2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

DRL-Deploy:Adaptive Service Function Chains Deployment with Deep Reinforcement Learning

Xiaohui Wei†, Yong Sheng†, Lina Li‡, Changbao Zhou†✉
†College of Computer Science & Technology, *Jilin University,* Changchun, Jilin, China
‡College of Computer Science & Technology, *Changchun University,* Changchun, Jilin, China
weixh@jlu.edu.cn; shengyong19@mails.jlu.edu.cn; liln@ccu.edu.cn; zhoucb18@mails.jlu.edu.cn

***Abstract*—With the development of software-defined networking (SDN) and network function virtualization (NFV), the service function chain (SFC) has become a popular paradigm for carrying and completing network services. In this new computing and networking paradigm, virtual network functions (VNFs) are deployed in software entities/virtual machines through physical device networks in a flexible manner to improve resource utilization and reduce management effectiveness. In this case, it is critical to effectively deploy SFCs within an acceptable time to improve quality of service, while meeting the constraints of the physical network. In this paper, we propose an adaptive deep reinforcement learning based method for the online deployment of SFC requests with different QoS requirements, called DRL-Deploy. DRL-Deploy integrates graph convolutional neural networks to effectively extract physical network features and then adopts a parallel method to improve training efficiency, which can converge to the best state. We compare with existing benchmarks, and extensive experiment results show that DRL-Deploy outperforms all others in terms of acceptance rate and long-term average revenue by 8.6% and 22.9%, respectively, while reducing long-term average cost by 36.4%.**

***Index Terms*—Deep reinforcement learning, network function virtualization, software-defined networks, SFC deployment, graph convolutional network**

I. Introduction

It is ubiquitous to place middleboxes in today's networks to provide customers with various network services. The traditional middlebox is implemented by dedicated hardware equipment, which leads to higher infrastructure and management costs [1]–[3]. Software-defined networking (SDN) is a new network paradigm that can separate the control plane from the data plane and centrally manage it through the SDN controller [4], [5]. At the same time, through network function virtualization (NFV) technology [6], Internet service providers (ISP) can separate network functions from proprietary and dedicated hardware, and can use virtual network functions (VNF) on standard off-the-shelf servers the form instantiates them flexibly.

Based on SDN/NFV technology, the Service Function Chain (SFC) standardized by the Internet Engineering Task Force (IETF) defines a set of ordered or partially ordered VNFs. For SFC requests, traffic needs to be directed to the sequence of the specified VNF that is traversed in a predefined order. With proper programming abstraction, the dynamic deployment of SFC requests can be performed in a way that greatly reduces the capital expenditure (CAPEX) and operational expenditure (OPEX) of the ISP. However, in a physical network, VNFs can be flexibly placed in different network nodes. For example, in Figure 1, There are 5 switching nodes and 4 functional nodes (nodes that can place VNFs) in the physical network. For a newly arrived service function chain request, the source node and destination nodes are A and H respectively. We need to transfer the user data flow the four VNF instances are processed in sequence. Because the physical network is complex, there are many deployment methods for this SFC request (for example, lines of different colors in the figure). Therefore, it is a challenge for ISPs to minimize the total CAPEX/OPEX with the SFC deployment. In addition, due to limited network resources and the order of VNFs in SFC, the optimal deployment of SFC requests should focus on optimizing network performance and reducing resource consumption.

In addition, to obtain the best strategy to solve the SFC deployment problem, most works currently use existing routing algorithms and heuristics based on rules, but this method is usually based on large-scale iterations and calculations, which will lead to higher computational cost and lower time efficiency [10]. At the same time, heuristic methods cannot adapt to continuous changes in the network system. Different network services usually have different QoS constraints [3], [12], [13], such as bandwidth, waiting time, etc. and traffic characteristics [14], [15], such as flow rate, packet size, etc. Therefore, an adaptive online method is needed to automatically deploy SFCs with different requirements.

Reinforcement learning (RL) is used as a learning architecture and uses deep learning techniques [16], [17], such as multilayer neural networks, as automatic feature extractors. It performs well on a series of complex sequential decision problems [18], such as AlphaGo and adaptive video flow. In RL, the agent starts from ignorance, and gradually learns the required behaviors by exploring the external environment. When the agent obtains the network status, it transforms the status into advanced functions and generates action. The environment does this and returns the reward signal to the agent. The reward signal is different from the traditional optimization goal. It does not necessarily maximize the performance of the current network state, but can achieve better performance in the future. The agent then uses the reward signal to dynamically improve the generation of its next action. As

Corresponding email:zhoucb18@mails.jlu.edu.cn

978-0-7381-2646-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00027

the improvement continues, the agent will eventually converge to a strategy that can maximize the cumulative reward for a long time without any clear goal function and optimization goal. After the agent is trained, it only needs one step, that is, the forward calculation of the neural network, to generate an effective strategy through the original state input, which can reduce the computational complexity and solve the SFC deployment problems at the same time.

In this paper, we focus on SFC deployment issues. Given the challenge introduced by SFC, there are two problems that should be solved: (i) how to minimize the total network cost with optimal VNF placement; (ii) How to automatically deploy SFCs with different requirements online and adaptively. To solve the problems raised above, we first introduced the Markov Decision Process (MDP) model to capture dynamic network state transitions. The state of the MDP model is structured like the current network state (for example, CPU, memory, and latency, etc.), the deployment result of the currently running SFC, and arrival requests with different QoS requirements (for example, required CPU, memory, bandwidth, and maximum allowable delay, etc.). Then, we propose a novel SFC deployment algorithm (DRL-Deploy) based on deep reinforcement learning (DRL) technology and graph convolutional neural networks which extract the features of the physical network. The reinforcement learning model of DRL-Deploy is constructed based on Asynchronous Advantage Actor-Critic(A3C) [21], which can realize the deployment of SFC request online. Our contributions are as follows:

- We formulate the online SFC deployment problem as the Markov Decision Process (MDP) model to describe the variation of the network state, where the network changes are automatically and continuously represented as a series of the state transitions.
- We propose an adaptive service function chain deployment scheme DRL-Deploy based on deep reinforcement learning. Particularly, for automatically extracting spatial features in irregular graph topologies, i.e., the physical networks, we use learning agents based on a new type of neural network for graph convolutional networks. At the same time, We use a popular parallel strategy gradient training method to ensure the efficiency and robustness of the sampled training experience, while optimizing the learning agent.
- We conducted extensive simulation experiments on the DRL-Deploy method to verify its performance. Experimental results show that compared with existing approaches, the proposed DRL-Deploy can improve the acceptance ratio, the long-term average revenue and reduce the long-term average cost.

