NFVdeep: adaptive online service function chain deployment with deep reinforcement learning

Conference Paper · June 2019

DOI: 10.1145/3326285.3329056

Zhang Qixia
Huazhong University of Science and Technology
7 PUBLICATIONS **169** CITATIONS
SEE PROFILE

Fangming Liu
State Oceanic Administration
74 PUBLICATIONS **2,486** CITATIONS
SEE PROFILE

Jia Wang
The Hong Kong Polytechnic University
15 PUBLICATIONS **188** CITATIONS
SEE PROFILE

NFVdeep: Adaptive Online Service Function Chain Deployment with Deep Reinforcement Learning

Yikai Xiao[1], Qixia Zhang[1], Fangming Liu*[1],
Jia Wang[2], Miao Zhao[2], Zhongxing Zhang[1], Jiaxing Zhang[1]
[1]National Engineering Research Center for Big Data Technology and System,
Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2] Department of Computing, The Hong Kong Polytechnic University, Hong Kong

ABSTRACT

With the evolution of network function virtualization (NFV), diverse network services can be flexibly offered as service function chains (SFCs) consisted of different virtual network functions (VNFs). However, network state and traffic typically exhibit unpredictable variations due to stochastically arriving requests with different quality of service (QoS) requirements. Thus, an adaptive online SFC deployment approach is needed to handle the real-time network variations and various service requests. In this paper, we firstly introduce a Markov decision process (MDP) model to capture the dynamic network state transitions. In order to jointly minimize the operation cost of NFV providers and maximize the total throughput of requests, we propose *NFVdeep*, an adaptive, online, deep reinforcement learning approach to automatically deploy SFCs for requests with different QoS requirements. Specifically, we use a serialization-and-backtracking method to effectively deal with large discrete action space. We also adopt a policy gradient based method to improve the training efficiency and convergence to optimality. Extensive experimental results demonstrate that NFVdeep converges fast in the training process and responds rapidly to arriving requests especially in large, frequently transferred network state space. Consequently, NFVdeep surpasses the state-of-the-art methods by 32.59% higher accepted throughput and 33.29% lower operation cost on average.

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**; **Network resources allocation**; **Network dynamics**; • **Computing methodologies → Machine learning**.

*The corresponding author is Fangming Liu (fmliu@hust.edu.cn). This work was supported in part by the NSFC under Grant 61761136014 (and 392046569 of NSFC-DFG) and 61722206 and 61520106005, in part by National Key Research & Development (R&D) Plan under grant 2017YFB1001703, in part by the Fundamental Research Funds for the Central Universities under Grant 2017KFKJXX009 and 3004210116, and in part by the National Program for Support of Top-notch Young Professionals in National Program for Special Support of Eminent Professionals.

KEYWORDS

Network Function Virtualization (NFV), Deep Reinforcement Learning, Service Function Chain, QoS-Aware Resource Management

ACM Reference Format:
Yikai Xiao[1], Qixia Zhang[1], Fangming Liu*[1], and Jia Wang[2], Miao Zhao[2], Zhongxing Zhang[1], Jiaxing Zhang[1]. 2019. *NFVdeep*: Adaptive Online Service Function Chain Deployment with Deep Reinforcement Learning. In *IEEE/ACM International Symposium on Quality of Service (IWQoS '19), June 24–25, 2019, Phoenix, AZ, USA.* ACM, 10 pages. https://doi.org/10.1145/3326285.3329056

1 INTRODUCTION

By shifting the way of implementing hardware middleboxes (e.g., firewalls, WAN optimizers and load balancers) to software-based virtual network function (VNF) instances, network function virtualization (NFV) emerges as a promising paradigm that embraces great flexibility, agility and efficiency [5, 17, 39]. Expected to reduce Capital Expenditure (CAPEX) and Operational Expenditure (OPEX), NFV offers new ways to design, orchestrate, deploy and manage a variety of network services for supporting growing customer demands [10, 23]. Notably, NFV and software defined network (SDN) are regarded as two of the most important enabling technologies that would be the keystones for 5G systems [36].

In an NFV system, a service request is typically represented by a service function chain (SFC), which is a sequence of network functions (NFs) that have to be processed in a pre-defined order [13, 43]. Generally, conventional hardware NFs are fixed with physical locations; on the contrary, NFV permits software-based VNFs to be placed in any resource-sufficient virtual machines (VMs) or containers deployed on commercial-off-the-shelf (COTS) servers [5]. In other words, NFV offers a good opportunity to improve the system performance and quality of service (QoS) by determining how to deploy the service-required SFCs among multiple candidate COTS servers in NFV network and further in service-customized 5G network slices [42].

Currently, some efforts have been paid to tackle the *VNF placement* problem or the *SFC deployment* problem, which is proved NP-hard [1, 43], and thus heuristic solutions are usually proposed for different optimization objectives [5]. Nevertheless, there still exist some challenges that have not been completely solved in previous works: (1) network state and traffic typically exhibit great variations due to stochastic arrival of requests [12], thus an appropriate model is needed to capture the dynamic network state transitions; (2) different network service requests may have different traffic characteristics (e.g., flow rate and packet size), QoS

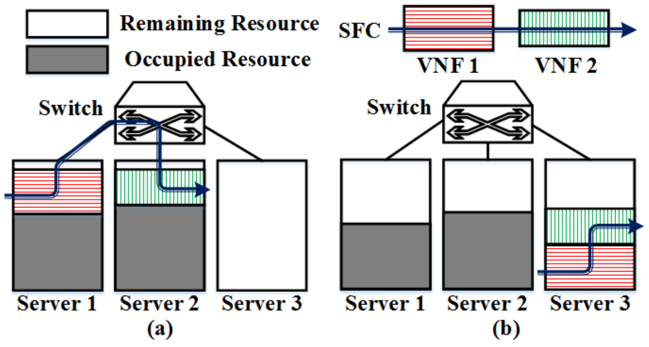

Figure 1: An example of two ways to deploy an SFC. There are three servers with different remaining resources. The SFC is composed of VNF 1 and VNF 2 with different resource demands.

requirements (e.g., latency, bandwidth and security requirements), thus an adaptive, online approach is needed to automatically deploy SFCs with different demands; (3) NFV providers and customers usually pursue different objectives (i.e., reducing operation cost and improving QoS), which can even be conflicting ones, thus it is usually difficult to achieve a win-win SFC deployment solution that jointly maximizes both sides' profits.

For instance, Fig. 1 plots two alternative solutions for deploying a request's SFC. Solution (a) benefits the NFV providers since they can shut down the idle server to reduce the operation cost, while it increases the bandwidth occupation of the NFV system and the communication latency of the request. On the other hand, as a result of VNF consolidation [37, 41], solution (b) improves the customers' QoS with improved communication efficiency; while as one more server is used, the operation cost would increase accordingly. Thus, it is worth investigating how to balance the trade-off between NFV providers and customers, and further considering how to jointly optimize the SFC deployment with multiple objectives.

Even though existing works like [1, 7, 25, 33, 38] have investigated the SFC deployment problem, [1, 7, 38] assume that the set of arriving requests is predetermined, regardless of the real-time network variations. Differing from previous works, we propose NFVdeep, an adaptive, online, deep reinforcement learning (DRL) approach for SFC deployment, which integrally tackles the three challenges. Firstly, to address the first challenge, we introduce a Markov decision process (MDP) model to capture the dynamic network state transitions. We construct the MDP state as the current network resource utilization (i.e., CPU, memory and bandwidth) and deployment results of currently-running SFCs. We also define the action as the VNF's deployment strategy for an arriving request. In this way, dynamic network variations can be automatically and continually expressed as MDP state transitions.

