{cases} -1 & x < 0 \\ 1 & x \ge 0, \end{cases} \tag{2}$$

$$\rho_i = \frac{\frac{1}{L_i}}{\sum_{i=1}^{K} \frac{1}{L_i}}, \ i = 1,2,...,K, \tag{3}$$

ρ_i denotes the priority of service i, element $a_{i,j}$ in matrix $\boldsymbol{A}$ indicates on which server is the j^{th} VNF of the i^{th} service located, and L_i and l_i respectively symbolize the latency requirement and the real latency of the i^{th} service.

In the constraints above, the binary variable $x_{i,j}^{n}$ indicates whether the j^{th} VNF of the i^{th} request nests on the n^{th}

server, $c_{i,j}$ and $w_{i,j}$ denote the required CPU and bandwidth resources of the j^{th} VNF of the i^{th} request, and C_{tot} and W_{tot} denote the CPU and bandwidth capacity of any server.

The constraints need to be modified if the infrastructure availability changes. Learning based techniques can be utilized to learn such network dynamics and solve the problem. The goal of the learning technique is to learn a policy that determines what action to take in each environment state. In the following, we introduce a model based on DQN for VNF placement problem regarding latency requirement.

III. Our Proposed VNF Placement Scheme

We model the VNF placement for all requests as an MDP, which can be resolved by a DQN framework, and represent the MDP action space as the possible VNF placements of a single request. A state can be defined as a vector of the available resources provided by the infrastructure and the characteristics of a service. Once the VNF placement action of the first service is determined, the state is updated. Then the action for the second service will be determined, so on and so forth. Since our aim is to maximize the acceptance ratio while ensuring prior placement of higher-priority services, we prioritize those requests based on their latency requirements and define the reward function of the MDP based on service priority. The details of the state, action, and reward function in the context of our problem will be provided in Sections III-B, III-C and III-D.

We first propose a DQN framework with the aim of maximizing the acceptance ratio and ensuring prior placement of higher-priority services. Then, to resolve the over-provisioning of resources, we propose the BSATL algorithm, in which the minimum number of online servers for maintaining the maximum acceptance ratio is iteratively searched for, to achieve automated and closed-loop optimization for the 6G Event Defined uRLLC (EDuRLLC) scenario. The details are presented below.

A. Overview of the Proposed DQN

A general overview of the DQN used in our scheme is shown in Fig. 2. The state, indicated by a vector, is fed into the neural network, which outputs a vector of Q-values, with each indicating the expected discounted cumulative reward of a corresponding action (VNF placement). At time step t, the Q-value of taking an action a_t in state s_t based on policy π is given by

$$Q^{\pi}(s_t, a_t) = E(\sum_{i=t}^{K} \gamma^{(i-t)} R(s_i, a_i) | s_t, a_t). \tag{4}$$

The objective of the agent is to learn a policy that maximizes the expected return $Q^{\pi}(s_t, a_t)$. At the beginning of the training, the weights of the evaluation neural network are random and thus the policy is poor. Given a state, the maximum Q-value may not account for the optimal policy. Hence, we train the framework episode by episode. When an episode starts, we re-initialize the state of the environment and feed the agent K requests, which will be handled one by one in K time

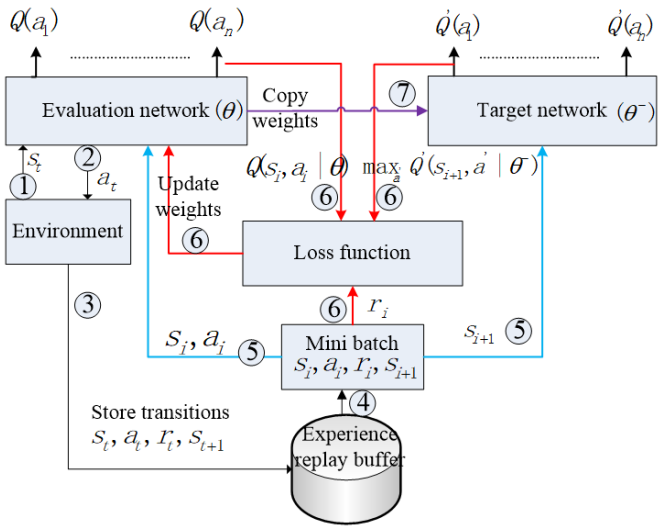

Fig. 2: Flowchart of the Deep Q learning network.