The rest of the paper is organized as follows. In Section II, we introduce related work of SFC deployment problem. In Section III, we provide the system model and formal definition of the problem. The details of DRL-Deploy approach are presented in Section IV. Section IV presents the simulation results. Finally, we conclude our work in Section V.

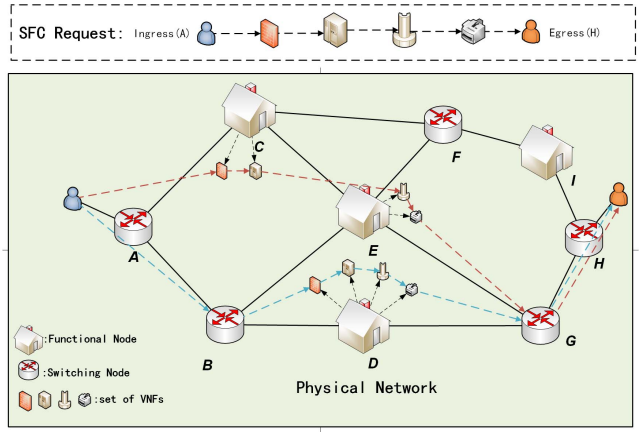

Fig. 1: Example of sfc deployment.

II. Related Work

In recent years, solving the problems of SFC request deployment and routing path selection has become a hot issue in academia, and many solutions have been proposed. Paper [22] and [23] separately studied the VNF layout and routing optimization problems, and formulated them as a Mixed Integer Linear Problem (MILP) model. Considering the resource consumption of links and nodes in a distributed cloud environment, Paper [10] proposed a novel method based on feature decomposition to solve the problem of SFC deployment and linking. Paper [24] and [25] respectively proposed the method of calculating the routing path by connecting the established VNF instances in a predefined order using the graph layering method. Paper [26] Consider the SFC VNF selection and flow control issues, and formulate it as an integer linear programming (ILP) model, whose goal is to maximize network throughput in an environment where SDN/NFV is enabled. To maximize revenue and minimize the cost of infrastructure provider (InP), [20] uses a greedy load balancing method to place each VNF in turn, and then uses the shortest path route to embed each virtual link. The Monte Carlo Tree Search (MCTS) method is used to solve the SFC request deployment problem and maximize the acceptance rate of requests [27]. [28] proposed a method of constructing an SFC request routing path and realizing load balancing by considering multi-resource constraints and flow characteristics. In addition, the commercial SDN controller opendaylight already supports four SFC scheduling algorithms, including random, round-robin, load balancing , and shortest path [29]. However, all of these mentioned solutions are rule-based and cannot realize the intelligent flow control of SFC requests, which usually leads to complicated strategy design and low time efficiency of routing path calculation.

Faced with the huge increase in network traffic and the decline in Internet service provider (ISP) profits, deep reinforcement learning seems to be a feasible method for effective routing path calculation through intelligent traffic management [8], [9] proposed a sequence to sequence neural

network Framework and policy gradient methods to solve VNF placement and minimize the overall power consumption of the system. NFVdeep [19] is an adaptive online deep learning method that can automatically deploy VNF with different QoS requirements. But it does not consider the traffic problem between VNFs. The high computational complexity and low time efficiency of the SFC request routing path calculation method are solved through supervised learning and unsupervised learning, but the training of deep learning model is slow and time-consuming, which is very sensitive to unknown SFC request and network dynamics. The ability to adapt to changes is weak.

III. System Model and Problem Formulation

In this section, we define the physical network model and the SFC request model, and formulate the SFC deployment problem.

A. System Model

1) Physical Network: The physical network is considered as an undirected graph, $G = (\mathcal{V}, \mathcal{L})$. Where $\mathcal{V}$ is a collection of physical nodes composed of switching nodes $\mathcal{V}_S$ and functional nodes $\mathcal{V}_S$, We use $u, v \in \mathcal{V}$ to represent two physical nodes, and $uv \in \mathcal{L}$ represents a physical link. The bandwidth capacity of link $uv \in \mathcal{L}$ is denoted as C_{uv}^{bw}. We denote C_u^{mem} as the memory capacity of $u \in \mathcal{V}$, and C_u^{cpu} indicates the CPU capacity of $u \in \mathcal{V}$. The available resource ratios of link $uv \in \mathcal{L}$ and node $u \in \mathcal{V}$ are symbolized as r_{uv}^{bw},r_u^{mem} and r_u^{cpu}, respectively. All the symbols and variables of this section are listed in Table I.