To address the second challenge, we devise a policy gradient (PG) based DRL [18] approach to automatically deploy SFCs. DRL is an emerging approach to deal with large network state space and real-time network state transitions; while PG is a model-free approach, which shows good advantages in improving the training efficiency and convergence to optimality. After converged in the training procedure, NFVdeep can efficiently provide a high-reward SFC deployment solution to each arriving request, considering its resource demand and the current resource utilization. This online, adaptive method can support a variety of service requests with different QoS requirements in NFV network and further 5G network slices.

To address the third challenge, we aim to jointly achieve two objectives: (1) *minimizing the operation cost of occupied servers* for NFV providers and (2) *maximizing the total throughput of accepted requests* for customers. We model them together in the MDP reward function, which is defined as the weighted total throughput of accepted requests (income) minus the weighted total cost of occupied servers (expenditure) for deploying SFCs. This reward can be regarded as the total profits of the NFV system. Besides, we use a *serialization-and-backtracking* method to reduce the high-dimension action space for placing SFCs among muitiple candidate servers. In particular, we serially deal with each VNF of an SFC within each MDP state transition, and backtrack to the previous state if an SFC cannot be completely deployed.

In summary, the PG based NFVdeep can respond rapidly to stochastically arriving requests and automatically provide QoS-aware SFC deployment solutions with high rewards. Through extensive trace-driven simulations, we demonstrate that NFVdeep not only handles dynamic network variations well, but also achieves superior performance as compared with the state-of-the-art methods.

The main contributions of this paper are as follows:

- To capture the real-time network variations, we use an MDP model to formulate the online SFC deployment problem, where network variations are automatically and continually expressed as MDP state transitions.
- In order to jointly minimize the operation cost of NFV providers and maximize the total throughput of requests, we propose NFVdeep, an adaptive, online, PG based DRL approach to automatically deploy SFCs for requests with different QoS requirements.
- Extensive trace-driven simulation results show that NFVdeep not only responds rapidly to dynamic arriving requests, but also outperforms the state-of-the-art solutions by 32.59% higher accepted throughput and 33.29% lower operation cost on average.

The rest of this paper is organized as follows. Section II discusses the related work and motivation. Section III presents the MDP model and the formulation of the SFC deployment problem. In Section IV, we formally propose NFVdeep, an adaptive, online, PG based DRL approach with detailed introductions on its architecture, neural network design and training procedure. Section V presents the evaluations and at last, Section VI concludes the paper.

2 RELATED WORK AND MOTIVATION

In this section, we first investigate the state-of-the-art works dealing with the VNF placement problem. Then we elaborate on why adaptive online SFC deployment is needed and why PG based DRL approach is suited to solve the problem.

2.1 VNF Placement

Recent years have witnessed the proliferation of the studies on the *VNF placement* problem and the *SFC deployment* problem considering the service chaining requirements. Typically this problem is categorized as a resource management problem in NFV systems

[23], which has been proved NP-hard in [1]. In order to achieve some specific optimization objectives (e.g., minimizing the number of occupied servers, minimizing the end-to-end latency and maximizing the acceptance rate of requests), some mathematical programming methods such as integer linear programming (ILP) [34] and mixed ILP (MILP) [20] are generally used. Since deriving the optimal solutions is usually computational-expensive in large network scale, heuristic solutions are usually proposed with near-optimal solutions but small execution time (e.g., [15, 21, 30, 43]).

For instance, Moens *et al.* [25] formulate the VNF placement problem as an ILP, which aims to allocate SFCs to the physical network with the minimum number of used servers. Cohen *et al.*[7] explore the NFV location problem and solve it by jointly placing network functions and calculating path in the embedding process. Addis *et al.*[1] propose a VNF chaining and placement model, formulate it as an MILP and then devise a heuristic algorithm.

Specifically, these VNF placement or SFC deployment solutions can be categorized based on their application scenarios, such as traditional core cloud datacenters or NFV providers' network (e.g., [1, 7, 19, 22, 25, 30, 44]), geo-distributed datacenters (e.g., [11]), content delivery network (CDN, e.g., [9]), autonomous response network (e.g., [27]), edge cloud datacenters (e.g., [16]), 5G network and service-specific 5G network slices (e.g., [2, 6]). However, these existing solutions have some limitations, for instance [7] does not consider VNF service chains (i.e., SFCs) and [1, 7, 33, 38] assume that the set of requests is pre-determined, regardless of the real-time arrival of requests, which may cause severe network traffic fluctuations.

2.2 Why Need Adaptive Online SFC Deployment?

In summary, these existing works have not integrally tackled the three challenges as mentioned in Section I. First, many existing VNF placement and SFC deployment solutions do not take account of the real-time network variations. In fact, network state and traffic typically exhibit unpredictable variations due to stochastic arrival of requests, thus an appropriate model is needed to capture the dynamic network state transitions.

Second, most existing solutions are customized for a certain application scenario (e.g., the traditional core cloud datacenters, CDN or 5G network). Despite the efficiency of scenario-customized solutions, their network-specific or service-specific constraints make them difficult to be flexibly applied to other network topologies and scenarios. Thus, an efficient, adaptive solution is needed to automatically deploy SFCs for NFV providers without manual intervention, meanwhile considering each request's traffic characteristics (e.g., flow rate) and QoS requirements (e.g., latency and bandwidth requirements). This solution should be scalable in different network typologies and applicable to different scenarios, such as different 5G use cases provided by different 5G network slices.

Third, most existing works (e.g., [1, 7, 25, 34, 38]) basically focus on optimizing the VNF placement or the SFC deployment from the NFV providers' perspective (i.e., minimizing the CAPEX and OPEX, resource allocation and power consumption) while do not consider improving the customers' QoS. Only a few works consider optimizing the SFC deployment from the customers' perspective (e.g., [19, 33, 40]), aiming at improving the request acceptance rate or reducing the end-to-end response latency. Even though it is challenging to deal with the NP-hard SFC deployment problem with multiple objectives (sometimes even conflicting ones), we still find a way to jointly maximize the profits of both NFV providers and customers. In particular, we combine the two objectives (i.e., minimizing the operation cost and maximizing the total throughput of requests) in the MDP reward function, which can automatically maximize the NFV system's profits in the SFC deployment process.

2.3 Why Adopt PG Based DRL Approach?

Deep Reinforcement Learning. In recent years, deep reinforcement learning (DRL) has been prevailing in natural language processing problems, robotics, decision games, etc., and achieves superior results, like the Deep Q Learning (DQN) [24] algorithm and AlphaGo [32]. DRL combines deep learning with reinforcement learning (RL) to implement machine learning from perception to action [18]. Based on MDP, RL trains an intelligent agent to learn policies directly through interacting with the environment and automatically maximize the reward. The general RL approach maintains a look-up table to store policies, which is not capable of dealing with large infinite state space. To overcome this problem, DRL approach emerges, utilizing deep neural network (DNN) as an approximation function to learn policy and state value representations. In NFV systems, the network state space is associated with the number of servers and links and the frequency of state transition is positively correlated with the the number of arriving requests. With DRL, NFVdeep can efficiently handle a large number of real-time arriving requests especially in large, frequently transferred network state space. In addition, NFVdeep can flexibly scale with different network topologies and can also be easily applied to different application scenarios.