steps. In each time step, the agent serves a request and updates the neural networks. When the agent finishes the placement of K requests, this episode ends and the next one starts. The loop ends as the weights of the neural networks converge. Then, given a state, the agent is able to find the best action by taking the largest Q-value from the output vector. The workflow for each time step can be summarized as follows:

1) Step 1: The agent observes the state (s_t) of the environment.
2) Step 2: The agent performs an action (a_t), i.e., the VNF placement, randomly with probability ϵ or according to the evaluation network with probability $1-\epsilon$, and thus gets a reward (r_t).
3) Step 3: The experience tuple (s_t, a_t, r_t, s_{t+1}) is stored into an experience replay buffer.
4) Step 4: To train the DQN framework, a mini-batch of N tuples are uniformly sampled from the experience replay buffer.
5) Step 5: For tuple i, s_i and a_i are fed into the evaluation network (θ), while s_{i+1} is fed into the target network (θ^-).
6) Step 6: The weights of the evaluation network are updated by minimizing a loss function:

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q(s_i, a_i|\theta))^2, \tag{5}$$

where

$$y_i = r_i + \gamma max_{a'} Q'(s_{i+1}, a'|\theta^-). \tag{6}$$

7) Step 7: The weights of the target network are updated by copying the weights of the evaluation network every C time steps.

B. State

We define the state as a vector of available resources provided by the infrastructure and the characteristics of a

service request. As a result, the state set of the environment can be written as:

$$\boldsymbol{S} = \{c_n, w_n, c_{i,j}, w_{i,j}, L_i, \lambda_i\},$$
$$n = 1, 2, ..., |\mathbb{D}| * |\mathbb{E}|, j = 1, 2, ..., V, i \in \{1, 2, ..., K\}, \quad (7)$$

where c_n and w_n respectively represent the remaining CPU and bandwidth resources of the n^{th} server, $c_{i,j}$ and $w_{i,j}$, which can be selected from VM 'flavors', denote the required CPU and bandwidth resources of the j^{th} VNF of the i^{th} service, and L_i and λ_i respectively denote the latency requirement and the incoming traffic of the i^{th} service.

C. Action

We define the action as the possible placement for an incoming service i:

$$\boldsymbol{A} = \{a_{i,j}\}, j = 1, 2, ..., V, i \in \{1, 2, ..., K\}, \quad (8)$$

where $a_{i,j}$ indicates on which server does the j^{th} VNF of the i^{th} service nest.

D. Reward

We leverage a simulation tool CloudSimSDN-NFV [13] to obtain the real latency of an embedded service. If the service's latency requirement is not satisfied, the agent will receive a penalty. Otherwise, it will be granted a reward. For the i^{th} service, the reward function regarding delay is defined as

$$R_{\text{delay}}(i) = \begin{cases} 1 & l_i \leq L_i \\ -1 & l_i > L_i, \end{cases} \quad (9)$$

where l_i and L_i denote the real latency and the latency requirement of the i^{th} service.