2) Service Function Chain Requests: In this paper, each SFC request consists of an ingress node, and an egress node and a sequence of VNF requests. A 4-tuple, $\{\mathcal{S}_i,\ \mathcal{T}_i,\ Q_i,\ \Psi_i^{td}\}$ is used to represent SFC request i, where $\mathcal{S}_i$ and $\mathcal{T}_i$ represent the ingress node and egress node, respectively. The set of VNFs requested by SFC request i is denoted by $Q_i = \{Q_i^1, Q_i^2, ..., Q_i^l\}$, $l = |Q_i|$, where $Q_i^1, Q_i^2, ..., Q_i^l$ represent the $1^{st}, 2^{nd}, ..., l^{th}$ VNF requests in Q_i, respectively. Ψ_i^{td} means the maximum tolerated delay of SFC request i. We use $G_i = (\mathcal{V}_i, \mathcal{L}_i)$ to denote the service function graph of SFC request i. Service function graph is a directed graph, and the directions of links satisfy the order constraint of VNF requests. In service function graph, the parameters $\bar{u}, \bar{v} \in \mathcal{V}_i$ represent two VNF request nodes, and $\bar{u}\bar{v} \in \mathcal{L}_i$ is the link connecting node $\bar{u}$ and $\bar{v}$ in G_i. When dealing with SFC request i, d_{uv}, d_u denote the delays suffered in $u \in \mathcal{V}$ and $uv \in \mathcal{L}$, respectively. The maximum tolerable delay of SFC request i is symbolized as Ψ_i^{td}

B. Problem Formulation

In this section, we describe the SFC deployment problem for SDN/NFV.

For SFC request i, the consumptions of bandwidth, memory, and CPU cannot exceed the available resources on links and nodes, respectively, which are ensured as

$$\sum_{\bar{u}\bar{v} \in \mathcal{L}_i} bw_{i,\bar{u}\bar{v}} z_{i,uv}^{\bar{u}\bar{v}} \leq C_{uv}^{bw} r_{uv}^{bw}, \forall uv \in \mathcal{L}, \tag{1}$$

TABLE I: SYMBOLS AND KEY NOTATIONS

Symbols and Variables	Description
Physical Network	
$G = (\mathcal{V}, \mathcal{L})$	Physical network G with the sets of nodes $\mathcal{V}$ and links $\mathcal{L}$, $u, v \in \mathcal{V}$, $uv \in \mathcal{L}$.
$\mathcal{V}_S$, $\mathcal{V}_F$	Switching nodes and functional nodes in the physical network.
C_{uv}^{bw},C_u^{mem},C_u^{cpu}	Capacities of bandwidth, memory and CPU of link $uv \in \mathcal{L}$ and node $u \in \mathcal{V}$.
r_{uv}^{bw},r_u^{mem},r_u^{cpu}	Available ratios of bandwidth, memory and CPU of link $uv \in \mathcal{L}$ and node $u \in \mathcal{V}$.
d_{uv},d_u	Delay on link $uv \in \mathcal{L}$ and node $u \in \mathcal{V}$.
Service Function Graph	
$G_i = (\mathcal{V}_i, \mathcal{L}_i)$	Service function graph G_i with the sets of nodes $\mathcal{V}_i$ and links $\mathcal{L}_i$, $\bar{u}, \bar{v} \in \mathcal{V}_i$, $\bar{u}\bar{v} \in \mathcal{L}_i$.
S_i, T_i, Q_i	The ingress node, egress node and set of necessary VNF requests of SFC request i;$Q_i = \{Q_i^1, Q_i^2, ..., Q_i^l\}, l = \|Q_i\|$.
$cpu_{i,\bar{u}}$, $mem_{i,\bar{u}}$	Cpu and memory required by VNF node $\bar{u} \in \mathcal{V}_i$.
$bw_{i,\bar{u}\bar{v}}$	Bandwidth required by link $\bar{u}\bar{v}$, $\bar{u}\bar{v} \in \mathcal{L}_i$.
Ψ_i^{td}	the maximum tolerable delay of SFC request i.
Binary Variables	
$z_{i,uv}^{\bar{u}\bar{v}}$	Whether $\bar{u}\bar{v} \in \mathcal{L}_i$ traverses link $uv \in \mathcal{L}$.
$z_{i,u}^{\bar{u}}$	Whether $\bar{u} \in \mathcal{V}_i$ is served by function node $u \in \mathcal{V}_F$.

$$\sum_{\bar{u} \in \mathcal{V}_i} mem_{i,\bar{u}} z_{i,u}^{\bar{u}} \leq C_u^{mem} r_u^{mem}, \forall u \in \mathcal{V}_F. \tag{2}$$

$$\sum_{\bar{u} \in \mathcal{V}_i} cpu_{i,\bar{u}} z_{i,u}^{\bar{u}} \leq C_u^{cpu} r_u^{cpu}, \forall u \in \mathcal{V}_F. \tag{3}$$

Here, we use the binary variables $z_{i,uv}^{\bar{u}\bar{v}}$ to indicate whether $\bar{u}\bar{v} \in \mathcal{L}_i$ traverses the link $uv \in \mathcal{L}$. The variables $z_{i,uv}^{\bar{u}\bar{v}}$ equal 1, if $\bar{u}\bar{v} \in \mathcal{L}_i$ traverses the link $uv \in \mathcal{L}$, and 0 otherwise. Also, the binary variable $z_{i,u}^{\bar{u}}$ is used to indicate whether $\bar{u} \in \mathcal{V}_i$ is served by the function node $u \in \mathcal{V}_F$. $z_{i,u}^{\bar{u}}$ equals 1, if $\bar{u} \in \mathcal{V}_i$ is served by the function node $u \in \mathcal{V}_F$, and 0 otherwise.