Policy Gradient. The DRL approach can be classified into two categories, one is value-based approach (e.g., DQN) and the other is policy-based approach. Policy gradient (PG) [29] based approach enables automatic feature engineering and end-to-end learning, thus the reliance on domain knowledge is significantly reduced and even removed [18]. For some environments with continuous control or particularly large action space, it is difficult to calculate all the value functions to get the best strategy with the value-based DQN approach. In the SFC deployment problem, even for one VNF, there are usually many candidate servers for placing it; the chaining of VNFs makes the action space even larger. Under these circumstances, the policy-based approach is more practicable and efficient, since it not only shows a good advantage in high-dimension action space, but also improves the training efficiency and convergence to the optimal solution. The core concept of PG is that if an action results in more rewards, then increase the probability of its occurrence; otherwise, reduce the probability of its occurrence. Through PG based training procedure, NFVdeep can efficiently and automatically learn and act to arriving requests with different QoS requirements.

3 MODEL AND PROBLEM FORMULATION

In this section, we begin with the NFV system description including the NFV network structure, VNFs and requests. Then we elaborate

Table 1: Key Notations

Symbol	Description
G	The graph $G = (V, E)$ representing the underlying NFV network
V	The set of server nodes within the network
E	The set of edges (or links) between the servers, $\forall e = (v_1, v_2) \in E, v_1, v_2 \in V$
F	The set of service-required VNFs
R	The set of service requests
$C_v = (C_v^{cpu}, C_v^{mem})$	The quantity of available resources of server $v \in V$ in terms of CPU and memory
$D_f = (D_f^{cpu}, D_f^{mem})$	The resource demand of VNF $f \in F$ in terms of CPU and memory
$T_{u,v}$	The communication latency between every two nodes $u, v \in V$
W_v	The output bandwidth of server node $v \in V$
W_r	The bandwidth demand of request $r \in R$
λ_r	The packet arrival rate of request $r \in R$
T_r	The response latency limitation of request $r \in R$
t_r	The total end-to-end latency of request $r \in R$
τ_r	The time to live (TTL) of the SFC for request $r \in R$, $\tau_r = l * \Delta, l \in \mathbb{N}, r \in R$
$a_{r,\tau}$	1 if a request $r \in R$ is in service in time slot $[\tau_r^s, \tau_r^s + \tau_r]$, 0 otherwise
$n_{v,\tau}^f$	The number of service instances of VNF $f \in F$ that are deployed on server $v \in V$ in time slot $[\tau_r^s, \tau_r^s + \tau_r]$
$u_{v,\tau}^r$	1 if any VNF of request $r \in R$ is placed at node $v \in V$ in time slot $[\tau_r^s, \tau_r^s + \tau_r]$, 0 otherwise
$x_{r,v}^i$	1 if the i-th VNF of SFC for request $r \in R$ is deployed at node $v \in V$, 0 otherwise
y_r	1 if request $r \in R$ is accepted, 0 otherwise
$z_{v,\tau}$	1 if any VNF instance is placed on node $v \in V$ in time slot $[\tau_r^s, \tau_r^s + \tau_r]$, 0 otherwise

on how to use the MDP model to capture the dynamic network state transitions. At last, we formally present the SFC deployment problem formulation with objectives and constraints. Key notations are listed in Table 1.

3.1 NFV System Description

In a common NFV network structure (e.g., three-tier topology or fat-tree), the server nodes are connected via multi-level switches [35]. Thus we represent the network as a connected graph $G = (V, E)$, where V is the set of server nodes, and E is the set of edges (or links) that connect every two nodes, $\forall e = (v_1, v_2) \in E, v_1, v_2 \in V$. In fact, each server has a resource capacity, which contains the computing resources (i.e., CPU), memory and storage resources (i.e., RAM and hard disk). Thus, we denote the resource capacity of each server by $C_v = (C_v^{cpu}, C_v^{mem})$, representing its quantity of available resources in terms of CPU and memory. Note that other types of resource are relatively sufficient in servers, such as hard disk storage; while they can also be added in C_v if necessary. W_v represents the total output bandwidth of server $v \in V$ and $T_{u,v}$ represents the communication latency on the link between server nodes $u \in V$ and $v \in V$.

We use $F = \{f_1, f_2, ..., f_{|F|}\}$ to represent the service-required VNFs, including commonly-used ones, such as firewall, network address translation (NAT), deep packet inspection (DPI), load balancer (LB), traffic monitor, etc., and other service-customized VNFs (e.g., video processing VNF). Each service instance of VNF $f \in F$ has a resource demand in terms of CPU and memory, denoted by $D_f = (D_f^{cpu}, D_f^{mem})$.

Next, we use R to denote the set of stochastically arriving requests. Since each request $r \in R$ needs to be steered through a sequence of VNFs based on its service requirements, we denote its service-related SFC $\widehat{r}$ as follows:

$$\widehat{r} = [f_1, f_2, ..., f_{|\widehat{r}|}], f_i \in F, i = 1, 2, ..., |\widehat{r}|.$$

Each service request $r \in R$ has its traffic characteristics, i.e., the packet arrival rate λ_r and its specific QoS requirements, including the bandwidth requirement W_r and the response latency limitation T_r. Besides, for each request $r \in R$, we use a binary variable $x_{r,v}^i$ to indicate the deployment decision of each VNF f_i in sequence $\widehat{r}$ of its SFC. Specifically, if VNF f_i can be successfully placed on server node $v \in V$, $x_{r,v}^i = 1$; otherwise, $x_{r,v}^i = 0$.

3.2 MDP Model

To deal with the 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 (i.e., $\tau = n * \Delta, n \in \mathbb{N}, \Delta = 1\ \mu$s, 1 ms or 1 s based on the actual demand). At each time slot τ, the NFV system executes the following procedures: rescanning all the servers and links, removing timeout requests, receiving arriving requests, making SFC deployment decisions and then updating the network states. Thus, at time slot τ, we define $\widehat{C}_{v,\tau}$ as the remaining resource capacity of server node $v \in V$ after removing timeout requests, and $\widehat{W}_{v,\tau}$ as the remaining output bandwidth.

In particular, we define a list $R_\tau \subset R$ to represent arriving requests at time slot τ. In each time slot, if a request is arriving alone, it can be handled immediately; if several requests are arriving simultaneously, they will be processed in serial based on the arriving time. Different requests can also be processed in pipeline and even in parallel for improving efficiency if the NFV system supports. We also denote the arriving time of request $r \in R$ by $\tau_r^s = m * \Delta, m \in \mathbb{N}$, and its SFC's time to live (TTL) by $\tau_r = l * \Delta, l \in \mathbb{N}$, if it has been successfully deployed. The definition of TTL helps to multiplex a deployed SFC if the requests have the same traffic characteristics, resource demands, QoS requirements, which can further improve the resource utilization and reduce the service deployment cost. At a time slot τ, we use binary $a_{r,\tau}$ to indicate whether request $r \in R$ is still in service:

$$\forall r \in R : a_{r,\tau} = \begin{cases} 1, \tau_r^s \le \tau < (\tau_r^s + \tau_r), \\ 0, \text{otherwise.} \end{cases} \tag{1}$$

Since multiple service instances of a VNF can be deployed at the same node to deal with multiple requests, we use $n_{v,\tau}^f$ to indicate the number of service instances of VNF $f \in F$ that are deployed on node $v \in V$. Hence, we have:

$$\forall v \in V, f \in F : n_{v,\tau}^f = \sum_{r \in R} \sum_{1 \le i \le |r|}^{r(i)=f} x_{r,v}^i a_{r,\tau}, \tag{2}$$

where $r(i)$ represents the i-th VNF of $\widehat{r}$.