We suppose that the j^{th} VNF is planned to be placed on the n^{th} server. If the remaining bandwidth or CPU of server n is insufficient to accommodate VNF j, then the placement for this VNF is a failure. When DQN-agent performs an action for service i, the failure placement for any VNF can lead to a penalty for this action. Therefore, the reward function regarding resources constraint is defined as

$$\mathrm{R}_{\text{res}}(i) = \begin{cases} -1 \\ if\ \exists j \in [1, V],\ c_{i,j} > c_n \text{ or } w_{i,j} > w_n \\ 0 \text{ otherwise}, \end{cases} \quad (10)$$

where $c_{i,j}$ and $w_{i,j}$ denote the required CPU and bandwidth resources of the j^{th} VNF of the i^{th} service, and c_n and w_n respectively represent the remaining CPU and bandwidth resources of the n^{th} server.

We place VNFs for all services such that a global utility function expressed below is maximized.

$$U = \max \sum_{i=1}^{K} (\eta_1 \times \rho_i \times R_{\text{delay}}(i) + \eta_2 \times R_{\text{res}}(i)), \quad (11)$$

where η_1 and η_2 are the weights of two reward functions and ρ_i indicates the priority of service i. For any VNF placement, satisfying the resources constraint is the prerequisite for being measured the delay. Thus, η_2 is greater than η_1.

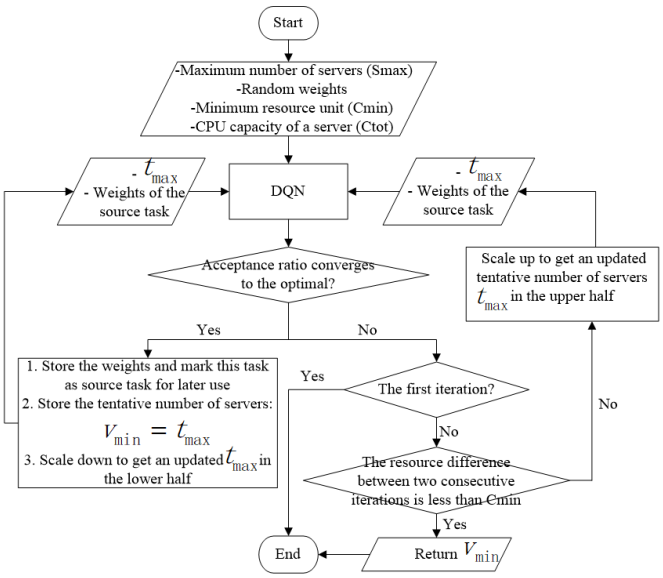

Fig. 3: Workflow of the BSATL algorithm.

E. The Binary Search Assisted Transfer Learning Algorithm

Our BSATL algorithm is summarized in Algorithm 1. The main workflow is shown in Fig 3. The algorithm starts from a DQN learning process in an initial environment with the maximum number of servers (S_{max}). S_{max} and 0 are considered the two ends of the binary search (BS). A new iteration of transfer learning is triggered once the BS creates a new environment, indicated by the tentative maximum number of servers (t_{max}). In each iteration, the source task and target task [14] share the same action space and reward function, and the only difference between the two tasks is the DQN state. Hence, it is reasonable to reuse the neural network model developed in the source task as the starting point of the target task. When the resource gap between the most recently verified environment and the current tentative environment is less than the minimum resource unit (C_{min}), the loop ends and the minimum number of servers (v_{min}) for maintaining the maximum acceptance ratio is returned.

IV. Simulation Results

In this section, we evaluate the performance of the proposed DQN algorithm regarding acceptance ratio in phase one and the cost-utility of the proposed BSATL algorithm in phase two. We use the CloudSimSDN-NFV as our simulation framework. As for the infrastructure (Fig. 1), the bandwidth is set to 1 Mbps for each link and the MIPS of each server is 1000. In our experiment, we assume that there are five categories of latency-sensitive services, each of which consists of a chain of three VNFs and has a distinctive latency threshold. In terms of priority, flow1>flow2>flow3>flow4>flow5. For security purposes, we assume that they are isolated networks and thus they do not share VNFs.