As shown in Eq.(4), the total end-to-end delay which consists of the delay of links and nodes in a path cannot exceed the maximum tolerable delay of SFC request i:

$$\sum_{uv \in \mathcal{L}} \sum_{\bar{u}\bar{v} \in \mathcal{L}_i} d_{uv} z_{i,uv}^{\bar{u}\bar{v}} + \sum_{u \in \mathcal{V}_F} \sum_{\bar{u} \in \mathcal{V}_i} d_u z_{i,u}^{\bar{u}} \leq \Psi_i^{td}. \tag{4}$$

We must guarantee that the links on the path to embedding SFC request i are connected head-to-tail as

$$\sum_{u \in \mathcal{V}} \sum_{\bar{u}\bar{v} \in \mathcal{L}_i} \left(z_{i,uv}^{\bar{u}\bar{v}} - z_{i,vu}^{\bar{u}\bar{v}} \right) = \begin{cases} 1, & u = S_i, \\ -1, & u = T_i, \\ 0, & otherwise. \end{cases} \tag{5}$$

We should ensure that each VNF instance of SFC request i can be only placed on one function node as:

$$\sum_{u \in \mathcal{V}_F} z_{i,u}^{\bar{u}} \leq 1, \forall \bar{u} \in \mathcal{V}_i, \tag{6}$$

where the binary variable $z_{i,u}^{\bar{u}}$ is used to indicate whether VNF instance $\bar{u} \in \mathcal{V}_i$ of SFC request i is placed on function node

$u \in \mathcal{V}_F$. And $z_{i,u}^{\bar{u}}$ equals 1, if $\bar{u} \in \mathcal{V}_i$ is placed on $u \in \mathcal{V}_F$, and 0 otherwise.

After all the above constraints, for SFC request i, the consumption of CPU, memory, and bandwidth resources can be calculated:

$$\textit{CPU}(G_i) = \sum_{u \in \mathcal{V}} \sum_{\bar{u} \in \mathcal{V}_i} cpu_{i,\bar{u}} \cdot z_{i,u}^{\bar{u}} \tag{7}$$

$$\textit{MEM}(G_i) = \sum_{u \in \mathcal{V}} \sum_{\bar{u} \in \mathcal{V}_i} mem_{i,\bar{u}} \cdot z_{i,u}^{\bar{u}} \tag{8}$$

$$\textit{BAND}(G_i) = \sum_{uv \in \mathcal{L}} \sum_{\bar{u}\bar{v} \in \mathcal{L}_i} bw_{i,\bar{u}\bar{v}} \cdot z_{i,uv}^{\bar{u}\bar{v}} \tag{9}$$

So the total resource cost of SFC request i is:

$$\textit{COST}(G_i) = \eta_{cpu} \cdot \textit{CPU}(G_i) + \eta_{mem} \cdot \textit{MEM}(G_i) + \eta_{bw} \cdot \textit{BAND}(G_i) \tag{10}$$

Among them, η_{cpu}, η_{mem}, η_{bw} are weighting factors that weigh different costs. At the same time, for SFC request i, if SFC request i is successfully deployed, the revenue $\textit{REV}(G_i)$ is:

$$\textit{REV}(G_i) = \sum_{\bar{u} \in \mathcal{V}_i} \left(cpu_{i,\bar{u}} + mem_{i,\bar{u}} \right) + \sum_{\bar{u}\bar{v} \in \mathcal{L}_i} bw_{i,\bar{u}\bar{v}} \tag{11}$$

As shown in Eq.(12), the goal of our work is to obtain the path of SFC request i that has the minimum cost and the maximum benefit.

$$\begin{aligned} &\textit{Minimize COST}(G_i) \\ &\textit{Maximize REV}(G_i) \end{aligned} \tag{12}$$

C. MDP Model

To deal with real-time network variations caused by stochastic arrival and departure of requests, we introduce the concept of time slot τ, which can be defined as the integral multiple of a constant time period Δ. At each time slot Δ, the system executes the following procedures: rescanning the physical network, receiving arriving requests, making SFC deployment decisions, and then updating the network states. In particular, we define a list R_τ to represent arriving requests at time slot τ. With all these preparations, we now formally present the MDP model, which is typically defined as $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, where $\mathcal{S}$ is the set of discrete states, $\mathcal{A}$ is the set of discrete actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a transition probability distribution, $\mathcal{R} : \mathcal{S} \times \mathcal{A}$ is the reward function, and $\gamma \in [0, 1]$ is a discount factor for future rewards.

1) State Representation: For each state $s_t \in \mathcal{S}$ in the framework should include two parts, that is, the features of substrate network and the features of arrival requests being processed, which is represented by $s_t = \left(s_t^p, s_t^i\right)$. Here, $s_t^p = (A, X)$ represents the current state of the physical network, where $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the feature matrix of network represents the link features of the physical network. And X is the feature matrix of physical nodes, which contains the available ratios of bandwidth, memory and each type of VNF's CPU of node $u \in \mathcal{V}$. $s_t^i = \left(g_i^{ing}, g_i^{eg}, g_i^{last}, \Psi_{Q_i^j}^{mem*}, \Psi_{Q_i^j}^{cpu*}, \Psi_i^e\right)$ is the current state of SFC request i. Noting that the values of s_t^p are in the range of [0,1], we encode $u \in \mathcal{V}$ based on binary coding. The binary codes of the ingress and egress nodes of SFC request i are denoted as g_i^{ing} and g_i^{eg}. The length of the binary code q satisfies $2^{q-1} \leq |\mathcal{V}| < 2^q$. g_i^{last} represents the binary code of the node successfully placed by the previous VNF in SFC request i. If this VNF is the first in SFC request i, then g_i^{last} is the ingress node. The parameters $\Psi_{Q_i^j}^{mem*}, \Psi_{Q_i^j}^{cpu*}$ represent the normalization of VNF Q_i^j resource consumptions of SFC request i in nodes. Ψ_i^e is represent the embedding result of the current SFC request i. Each VNF of the current SFC request i is set to 1 if it has been placed on the node and 0 otherwise.

2) Action Definition: The agent action set is defined as: $\mathcal{A} = \{1, 2, ..., |\mathcal{V}_F|\}$. $|\mathcal{V}_F|$ indicates the number of function nodes on which VNF instances can be placed.