Besides, we use $z_{v,\tau}$ to indicate whether any VNF instances are placed on server node $v \in V$. We have:

$$\forall v \in V : z_{v,\tau} = \begin{cases} 1, \sum_{f \in F} n_{v,\tau}^{f} > 0, \\ 0, \sum_{f \in F} n_{v,\tau}^{f} = 0. \end{cases} \tag{3}$$

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 the 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.

State Definition. For each state $s_t \in \mathcal{S}$, we define it as a vector $(\widehat{C}_t, \widehat{W}_t, I_t)$. $\widehat{C}_t^s = (\widehat{C}_1^t, \widehat{C}_2^t, ..., \widehat{C}_{|V|}^t)$ represents the remaining resource of each node, while $\widehat{W}_t = (\widehat{W}_1^t, \widehat{W}_2^t, ..., \widehat{W}_{|V|}^t)$ represents the remaining output bandwidth. $I_t = (W_{r_i}, \widehat{T}_{r_i}^j, \widehat{N}_{r_i}^j, C_{f_{i,j}}, P_{r_i})$ reveals the characteristics of the current VNF being processed, $f_{i,j}$, where W_{r_i} is the bandwidth demand, $\widehat{N}_{r_i}^j$ is the number of undeployed VNFs in r_i, $\widehat{T}_{r_i}^j$ is the residual latency space (i.e., the response latency limitation T_r of r_i minus the current total latency of deployed VNFs), $C_{f_{j,i}}$ is the resource demand on servers and P_{r_i} is the TTL of the request r_i.

Action Definition. We first label each server node in the network with an integral index $k = 1, 2, ..., |V|$. Then we denote action $a \in \mathcal{A}$ as an integer, where $\mathcal{A} = \{0, 1, 2, ..., |V|\}$ is the set of server indexes. We use $a = 0$ to represent the case that VNF $f_{i,j}$ can not be deployed; otherwise, a denotes the specific index of server node in V, which means we have successfully place VNF $f_{i,j}$ on the a-th server node.

Reward Function. Since we want to jointly optimize two objectives considering both NFV providers' and customers' profits, we define the reward function as the weighted total accepted requests (income) minus the weighted total cost of occupied servers (expenditure) to deploy the arriving requests, which can combines the two objectives. The mathematical formulation of the reward function is given in Section IV-B, considering two different cases as the time slot moves on.

State Transition. An MDP state transition is defined as (s_t, a_t, r_t, s_{t+1}), where s_t is the current network state, a_t is the action taken for dealing with one of the service-required VNFs in request r_t and s_{t+1} is the new network state. The MDP state transition will also be elaborated in Section IV-B.

3.3 Problem Formulation

Now we present the mathematical formulation of the **SFC deployment problem**. We begin with the constraints and then the objectives together with some insights.

First, in fact, NFV providers can place multiple VNFs at the same server node if it has sufficient resource, which is known as VNF consolidation [28] as we mentioned previously. Thus, we state the resource constraint on servers in inequality (4).

$$\forall v \in V : \sum_{f \in F} n_{v,\tau}^{f} C_f \le C_v. \tag{4}$$

Second, we introduce the latency constraint. We use t_r to represent the total response latency of a request $r \in R$, which is the sum of the communication latency on links and the processing latency on server nodes. Specifically, we use t_f to represent the processing latency of a service instance of VNF $f \in F$. Thus, the total end-to-end latency of request $r \in R$ can be represented by:

$$\forall r \in R : t_r = \sum_{v \in V} \sum_{i=1}^{r(i)=f} x_{r,v}^{i} t_f + \sum_{u,v \in V} \sum_{i=1}^{|\widehat{r}|-1} x_{r,v}^{i} x_{r,v}^{i+1} T_{u,v}. \tag{5}$$

If a request $r \in R$ is accepted, its total response latency t_r can not exceed its response latency limitation T_r. Thus, we use a binary variable y_r to indicate whether r is accepted or not, which can be expressed as follows:

$$\forall r \in R : y_r = \begin{cases} 1, \sum_{i=1}^{|\widehat{r}|} \sum_{v \in V} x_{r,v}^{i} = |\widehat{r}| \text{ and } t_r \le T_r, \\ 0, \sum_{i=1}^{|\widehat{r}|} \sum_{v \in V} x_{r,v}^{i} < |\widehat{r}| \text{ or } t_r > T_r. \end{cases} \tag{6}$$

Third, we explore the bandwidth resource constraint. We use $u_{v,\tau}^{r}$ to indicate whether any VNFs of request $r \in R$ are placed at node $v \in V$, thus we have:

$$\forall v \in V, r \in R : u_{v,\tau}^{r} = \begin{cases} 1, \sum_{i=1}^{|\widehat{r}|} x_{r,v}^{i} a_{r,\tau} > 0, \\ 0, \sum_{i=1}^{|\widehat{r}|} x_{r,v}^{i} a_{r,\tau} = 0. \end{cases} \tag{7}$$

Since the bandwidth demand of all requests passing through server node $v \in V$ cannot exceed its total output bandwidth, we have:

$$\forall v \in V : \sum_{r \in R} y_r W_r u_{v,\tau}^{r} \le W_v. \tag{8}$$

In general, we want to achieve two objectives.

Objective 1 is to **minimize the operation cost of occupied servers**, which can be expressed as:

$$\begin{aligned} &\min \sum_{\tau=1} \sum_{v \in V} z_{v,\tau} (\xi_C C_v + \xi_W W_v), \\ &s.t. (1), (2), (3), (4), \end{aligned} \tag{9}$$

where ξ_C is the unit cost of server resource, while ξ_W is the unit cost of bandwidth. Both ξ_C and ξ_W are determined by the real NFV market and NFV service providers.

Insight: This objective benefits the NFV providers for reduction in the CAPEX and OPEX. In order to minimize the operation cost of occupied servers, the NFV providers have to fully utilize the remaining resource of occupied servers, and thus more idle servers can be shut down. However, simply focusing on **Objective 1** will be likely to increase the processing latency of requests as plotted previously in Fig. 1 (a). Moreover, the growth in communication latency may lead to a higher rejection rate of requests, which will further reduce some levels of QoS.

Objective 2 is to **maximize the total throughput of accepted requests**, which can be expressed as:

$$\max \sum_{r \in R} y_r W_r \tau_r, \quad s.t.\ (1), (5), (6), (7), (8). \tag{10}$$

Insight: This objective intuitively benefits the customers. To achieve **Objective 2**, the paramount consideration is to maximize the acceptance rate of requests. For instance, the NFV providers may consider consolidating some VNFs on the same server to reduce the communication latency of some latency-sensitive requests, which is shown in Fig. 1 (b). This benefits the customers with improved QoS, however, it uses more servers in service and leads to higher operation cost.

Note that an NFV system's profit is primarily obtained from accepting requests. To jointly optimize the two objectives, we define the total reward in the MDP model as the income of the total accepted requests minus the total cost of used servers to deploy the arriving requests. There still exist some trade-offs between different deployment decisions — whether to accept a request with strict QoS requirements by occupying more servers, or reject it but loss some profits.