The DQN network is set up with the following parameters. Adam is adopted to learn the neural network parameters. The learning rate is 0.005 and the discount factor is 0.99. The weights of the target network are updated every 100 episodes.

Algorithm 1 BSATL Algorithm

Input: $S_{\max}, C_{\text{tot}}, C_{\min}$
Output: $v_{\min}$

$t_{\min} = 0, t_{\max} = S_{\max}, v_{\min} = 0$
$i = 1$
while $(v_{\min} - t_{\max})C_{\text{tot}} > C_{\min}$ or $i = 1$ **do**
  Substituting $t_{\max}$ into DQN state and training or
  retraining the network to see whether the DQN converges
  if the acceptance ratio converges to the optimal **then**
    $v_{\min} = t_{\max}, t_{\max} = \frac{t_{\max}+t_{\min}}{2}$
  else
    $t_{\min} = t_{\max}, t_{\max} = \frac{t_{\min}+v_{\min}}{2}$
  end if
  $i = i + 1$
end while
return $v_{\min}$

The batch size is 64. We set two hidden layers for the fully connected network, the number of units is 4096 for the first hidden layer and 1024 for the second hidden layer. We adopt the Rectified Linear Unit (ReLU) activation for hidden layers.

Our proposed algorithms both consist of an offline training phase and an online testing phase. In the training phase, we create service requests according to their traffic and arrival pattern history and continuously train the framework until the weights of neural networks converge.

During the online testing phase, we compare the performance of our proposed algorithms with a state-of-the-art heuristic algorithm – Holu [8] and the typical Best-Fit algorithm. The Holu algorithm, which aims to minimize the number of online Physical Machines (PMs) while meeting end-to-end delay and resource capacities, maps VNFs to PMs using a centrality-based PM ranking strategy. The Best-Fit algorithm deploys VNFs one after another to the smallest free partition which meets the resources requirement of the VNFs.

In phase one, we compare our DQN algorithm with the Holu and the Best-Fit regarding acceptance ratio. From Fig. 4, we can see that, when we restrict the number of online servers to be four, the DQN and the Holu accept four requests while the Best-Fit accepts three. Furthermore, only our DQN algorithm accepts four higher-priority services and discards the lowest priority service. This is because we assign greater reward/penalty values to the success/failure of higher-priority service requests, and the DQN algorithm achieves the maximization of the total discounted cumulative reward at the sacrifice of some instantaneous reward. However, the other two algorithms, which do not consider the priority issue, handle requests based on their arrival sequence. As a result, with the same number of online servers, our DQN algorithm achieves the same acceptance ratio as the Holu. Meanwhile, it ensures prior placement of higher-priority services.

In phase two, to create an over-provisioning of resources scenario, we first turn on all eight servers to pre-train a DQN framework that can accept all five requests. From Fig. 5, we can see that, before episode 10000, DQN-agent

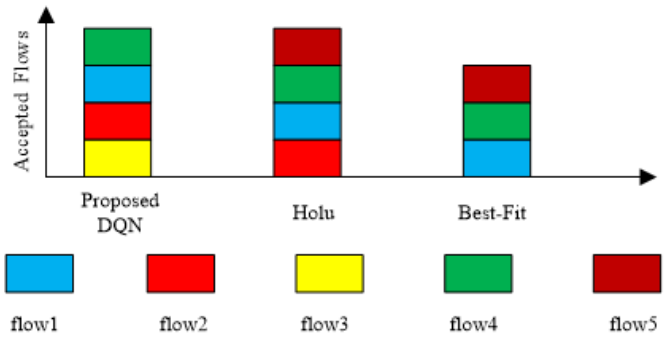

Fig. 4: Accepted services with different priorities.