3) Reward Description: The reward signal is designed to encourage the agent to deploy SFCs to maximize the long-term average revenue and minimize the long-term average cost. Generally, a return positive reward is returned for a successful placement and a negative one returns vice versa. However, successful actions themselves may also vary as they may lead to many possible state representations, and thus make the long-term accumulative rewards different. To make the difference between "slightly good" and "really good" actions, we need to design the reward function more precisely. To achieve a high acceptance ratio, we should encourage successful placement actions. In addition, the learning agent needs not only to make successful but also cost-efficient actions. A better placement policy will consume fewer physical network resources when processing the same SFC request. According to the above two aspects, we design the reward function as:

$$\textit{Reward} = \begin{cases} \gamma_a \frac{\textit{REV}(a_t)}{\textit{COST}(a_t)}, & action\ a_t\ is\ successful, \\ -1, & otherwise. \end{cases} \tag{13}$$

where γ_a is the discount factor that starts from $(1/Size\,(Q_i))$ and gradually increases to 1 when the last VNF of SFC request is in processing, and $Size\,(Q_i)$ means the VNF numbers of an SFC request. $\textit{COST}(a_t)$ and $\textit{REV}(a_t)$ indicate the cost and revenue of the action a_t.

IV. AN A3C-BASED DEEP REINFORCEMENT LEARNING APPROACH FOR ONLINE SFC DEPLOYMENT

In this section, we begin with the architecture of DRL-Deploy together with its neural network design. Then we introduce that how this adaptive online DRL approach, DRL-Deploy works to deploy SFCs with different QoS requirements. Finally, we introduce the A3C-based training procedure of DRL-Deploy.

A. Architecture of DRL-Deploy

With MDP, we can automatically and continually characterize the network traffic variations and network state transitions. Next, we need to find an appropriate, efficient SFC deployment policy that can automatically take appropriate actions in each

state to achieve a high reward. Thus, we propose DRL-Deploy, an adaptive, online A3C-based DRL approach to adaptively deploy SFCs with different QoS requirements.

The architecture of DRL-Deploy is illustrated in Figure 2. To explore the non-Euclidean structure of network topology, we utilize GCN [7] based on semi-supervised learning to extract features of the physical network. At each time step t, the current state of physical network $s_t^p = (A, X)$ is fed into a GCN layer to learn a new representation matrix $Z_t \in \mathbb{R}^{|\mathcal{V}| \times U_{gcn}}$, where U_{gcn} is the units number of GCN layer. The arithmetic operation of GCN is briefly formalized as:

$$Z_t = GCN\left(s_t^p\right) = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}XW\right), \qquad (14)$$

where σ is the activation function, W is the trainable parameters. $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ is the approximated graph convolution filter that is similar to the convolutional neural network(CNN).$\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$ and $\widetilde{A} = A + \Lambda$ is the adjacency matrix of physical network G with added self-connections using a renormalization trick, where Λ is the identity matrix.

With DRL-Deploy, the agent is designed as an actor network and a critic network. A policy is a probability distribution over legal actions $\pi : \pi\left(s_t, a_t\right)$, which indicates the probability to take an action a_t under a state s_t. We first splice and convert the physical network state s_t^p processed by GCN and service function chain state s_t^i into a single-column vector, and then pass through the complete connection layer to make the input size and the action space size consistent. To interpret the output as a probability distribution, we use the well-known softmax layer after the output of the complete connection layer. The softmax function converts any real vector into a vector with a range of (0,1) on each index, and the sum of this vector is also 1, without changing the relative order of the previous vector. Now, the learning agent can use this probability distribution to select actions.

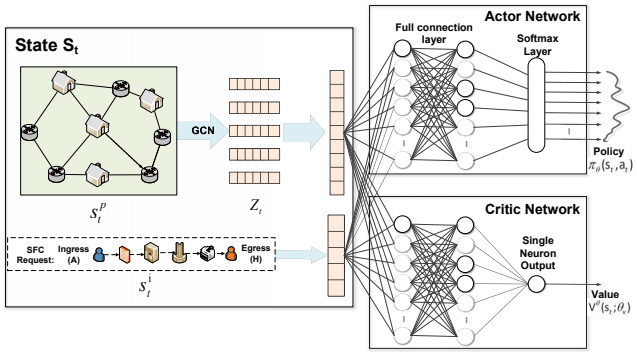

Fig. 2: Architecture of DRL-Deploy.

B. Adaptive, Online Approach for SFC Deployment

To efficiently deal with the dynamic network variations, we introduce the time slot as defined previously. During each time slot τ, the system successively processes all requests in R_τ, making a series of decisions about whether to reject or accept each SFC request and then updates the network states. To reduce the large discrete action space, we devise a serialization-and-backtracking method, which deals with only one VNF within each MDP state transition. DRL-Deploy serially deals with the VNFs of an SFC request and backtracks to the previous network state if an SFC request cannot be completely deployed(i.e., some VNF(s) of the SFC request cannot be placed due to the resource shortage, or the latency or bandwidth constraint of the request can not be satisfied). There are two cases between every two time slots. In the intra-time slot, when there are several arriving requests in a time slot. DRL-Deploy sequentially deals with these arriving requests, specifically one VNF of an SFC request after another. In this case, an MDP state transition happens when a VNF is deployed or rejected. In the inter-time slots, in this case, no action can be taken and the network state stays the same.

The whole procedure of DRL-Deploy is listed in Algorithm 1. The judgment of whether Q_i^j is received in line 10 means that the node meets the resource requirements requested by the service function chain, and the shortest path algorithm(i.e.the Dijkstra algorithm) is used to find the path between the node placed by the previous VNF Q_i^{j-1} and the node. If it is not found, then Q_i^j is rejected and G_i is rejected. In special cases, when Q_i^j is the first VNF requested by the service function chain, the previous node is represented as the ingress node. When Q_i^j is the last VNF requested by the service function chain, it also needs to find the path between Q_i^j and the egress node.

C. Learning Algorithm

We adopt an Asynchronous Advantage Actor-Critic(A3C) algorithm [21], an advanced version of the actor-critic based policy gradient method for training. This algorithm makes two major improvements. Firstly, it uses the advantage function instead of the mere state-action value function which can reduce the variance of training experience. Second, it is a "master-worker" parallel training architecture composed of multiple worker agents U and a master agent. There are two networks in this algorithm: the actor network(with a set of parameters θ), which is used to generate policy π_θ, and the critic network(with a set of parameters θ_v), which generates the estimation of values in different states $V^{\pi_\theta}(s_t;\theta_v)$ and helps compute the advantage function. In principle, these two networks share a similar structure except for the output layer.