4 A POLICY GRADIENT BASED DEEP REINFORCEMENT LEARNING APPROACH FOR ONLINE SFC DEPLOYMENT

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

4.1 Architecture of NFVdeep

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 which can automatically take appropriate actions in each state so as to achieve a high reward. Thus, we propose NFVdeep, an adaptive, online, PG based DRL approach to adaptively deploy SFCs with different QoS requirements.

The architecture of NFVdeep is illustrated in Fig. 2. Note that the NFV environment is the NFV network, including the servers and links in the network topology; with DRL, the NFVdeep agent is designed as a deep neural network (DNN) [8]. In particular, the NFVdeep agent gets the state information from the NFV environment and automatically selects an action as return. After the action is taken, the NFV environment transfers the reward to the agent. Finally, the agent updates related policies according to the reward. Repeat this procedure until the reward converges.

With PG, a policy $\pi|\mathcal{S} \times \mathcal{A}$ is defined as a multi-layer fullly-connected DNN Q^π based on the back propagation (BP) network, which performs better in large action space than DQN. It has an input layer, an output layer and several hidden layers. As shown in Fig. 3, the input layer is the state vector and the output layer is the actions' probability distribution.

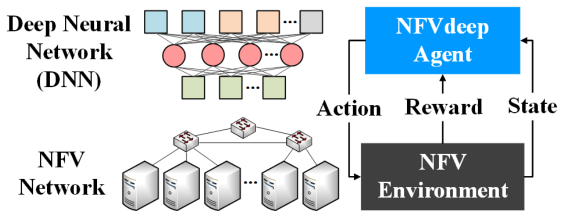

Figure 2: Architecture of NFVdeep.

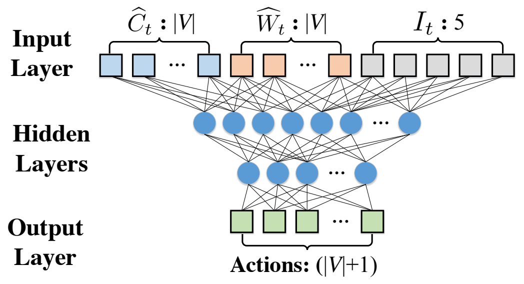

Figure 3: The PG based neural network design (with two hidden layers for example).

Table 2: Parameters of Hidden Layers

$\lvert V \rvert$	l_1	l_2	l_3	l_4
24	51	40 ($tanh$)		
50	97	83 ($tanh$)	68 ($tanh$)	
100	180 ($ReLU$)	151 ($tanh$)	127 ($tanh$)	
200	355 ($ReLU$)	307 ($tanh$)	272 ($tanh$)	240 ($tanh$)
500	875 ($ReLU$)	761 ($tanh$)	664 ($tanh$)	580 ($tanh$)

Now we elaborate on how to determine the number of hidden layers and the width of each hidden layer in DNN Q^π. We begin with one hidden layer h with S as its input layer and A as its output layer. According to [26], we adopt a useful empirical equation $|h| = \sqrt[2]{|S| \times |A|}$, $\alpha = 0, 1, \ldots, 10$ to decide the width of each layer. For a DNN with two hidden layers, the number of nodes of its first hidden layer is $|S|^{\frac{2}{3}}|A|^{\frac{1}{3}} + \alpha$, and the second is $|S|^{\frac{1}{3}}|A|^{\frac{2}{3}} + \alpha$. We find that when the number of nodes $|V|$ is less than 200, three hidden layers are enough. However, when $|V|$ is over 200, a DNN with four hidden layers performs better.

The detailed design of DNN's hidden layers is listed in Table. 2, where *ReLU* and *tanh* are two *activation functions* [26] added to related layers, which introduce non-linear features to the neural network and improve the DNN's capacity.

At last, we also use the *input data normalization* [14] method to format the input data into a small range (i.e., from -1 to 1), which makes the neural network easier to train. In order to keep the quantitative relations of input data such as the resource utilization, we devise a simple yet effective scaling approach, which compresses the related inputs with a constant. For the resource related inputs in state s, we divide it by a maximum $C_{max} = max(C_v), \forall v \in V$, including the remaining resource of each node $\widehat{C}_j^t, \forall j \in 1, 2, \ldots, |V|$ and the resource demand $C_{f_{j,i}}$ of current VNF $f_{j,i}$. We also compress the bandwidth related inputs and others in the same manner. In this way, the normalized input state is: $(\frac{\widehat{C}_1^t}{C_{max}}, \frac{\widehat{C}_2^t}{C_{max}}, \cdots, \frac{\widehat{C}_{|V|}^t}{C_{max}}, \frac{\widehat{W}_1^t}{W_{max}}, \frac{\widehat{W}_2^t}{W_{max}}, \cdots, \frac{\widehat{W}_{|V|}^t}{W_{max}}, \frac{W_{r_i}}{W_{max}}, \frac{\widehat{T}_{r_i}^j}{1000}, \frac{\widehat{N}_{r_i}^j}{100}, \frac{C_{f_{i,j}}}{C_{max}}, \frac{P_{r_i}}{1000})$.

4.2 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 NFV system firstly removes timeout requests, then successively processes all requests in R_τ, making a series of decisions about whether to reject or accept each SFC 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. NFVdeep serially deals with the VNFs of an SFC, and backtracks to the previous network state if an SFC cannot be completely deployed (i.e., some VNF(s) of the SFC cannot be placed due to the resource shortage, or the latency or bandwidth constraint of the request can not be satisfied). Fig. 4 illustrates how NFVdeep works as the time slot moves on.

As we can see, there are two cases between every two time slots: (1) **Intra time slot**, when there are several arriving requests in a time slot. NFVdeep sequentially deals with these arriving requests, specifically one VNF of an SFC after another. In this case, an MDP state transition happens when a VNF is deployed or rejected. (2) **Inter time slot**, when there is no request arriving during a number of continuous time slots. In this case, no action can be taken and the network state stays the same. Now we detailedly discuss each case in turn.

Intra time slot. As shown in Fig. 4, two requests arrive (i.e., SFC_1 and SFC_2) at time slot τ. The NFV system firstly removes the timeout requests and refreshes the network state by releasing the resources occupied by these requests. Then it processes these two arriving requests successively based on their arriving time. As we can see, SFC_1 has three VNFs to be placed, i.e., VNF_1, VNF_2 and VNF_3. At state s_t, the NFVdeep agent reads the network state and finds all candidate servers with sufficient resources to place VNF_1. Then, an action is taken to place VNF_1 on one of the candidate servers. The required resource is allocated and the server's index is recorded in the NFV system. Since SFC_1 is not completely deployed, no reward is returned to the NFVdeep agent. Then the system moves to state s_{t+1} with reward $U(s_t, a) = 0$.

At state s_{t+1}, since there is no server node having sufficient resource to place VNF_2, or the latency or bandwidth constraints can not be satisfied, no action will be taken and SFC_1 is rejected. The NFV system "backtracks" to the state before s_t and takes back the resource occupied by VNF_1. It then deals with SFC_2 in the same manner. As a result, SFC_2 is successfully deployed at state s_{t+3} with a reward, i.e., the throughput of SFC_2 minus its resource consumption cost, $U(s_{t+3}, a) = W_{r_2}P_{r_2} - Cost(s_{t+3}, a)$, where the resource consumption cost $Cost(s_{t+3}, a)$ is defined as Eq. (9). Finally, the system moves to a new state as the time slot moves to $\tau + 1$.