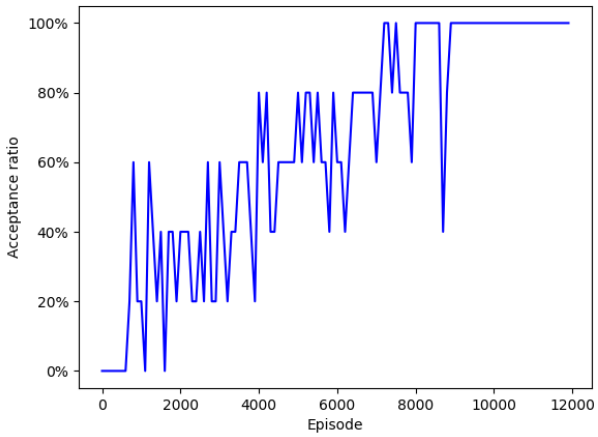

Fig. 5: DQN learning process of the initial task.

explores the environment. As the agent becomes smarter and smarter, the acceptance ratio increases gradually. After we finish the training at episode 10000, our DQN algorithm always achieves the maximum acceptance ratio.

Then, we run BSATL algorithm to see whether the over-provisioning problem can be resolved. From Fig. 6, we can see the transfer learning process during the loop of Algorithm 1. To maintain the maximum acceptance ratio, the minimum number of online servers should be five. Compare to the initial environment with 8 servers, we significantly improve the cost-utility while maintaining the acceptance ratio. Regarding the common characteristics, we can see that, all curves start from a value greater than or equal to 40% and finally converge within 1200 episodes, much faster than the rate of convergence in the initial task. This is because in any new environment, DQN-agent benefits from the knowledge learned in the source task. Regarding the differences, we can notice that, although the agent inherits knowledge from the initial task for both the environment with 6 servers and that with 4 servers, the curve of 6 servers starts from a higher acceptance ratio and converges faster than that of 4 servers. This is because the environment with 6 servers shares more similarities with the initial environment. Another finding is that the acceptance ratio for the environment with 5 servers falls to 80% and then climbs to 100% very soon. The reason

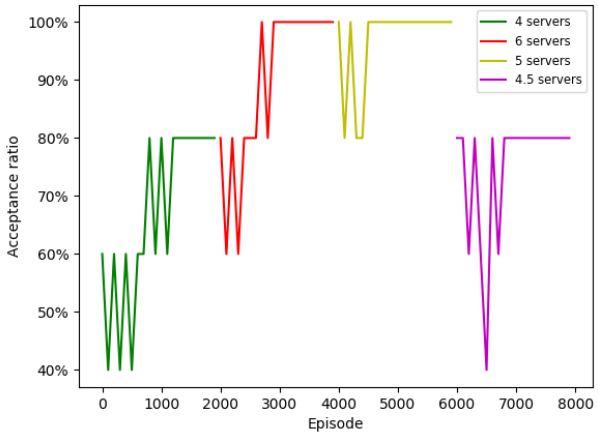

Fig. 6: The binary search assisted transfer learning process.

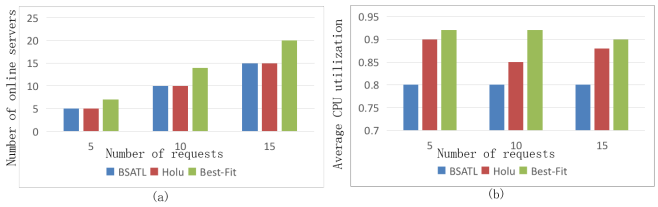

Fig. 7: Cost-utility comparison.

is that, the environment of this task is highly similar to that of its source task, the environment with 6 servers.