The traditional policy gradient method uses the following gradient of accumulative discounted expected rewards:

$$\bigtriangledown_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] = \mathbb{E}_{\pi_\theta}\left[\bigtriangledown_\theta log\pi_\theta\left(s,a\right)Q^{\pi_\theta}\left(s,a\right)\right] \qquad (15)$$

where $Q^{\pi_\theta}\left(s,a\right)$ is the state-action value function that estimates the expected long-term return of action a derived from policy π_θ under state s. However, the variance of $Q^{\pi_\theta}\left(s,a\right)$ is usually high, making the training process unstable. And if a state is in a good position, the Q value will stay high regardless of actions picked under this state, and it can overlook the

Algorithm 1 The DRL-Deploy Procedure

Begin: Initiate time slot $\tau \leftarrow 1$
while $R_\tau = \emptyset$ **do**
  $\tau \leftarrow \tau + 1$
end while
Select an SFC request G_i from R_τ based on the arriving time
Initiate $i \leftarrow 1$, $j \leftarrow 1$
for $t \leftarrow 1, T$ **do**
  Initiate state s_t
  Take action a from $\mathcal{A}$ to place Q_i^j
  if Q_i^j is accepted **then**
    $j \leftarrow j + 1$, $s_t \leftarrow s_{t+1}$
  end if
  if G_i is rejected or $j > |Q_i|$ **then**
    if G_i is rejected and $j > 1$ **then**
      Backtrack the network state to s_{t-j+1}
    end if
    if R_τ is all processed **then**
      repeat
        $\tau \leftarrow \tau + 1$
      until $R_\tau = \emptyset$
      Select a new SFC request G_1' from R_τ
      Reset $i \leftarrow 1$, $j \leftarrow 1$
    else
      Select request G_{i+1} from R_τ
      $i \leftarrow i + 1$
    end if
  end if
  Calculate the reward R_t
end for

difference between actions. Therefore, We use the advantage function instead of $Q^{\pi_\theta}(s, a)$, the advantage function indicates how better is the current action a from the "average action" derived from the corresponding policy under a certain state s. Based on the Bellman equation, the advantage function can be expressed as:

$$A^{\pi_\theta}(s, a) = r_t + \gamma V_\theta^{\pi}(s_{t+1}) - V^{\pi_\theta}(s) \tag{16}$$

where $V^{\pi_\theta}(s)$ is the state value function that estimates the accumulative return under state s. And $\gamma \in (0, 1)$ denotes the discount factor.

The update of actor network follows the policy gradient training method:

$$\theta \leftarrow \theta + \alpha \sum_t \bigtriangledown_\theta log \pi_\theta(s_t, a_t) A^{\pi_\theta}(s_t, a_t) + \beta \bigtriangledown_\theta H(\pi_\theta(\cdot \mid s_t)) \tag{17}$$

where $H(\cdot)$ is the entropy of the policy at each time step, and $\pi_\theta(\cdot \mid s_t)$ indicates the probabilistic distribution over actions under the states s_t. This entropy term is used as regularization to encourage the agent to explore the action space. Finally, α is the learning rate of the actor network, and β is a decaying factor that starts with a large value but shrinks as the training proceeds.

Algorithm 2 The DRL-Deploy With Parallel Training Algorithm

/**Master**/
Initialize the actor network and critic network;
Initialize the number of workers U;
for u in U **do**
  Create a worker agent $w[u]$ with a same copy of actor network and critic network;
end for
while TRUE **do**
  for u in U **do**
    Collect the experiences generated by $worker[u]$;
  end for
  Update the parameter of actor network and critic network using previously collected experiences under policy gradient training method;
  for u in U **do**
    Push the newest version of actor network and critic network to $w[u]$;
  end for
end while
/**Worker**/
Initialize the actor network and critic network;
Initialize the independent environments for SFC deployment;
while TRUE **do**
  Receive the parameters of actor network and critic network from master;
  Sample a trajectory from the environment using the copied network;
  Send the trajectory as $\{s_t, a_t, r_t, s_{t+1}\}$ experience tuples to master;
end while

The critic network is updated using the standard Temporal-Difference(TD) method:

$$\theta_v \leftarrow \theta_v + \alpha' \sum_t \bigtriangledown_{\theta_v} \left(r_t + \gamma V_\theta^{\pi}(s_{t+1}; \theta_v) - V^{\pi_\theta}(s_t; \theta_v)\right)^2 \tag{18}$$

where $V^{\pi_\theta}(\cdot; \theta_v)$ is the estimation of value function under a certain state, and α' is the learning rate of the critic network. And the A3C-based training procedure is listed in Algorithm 2.

V. PERFORMANCE EVALUATION

A. Simulation Setup

In the simulation, we randomly generate a physical network with 100 nodes and 250 links, following the Waxman topology model, which imitates a medium-sized InP. We treat all nodes as functional nodes. The resources of nodes and the bandwidth of links in the physical network are uniformly distributed with 50 to 150 units. We have also generated many service function chain requests. Each service function chain request contains 2 to 15 VNFs. The computing power of each VNF follows a normal distribution between 1 and 50 units. The bandwidth

requirement of each logical link follows a normal distribution between 1 and 50 units. The service function chain requests arrive according to Poisson distribution, and there is an average of 4 requests in 100 time units. The duration of each request follows an exponential distribution with an average of 1000 time units. The timeline we generated lasts about 50,000 time units, which contains about 2,000 requests. We divide the requests evenly into two groups: the training group and the test group.