Inter time slot. As shown in Fig. 4, at time slot τ, the NFV system has successfully deployed the last VNF of the last request in R_τ at state s_{t+2}. In the next k ($k \in \mathbb{N}_+$) time slots, there is no request arriving (i.e., $R_p = \emptyset$). Then at every two time slots (i.e.,

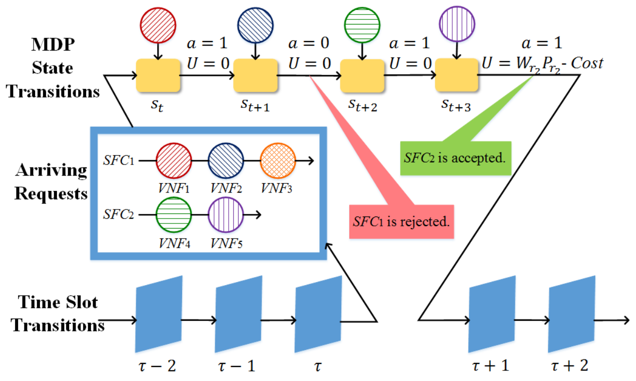

Figure 4: The procedure of NFVdeep as the time slot moves on.

Algorithm 1 The NFVdeep Procedure

```
1: Begin: Initiate time slot $\tau \leftarrow 1$
2: while $R_\tau = \emptyset$ do
3:     $\tau \leftarrow \tau + 1$
4: end while
5: Select a request $r_1$ from $R_\tau$ based on the arriving time
6: Initiate $i \leftarrow 1, j \leftarrow 1$
7: for $t \leftarrow 1, \mathbb{T}$ do
8:     Initiate state $s_t$
9:     Take action $a$ from $\mathcal{A}$ to place $f_{i,j}$
10:    if $f_{i,j}$ is accepted then
11:        $j \leftarrow j + 1, s_t \leftarrow s_{t+1}$
12:    end if
13:    if $r_i$ is rejected or $j > |\widehat{r}_i|$ then
14:        if $r_i$ is rejected and $j > 1$ then
15:            Backtrack the network state to $s_{t-j+1}$
16:        end if
17:        if $R_\tau$ is all processed then
18:            repeat
19:                $\tau \leftarrow \tau + 1$
20:            until $R_\tau \neq \emptyset$
21:            Select a new request $r'_1$ from $R_\tau$
22:            Reset $i \leftarrow 1, j \leftarrow 1$
23:        else
24:            Select request $r_{i+1}$ from $R_\tau$
25:            $i \leftarrow i + 1$
26:        end if
27:    end if
28:    Calculate the reward $U(s_t, a)$
29: end for
```

$\tau + k$ and $\tau + k + 1$), the NFV system only executes the following procedures: removing timeout requests and releasing the resources they occupied. Since there is no action to be taken, no state transition happens. Until the time slot when new requests arrive, the state transition happens and the reward is calculated when one arriving request is accepted or rejected. As expressed in Eq. (11), the reward function includes the state transitions across a series of time slots.

$$U(s_t, a) = \begin{cases} -\sigma_t Cost(s_t, a) + W_{r_i}P_{r_i} & ,r_i \text{ is accepted,} \\ 0 & ,r_i \text{ is rejected or} \\ & \text{ not fully deployed,} \end{cases} \tag{11}$$

where σ_t is a binary variable indicating whether the time slot changes after state s_t, and $Cost(s_t, a)$ is the total cost as defined in Eq. (9). To take the effect of the future decision's reward into account, we define the reward function at state s_t as:

$$U_{s_t} = \sum_{i=0}^{\infty} \gamma^i U(s_{t+i}, a), \tag{12}$$

where $\gamma \in [0, 1]$ is the discount factor for future reward.

The whole procedure of NFVdeep is listed in Algorithm 1. With the *serialization-and-backtracking* method, the NFV system can adaptively deploy SFCs for various requests with different QoS requirements.

4.3 Policy Gradient Based Training Procedure

We adopt the PG based method to directly optimize the quality of SFC deployment by the gradient computed from rollout estimates. With PG, our target is to find a policy to maximize the final reward after a sequence of state transitions. The policy $\pi_\theta(s, a) = \mathcal{P}(a|s, \theta)$ is parameterized with θ, which represents action a's probability at state s under parameter θ. An episode of training consists of a sequence of MDP state transitions. During each episode, all the state transitions are successively stored in a buffer and used for training until this episode ends. The objective function is defined as:

$$\mathcal{J}(\theta) = \sum_t \pi_\theta(s, a) R(t), \tag{13}$$

which is the final reward of an episode.

The policy gradient is defined as:

$$\nabla_\theta \mathcal{J}(\theta) = \left(\frac{\partial \mathcal{J}(\theta)}{\partial \theta_1}, ..., \frac{\partial \mathcal{J}(\theta)}{\partial \theta_n} \right), \tag{14}$$

which is the gradient descent of the parameter θ, where θ is updated as:

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta_i} \mathcal{J}(\theta_i), \tag{15}$$

where α is the learning rate, which can be adjusted based on the convergence speed in the training procedure. PG algorithm aims to maximize $\mathcal{J}(\theta)$ by ascending the gradient with respect to the product of the log sampling possibility and the accumulated reward associated with selected actions. Thus, policies with high-reward actions get encouraged, whereas policies with low-reward actions get discouraged in the future.

The PG based training procedure is listed in Algorithm 2. In each episode, we initialize the NFV system and in each MDP state transition NFVdeep processes one VNF of an SFC. When an episode comes to an end, the total reward U_t of each state s_t is calculated and transmitted to the NFVdeep agent. The NFVdeep agent is trained through one episode after another until the reward converges. Specifically, since the reward does not descend as the time slot moves on, we set the discount factor $\gamma = 1$. We also add a noise mechanism to avoid trapping into the local optimum and improve the training effect, which is a probability $\epsilon \in (0, 1)$ to choose a random action a_t, otherwise choose $a_t = \mathcal{P}(a|s, \theta)$.

Algorithm 2 PG Based Training Procedure

```
1:  Begin: Build and initialize PG neural network $Q^{\pi_\theta}$
2:  for $episode \leftarrow 1, M$ do
3:      Initialize the NFV environment and let $s \leftarrow s_1$
4:      for $t \leftarrow 1, T$ do
5:          Select an action $a$ randomly with the probability $\epsilon$, otherwise
        select action $a$ according to policy $\pi_\theta = \mathcal{P}(a|s, \theta)$
6:          Take action $a$ and calculate reward $U(s_t, a)$
7:          Transfer the state to $s_{t+1}$ and get the next VNF $f_{t+1}$
8:          Store transition $(s_t, a, U(s_t, a))$ to PG batch memory
9:      end for
10:     for $t \leftarrow 1, T$ do
11:         $U_t \leftarrow \sum_{q=1}^{t} \gamma^{t-q} U(s_q, a)$
12:     end for
13:     for $i \leftarrow 1, 10$ do
14:         for $t \leftarrow 1, T$ do
15:             $\theta = \theta + \sum_t \alpha \nabla \sum_{i=1}^{t} \pi_\theta(s_i, t_i) U(t_i)$
16:         end for
17:     end for
18: end for
```

5 PERFORMANCE EVALUATION

5.1 Simulation Setup

Network topology: We simulate a conventional NFV network topology based on a fat-tree architecture [35]. In this topology, all servers are connected via three layers of switches. We scale the number of server nodes $|V|$ from 24 to 500, each with [1, 500] units of CPU resource and [1, 64] units of memory. The output bandwidth of each server depends on its type and number of NICs, ranging from 100 Mbps to 100 Gbps. Finally, we set the intra-pod delay ranging from 40 to 100 μs and the inter-pod delay ranging from 50 to 200 μs.