Finally, we compare our BSATL algorithm with the Holu and the Best-Fit in terms of cost-utility under different network environments. From Fig. 7(a), we can see that our BSATL and the Holu occupy the same number of servers while the Best-Fit requires more, to accept all incoming requests. The reason is that, the Best-Fit, which places VNFs to the smallest free partition one by one but does not consider the closeness between adjacent VNFs, needs more resources to meet the delay requirement when adjacent VNFs are far away from each other. Unlike the Best-Fit, the Holu takes the closeness between node pairs into consideration when placing VNFs and our BSATL optimizes the VNF placement during the learning phase. From Fig. 7(b), we can see that the average CPU utilization of our BSATL is lower than that of the Holu. That is to say, our BSATL requires fewer CPU resources to accept the same number of requests. The first reason is that our BSATL flexibly reduces the VNFs size by selecting templates from 'flavors' to enhance the cost-utility. In contrast, the Holu fixes the capacity of VNFs, inevitably falling into the over-provisioning mire. The second reason is that the BSATL minimizes the cost from a global perspective, while the Holu only minimizes the cost instantaneously rather than farsightedly. In addition, the Best-Fit algorithm has the highest average CPU utilization and thus consumes the most resources for the aforementioned reason. In conclusion, our BSATL outperforms the other two algorithms in terms of cost-utility.

V. Conclusions

In this paper, we leveraged the DQN algorithm to maximize the acceptance ratio and ensure prior placement of higher-priority services for latency-aware network slices. We considered a multi-edge cloud scenario in which several service requests with different priorities are fed into the DQN-agent, and developed an intelligent policy to place the VNFs. First, considering the objective, we defined our new state, action, reward function for the DQN framework. Using simulations, we showed that DQN-agent is able to maximize the acceptance ratio and ensure prior placement of higher-priority services for latency-aware services with different latency requirements. Then, we proposed an algorithm – BSATL, which achieves automated and closed-loop optimization for a 6G EDuRLLC scenario, to resolve the over-provisioning of resources. Numerical results showed that our proposed scheme can improve cost-utility while maintaining the acceptance ratio, as expected.

References

[1] H. Tullberg *et al.*, "The metis 5g system concept: Meeting the 5g requirements," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 132–139, Dec. 2016.

[2] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "The 5g-enabled tactile internet: Applications, requirements, and architecture," in *2016 IEEE Wireless Communication and Networking Conference Workshops (WCNCW)*, Doha, Qatar, Apr. 2016, pp. 1–6.

[3] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, third quarter 2018.

[4] Q. Ye, W. Zhuang, X. Li, and J. Rao, "End-to-end delay modeling for embedded vnf chains in 5g core networks," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 692–704, Feb. 2019.

[5] A. Baumgartner, V. S. Reddy, and T. Bauschert, "Combined virtual mobile core network function placement and topology optimization with latency bounds," in *2015 Fourth European Workshop on Software Defined Networks*, Bilbao, Spain, Sep.-Oct. 2015, pp. 97–102.

[6] D. B. Oljira, K. Grinnemo, J. Taheri, and A. Brunstrom, "A model for qos-aware vnf placement and provisioning," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Berlin, Germany, Nov. 2017, pp. 1–7.

[7] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[8] A. Varasteh, B. Madiwalar, A. V. Bemten, W. Kellerer, and C. Mas-Machuca, "Holu: Power-aware and delay-constrained vnf placement and chaining," *IEEE Transactions on Network and Service Management*, no. 2, pp. 1524–1539, Jun. 2021.

[9] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, Dec. 2019.

[10] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-end performance-based autonomous vnf placement with adopted reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 534–547, Jun. 2020.

[11] A. Rkhami, Y. Hadjadj-Aoul, and A. Outtagarts, "Learn to improve: A novel deep reinforcement learning approach for beyond 5g network slicing," in *IEEE Consumer Communications & Networking Conference (CCNC)*, Virtual, United States, Jan. 2021, pp. 1–6.

[12] Open Science Data Cloud dev document, "Virtual machines (vms)," 2014. [Online]. Available: https://www.opensciencedatacloud.org/support/instances.html

[13] J. SON, T. He, and R. Buyya, "CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments," *Software: Practice and Experience*, vol. 49, no. 12, pp. 1748–1764, Dec. 2019.

[14] F. Zhuang *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021.