The whole model architecture is built with Tensorflow and Adam optimizer is employed to update the parameters of neural networks. Our simulation experiments are executed on a computer with a 3.60 GHz Intel Core i7-7700 CPU and 16 GB RAM. In the training phase, actors utilize ϵ-greedy search strategy to select actions according to the probability distributions, which facilitates sufficient exploration and excellent exploitation. In the testing phase, only the actor network of the trained agent works to place SFC requests, using the greedy search strategy. The values of simulation parameters are given in Table II.

TABLE II: SIMULATION PARAMETERS

Name	Value	Description
U	4	the number of actor networks
η_{cpu}, η_{mem}, η_{bw}	1	the weighting factors of different costs
α	0.005	the learning rate of actor network
α'	0.01	the learning rate of actor network
γ	0.95	the discount factor of TD error
U_{gcn}	64	the units number of GCN layer

B. Baseline Algorithms

To verify the effectiveness of the proposed DRL-Deploy approach, we select the following two approaches for comparison:

- **GRC**: A heuristic-based algorithm based on global resource capacity to map VNFs onto physical nodes.
- **MCTS**: A RL-based algorithm using Monte Carlo tree search to make the SFC placement decision.

Both of them utilize the shortest path algorithm to conduct link mapping.

C. Simulation Results

1) Acceptance Ratio: The SFC requests acceptance ratio of the physical network of the duration step $0 \sim T$ can be defined as:

$$ACR(T) = \frac{\sum_{t=0}^{T} NUM\_SFC\_S}{\sum_{t=0}^{T} NUM\_SFC} \tag{19}$$

where NUM_SFC_S and NUM_SFC are the number of successful SFC request deployment and the numbers of total SFC requests, respectively. The acceptance ratio of all the three algorithms in the test group is illustrated in Fig.3. We can observe that at the beginning, the acceptance ratios of all the three algorithms decrease because the resources of the physical network are gradually occupied as SFC requests arrive. At the end of time step T, DRL-Deploy showed the highest acceptance ratio of 77.35%. Compared to GRC and MCTS, DRL-Deploy improves performance up to 8.6% and 23.2%. This shows that our algorithm can reasonably allocate resources for each SFC in the physical network with limited resources, thereby deploying more SFC requests.

2) Long-term Average Revenue and Cost: The long-term average revenue and cost of the duration step $0 \sim T$ can be decided by:

$$\textit{REV}(T) = \frac{\sum_{t=0}^{T} \textit{REV}(G^t)}{T} \tag{20}$$

$$\textit{COST}(T) = \frac{\sum_{t=0}^{T} \textit{COST}(G^t)}{T} \tag{21}$$

where $\textit{REV}(G^t)$ and $\textit{COST}(G^t)$ denotes the revenue and cost of successful SFC request deployment arrived at time step t, respectively. In Fig.4 and Fig.5, we compare the long-term average revenue and cost of these three algorithms when SFC requests arrive in sequence. We can observe that due to the superiority of the DRL-Deploy algorithm in terms of acceptance rate, the DRL-Deploy algorithm will be significantly higher than other algorithms in terms of long-term average income. At the end of time step T, DRL-Deploy reached the highest long-term average income of 5.94, which was 22.9% and 53.6% higher than GRC and MCTS, respectively. At the same time, the DRL-Deploy algorithm has significantly reduced the long-term average cost. The lowest long-term average cost of DRL-Deploy is 5. Compared with GRC and MCTS, this is a decrease of 62.4% and 38.2%, respectively. This means that our algorithm can provide better SFC deployment quality and make more efficient use of physical network resources.

3) Resource Utilization: To reflect the superiority of our algorithm in resource utilization, in Fig.6, we compare the average CPU utilization of the physical network at time step T. In the whole operation process, with a relatively high acceptance rate, the DRL-Deploy algorithm has a high CPU average node utilization. At the end of time step T, DRL-Deploy shows high node utilization in the physical network. Compared with GRC and MCTS, the node utilization rate has increased by 36.4% and 71.6%, respectively. This shows that our algorithm can intelligently make SFC deployment decisions based on the inherent information required by the physical network and SFC.

VI. Conclusion

In this paper, we proposed a novel DRL-based method for the SFC deployment problem by formulating it as the MDP model, called DRL-Deploy. DRL-Deploy uses GCN to effectively extract the characteristics of the physical network. In addition, we designed the A3C algorithm to speed up the training process and enhance the robustness of the model. Due to the sufficient information obtained from the environment, DRL-Deploy can intelligently generate SFC deployment strategies with different QoS requirements, which helps to improve the resource utilization of the physical network.

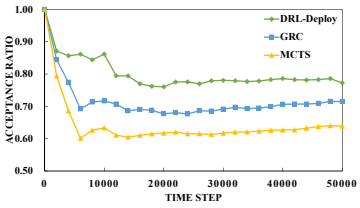

Fig. 3: SFC requests acceptance ratio of the physical network at time step T.

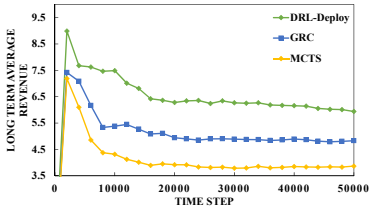

Fig. 4: Long-term average revenue of the physical network at time step T.

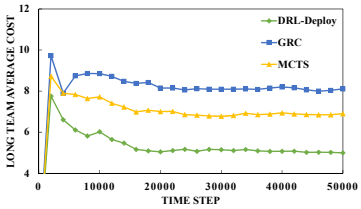

Fig. 5: Long-term average cost of the physical network at time step T.

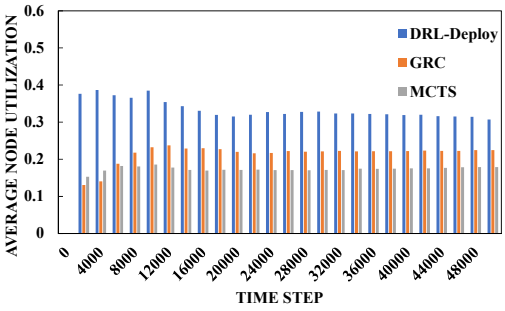

Fig. 6: Average node utilization of the physical network at time step T.