SFC of requests: The requests we simulate are based on real-world trace in [3]. According to [1], 30 different commonly-deployed VNFs are simulated for composing SFCs, including 6 typical VNFs (i.e., firewall, NAT, IDS, load balancer, WAN optimizer and flow monitor) and other service-customized VNFs. We simulate from 24 to 1,000 requests and each request requires an SFC consisted of 1 to 7 VNFs. Different requests have different QoS requirements in terms of latency and throughput. We assume that each service instance of VNF serves a request independently with a fixed service rate μ ranging from 100 to 1,000. We simulate from 1,000 to 60,000 time slots in each episode. For each request, we assume that its packet arrival rate ranges from 1 to 100 packets/s. For the cost defined in Eq. (9), we set $\xi_C = 0.2$ and $\xi_W = 6.0 \times 10^{-4}$.

Baseline and schemes compared: We first use a fast and effective algorithm as the baseline, the non-recursive greedy SFC placement (NGSP) algorithm, which is reduced from the recursive greedy SFC placement (RGSP) algorithm in [4]. NGSP preferentially deploys VNFs of SFCs at used nodes with high resource utilization rate and can get a feasible result with $O(m \times n)$ time complexity as compared with RGSP's $O(m^n)$. To evaluate the performance of NFVdeep, we compare it with GPLL (a greedy-based policy to find path with the lowest latency) [33] and Bayes (a Bayesian learning based approach) [31]. GPLL aims at finding a path for each SFC with the lowest end-to-end latency. Since GPLL is an offline algorithm, we divide the whole request set into different R_τ as the time slot τ moves on. Bayes method adopts Bayesian learning approach to solve NFV components prediction, partition and deployment. For intuitively showing the experiment results, we normalize other results as divided by NGSP in all figures (i.e., from Fig. 5 to Fig. 12).

Simulation platform: We use a Python-based framework, TensorFlow to construct the architecture of NFVdeep and its deep neural network. All experiments are conducted on a Dell machine

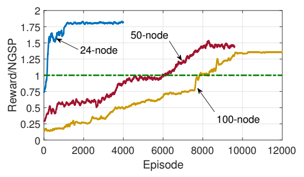

Figure 5: The average total rewards with different number of nodes.

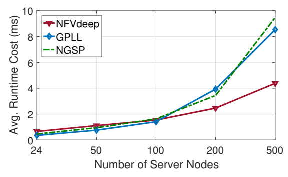

Figure 6: The average runtime cost of different schemes with different number of nodes.

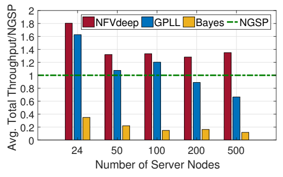

Figure 9: The average total throughput with different number of nodes.

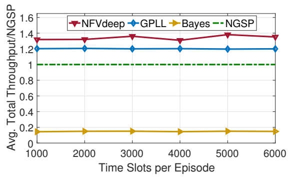

Figure 10: The average total throughput with different time slots each episode.

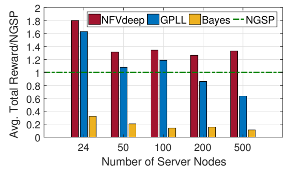

Figure 7: The average total reward with different number of nodes.

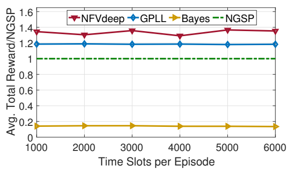

Figure 8: The average total reward with different time slots.

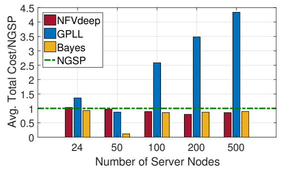

Figure 11: The average total operation cost of different schemes.

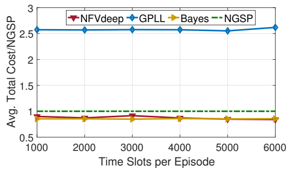

Figure 12: The average total operation cost with different time slots.

learning workstation, in which the CPU is Intel(R) Core(TM) i7-6850K with 6 cores 12 threads.

5.2 Performance Evaluation Results

Training efficiency: To show the training efficiency, we train NFVdeep in different network scales with the different number of server nodes and links. Fig. 5 plots the training procedure for 24-, 50- and 100-node network topologies. In order to accelerate the training speed and achieve better training effect, we adjust various parameters such as learning rate λ and the number of time slots per episode. As a result, the reward converges at about 1,600 episodes with 24 nodes, and at about 8,500 and 9,800 episodes with 50 and 100 nodes respectively. Consequently, NFVdeep converges fast in the training procedure; as the number of server nodes grows, NFVdeep takes more episodes to converge.

Runtime cost: To illustrate the runtime cost of each scheme, we compare the average runtime of NFVdeep in a time slot with other methods as the number of server nodes $|V|$ ranging from 24 to 500. In particular, we build a local TensorFlow to support full CPU utilization. As shown in Fig. 6, the runtime of the three schemes are nearly the same when the number of server nodes is no more than 100. As the number of server nodes increases, NFVdeep always achieves the lowest runtime within 5 ms. In summary, NFVdeep can respond rapidly to arriving requests and is more time-efficient than other approaches in large network scales.

Total reward: We explore the average total reward of NFVdeep as compared with the state-of-the-art methods. Fig. 7 plots the results with 2,000 time slots each episode as $|V|$ ranges from 24 to 500. As a result, NFVdeep is the only method whose total reward is always above the baseline, which achieves averagely 33.29% more rewards than GPLL. Fig. 8 also shows that NFVdeep always achieves the highest reward in a 100-node network as time slots moves from 1,000 to 6,000 in an episode, whose rewards are averagely 18.68% more than GPLL. In summary, NFVdeep always achieves the maximal reward as compared with other methods.

Total throughput: We compare the total throughput of accepted requests of NFVdeep with other methods. Fig. 9 plots the results with 2,000 time slots each episode and Fig. 10 plots the results for a 100-node network with time slots ranging from 1,000 to 6,000 in an episode. As shown in Fig. 9, NFVdeep's total throughput is always above the baseline as $|V|$ ranges from 24 to 500. The trend of this performance metric is similar as the total reward, with 32.59% improvement as compared with GPLL. Fig. 10 also shows that NFVdeep always achieves the most total throughput as the time slot moves on. In conclusion, NFVdeep achieves up to 32.59% more total throughput than other methods with different numbers of servers or in a long range of time slots.

Operation cost: Finally, the total operation costs of occupied servers are plotted in Fig. 11 and Fig. 12. Fig. 11 plots the results with 2,000 time slots each episode as $|V|$ ranges from 24 to 500, and Fig. 12 plots the results for a 100-node network with time slots ranging from 1,000 to 6,000 in an episode. Fig. 11 shows that the operation cost of Bayes is lower than NGSP when the number of servers is above 50, while GPLL's operation cost grows significantly as the number of server nodes increases. NFVdeep's operation cost is a little bit higher than the baseline with 24 server nodes, but it is always lower than the baseline as the number of server nodes scales from 50 to 500. Fig. 12 illustrates the same results as the time slot moves on. The operation costs of NFVdeep and Bayes are at the same level, which are both less than the baseline, while GPLL is averagely 2.6× of the baseline. Compared with other state-of-the-art methods, NFVdeep achieves the highest profits and reduces averagely 9.62% operation cost.