VII. Acknowledgment

This work is supported by the National Natural Science Foundation of China (NSFC) (Grants No. 61772228, No. U19A2061), National key research and development program of China under Grants No. 2017YFC1502306, Interdisciplinary Research Funding Program for Doctoral Students of Jilin University under Grants No. 101832020DJX063, No. 101832020DJX007 and the Natural Science Foundation of Jilin under Grant No. YDZJ202101ZYTS191.

References

[1] Mijumbi R, Serrat J, Gorricho JL, Bouten N, De Turck F, Boutaba R. Network function virtualization: State-of-the-art and research challenges. IEEE Comm. surveys & tutorials, 2015, 18(1):236-62.

[2] M. Qiu, H. Li, E.H.-M. Sha, Heterogeneous real-time embedded software optimization considering hardware platform, ACM Sym. on Applied Computing, pp. 1637-1641, 2009.

[3] K. Lin, W. Wang, et al., Human localization based on inertial sensors and fingerprints in the Industrial Internet of Things, Computer Networks, Vol. 101, pp. 113-126, 2016.

[4] Karakus M, Durresi A. Quality of service (QoS) in software defined networking (SDN): A survey. J. of Network and Computer App.. 2017, 80:200-18.

[5] Akyildiz IF, Lee A, Wang P, Luo M, Chou W. Research challenges for traffic engineering in software defined networks. IEEE Network, 2016, 30(3):52-8.

[6] Laghrissi A, Taleb T. A survey on the placement of virtual resources and virtual network functions. IEEE Comm. Surveys & Tutorials. 2018, 21(2):1409-34.

[7] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[8] Pei J, Hong P, Pan M, Liu J, Zhou J. Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks. IEEE J. on Selected Areas in Comm.. 2019, 38(2):263-78.

[9] Solozabal R, Ceberio J, et al., virtual network function placement optimization with deep reinforcement learning. IEEE J. on Selected Areas in Comm.. 2019, 38(2):292-303.

[10] Mechtri M, Ghribi C, Zeghlache D. A scalable algorithm for the placement of service function chains. IEEE Trans. on Network and Ser. Mana.. 2016, 13(3):533-46.

[11] M. Qiu, Z. Ming, J. Li, J. Liu, G. Quan, Y. Zhu, Informer homed routing fault tolerance mechanism for wireless sensor networks, J. of Sys. Arch., 59 (4-5), 260-270, 2013

[12] M. Qiu, J. Liu, J. Li, Z. Fei, Z. Ming, E.H.-M. Sha, A novel energy-aware fault tolerance mechanism for wireless sensor networks, IEEE/ACM Int'l Conf. on Green Comp. and Comm., 2011

[13] H. Su, M. Qiu, H. Wang, Secure wireless communication system for smart grid with rechargeable electric vehicles, IEEE Comm. Maga, 50 (8), 62-68, 2012

[14] K. Gai, M. Qiu, H. Zhao, X. Sun, Resource management in sustainable cyber-physical systems using heterogeneous cloud computing, IEEE Trans. on Sus. Comp., 3 (2), 60-72, 2017

[15] Z. Lu, N. Wang, J. Wu, M. Qiu, IoTDeM: An IoT Big Data-oriented MapReduce performance prediction extended model in multiple edge clouds, JPDC, Vol.118, pp. 316-327, 2018

[16] K Gai, M Qiu, Optimal resource allocation using reinforcement learning for IoT content-centric services, Applied Soft Comp., 70, 12-21, 2018

[17] K Gai, M Qiu, Reinforcement learning-based content-centric services in mobile sensing, IEEE Network, 32 (4), 34-39, 2018

[18] Mao H, Alizadeh M, Menache I, Kandula S. Resource management with deep reinforcement learning, 15th ACM workshop on hot topics in net., 2016, pp. 50-56.

[19] Xiao Y, Zhang Q, Liu F, et al., Adaptive online service function chain deployment with deep reinforcement learning. Int'l Sym. on Quality of Ser., 2019, pp. 1-10.

[20] Gong L, Wen Y, Zhu Z, Lee T. Toward profit-seeking virtual network embedding algorithm via global resource capacity. IEEE INFOCOM, 2014, pp. 1-9.

[21] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. Int'l conf. on machine learning, 2016, pp. 1928-1937, PMLR.

[22] Lin T, Zhou Z, Tornatore M, Mukherjee B. Optimal network function virtualization realizing end-to-end requests. IEEE GLOBECOM, 2015, pp. 1-6.

[23] Addis B, Belabed D, Bouet M, Secci S. Virtual network functions placement and routing optimization. IEEE 4th Int'l Conf. CloudNet, 2015, pp. 171-177.

[24] Yu R, Xue G, Zhang X. QoS-aware and reliable traffic steering for service function chaining in mobile networks. IEEE J. on Selected Areas in Comm. 2017, 35(11):2522-31.

[25] Dwaraki A, Wolf T. Adaptive service-chain routing for virtual network functions in software-defined networks, workshop on Hot topics in Middleboxes and NFV, 2016, pp. 32-37.

[26] Jiao S, Zhang X, Yu S, Song X, Xu Z. Joint virtual network function selection and traffic steering in telecom networks. IEEE GLOBECOM 2017, pp. 1-7.

[27] Soualah O, Mechtri M, Ghribi C, Zeghlache D. An efficient algorithm for virtual network function placement and chaining. 14th IEEE CCNC 2017, pp. 647-652.

[28] Hong P, Xue K, Li D. Resource aware routing for service function chains in SDN and NFV-enabled network. IEEE Trans. on Ser. Comp., 2018,.

[29] https://docs.opendaylight.org/en/stable-sodium/developer-guide/service-function-chaining.html#service-function-scheduling-algorithms