6 CONCLUSION

In this paper, we study the online SFC deployment problem in NFV systems. We firstly introduce a Markov decision process model to capture the dynamic network state transitions caused by stochastically arriving requests. In order to jointly minimize the operation cost of NFV providers and maximize the total throughput of requests, we propose NFVdeep, an adaptive, online, deep reinforcement learning approach to automatically deploy SFCs in large, frequently transferred network state space. Specifically, we use a serialization-and-backtracking method to deal with the large discrete action space and a policy gradient based method to improve the training efficiency and convergence to optimality. Extensive experimental results demonstrate that NFVdeep converges fast with different network scales and can respond within 5 ms to arriving requests with different QoS requirements. As compared with the state-of-the-art approaches, NFVdeep always achieves the highest total rewards with 32.59% higher total throughput of accepted requests and 33.29% lower operation cost on average.

REFERENCES

[1] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. 2015. Virtual network functions placement and routing optimization. In *2015 IEEE 4th CloudNet*. IEEE, 171–177. https://doi.org/10.1109/CloudNet.2015.7335301

[2] Satyam Agarwal, Francesco Malandrino, Carla-Fabiana Chiasserini, and Swedes De. 2018. Joint VNF placement and CPU allocation in 5G. In *IEEE INFOCOM 2018*. IEEE, 1943–1951. https://doi.org/10.1109/infocom.2018.8485943

[3] Alibaba. 2019. ClusterData201708 in Alibaba. https://github.com/alibaba/clusterdata[Online Accessed, 5-May-2019].

[4] Rumen Andonov, Vincent Poirriez, and Sanjay Rajopadhye. 2000. Unbounded knapsack problem: Dynamic programming revisited. *European Journal of Operational Research* 123, 2 (2000), 394–407. https://doi.org/10.1016/s0377-2217(99)00265-9

[5] Yi Bo, Xingwei Wang, Keqin Li, Sajal K. Das, and Huang Min. 2018. A comprehensive survey of Network Function Virtualization. *Computer Networks* 133 (2018), 212–262. https://doi.org/10.1016/j.comnet.2018.01.021

[6] Jiuyue Cao, Yan Zhang, Wei An, Xin Chen, Jiyan Sun, and Yanni Han. 2017. VNF-FG design and VNF placement for 5G mobile networks. *Science China Information Sciences* 60, 4 (2017), 040302. https://doi.org/10.1007/s11432-016-9031-x

[7] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. 2015. Near optimal placement of virtual network functions. In *IEEE INFOCOM 2015*. IEEE, 1346–1354. https://doi.org/10.1109/INFOCOM.2015.7218511

[8] Li Deng, Geoffrey Hinton, and Brian Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 8599–8603. https://doi.org/10.1109/ICASSP.2013.6639344

[9] Mouhamad Dieye, Shohreh Ahvar, Jagruti Sahoo, et al. 2018. CPVNF: Cost-efficient Proactive VNF Placement and Chaining for Value-Added Services in Content Delivery Networks. *IEEE Transactions on Network and Service Management* 15, 2 (2018), 774–786. https://doi.org/10.1109/TNSM.2018.2815986

[10] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca. 2017. An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures. *IEEE/ACM Transactions on Networking* PP, 99 (2017), 1–18. https://doi.org/10.1109/TNET.2017.2668470

[11] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. 2017. Towards load-balanced VNF assignment in geo-distributed NFV infrastructure. In *2017 IEEE/ACM 25th IWQoS*. IEEE, 1–10. https://doi.org/10.1109/IWQoS.2017.7969166

[12] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. 2018. Adaptive vnf scaling and flow routing with proactive demand prediction. In *IEEE INFOCOM 2018*. IEEE, 486–494. https://doi.org/10.1109/INFOCOM.2018.8486320

[13] Ed. Halpern, J. et al. 2015. *Service Function Chaining (SFC) Architecture*. http://www.rfc-editor.org/info/rfc7665[Online Accessed, 5-May-2019].

[14] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* abs/1502.03167 (2015). arXiv:1502.03167 http://arxiv.org/abs/1502.03167

[15] Selma Khebbache, Makhlouf Hadji, and Djamal Zeghlache. 2017. Scalable and cost-efficient algorithms for VNF chaining and placement problem. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. IEEE, 92–99. https://doi.org/10.1109/ICIN.2017.7899395

[16] Abdelquoddouss Laghrissi et al. 2017. Towards edge slicing: VNF placement algorithms for a dynamic & realistic edge cloud environment. In *IEEE GLOBECOM 2017*. IEEE, 1–6. https://doi.org/10.1109/GLOCOM.2017.8254653

[17] Xiaoyao Li, Xiuxiu Wang, Fangming Liu, and Hong Xu. 2018. DHL: Enabling Flexible Software Network Functions with FPGA Acceleration. In *2018 IEEE 38th ICDCS*. IEEE, 1–11. https://doi.org/10.1109/ICDCS.2018.00011

[18] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[19] Yang Li, Linh Thi Xuan Phan, and Boon Thau Loo. 2016. Network functions virtualization with soft real-time guarantees. In *IEEE INFOCOM 2016*. IEEE, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524563

[20] Tachun Lin, Zhili Zhou, et al. 2016. Demand-Aware Network Function Placement. *Journal of Lightwave Technology* 34, 11 (2016), 2590–2600. https://doi.org/10.1109/JLT.2016.2535401

[21] Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. 2016. Vnf placement and chaining in distributed cloud. In *2016 IEEE 9th CLOUD*. IEEE, 376–383. https://doi.org/10.1109/CLOUD.2016.0057

[22] Sevil Mehraghdam, Matthias Keller, and Holger Karl. 2014. Specifying and placing chains of virtual network functions. In *2014 IEEE 3rd CloudNet*. IEEE, 7–13. https://doi.org/10.1109/CloudNet.2014.6968961

[23] Rashid Mijumbi, Joan Serrat, et al. 2016. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys and Tutorials* 18, 1 (2016), 236–262. https://doi.org/10.1109/COMST.2015.2477041

[24] Volodymyr Mnih, Koray Kavukcuoglu, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236

[25] Hendrik Moens and Filip De Turck. 2014. VNF-P: A model for efficient placement of virtualized network functions. In *IEEE 10th CNSM 2014*. https://doi.org/10.1109/CNSM.2014.7014205

[26] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *27th ICML 2010*. 807–814. https://doi.org/doi:http://dx.doi.org/

[27] Leonardo Ochoa-Aday, Cristina Cervelló-Pastor, Adriana Fernández, and Paola Grosso. 2018. An Online Algorithm for Dynamic NFV Placement in Cloud-Based Autonomous Response Networks. *Symmetry* 10, 5 (2018), 163. https://doi.org/10.3390/sym10050163

[28] Aurojit Panda, Keon Jang, Keon Jang, Melvin Walls, et al. 2016. NetBricks: taking the V out of NFV. In *12th Usenix OSDI 2016*. 203–216.

[29] Satinder Singh Yishay Mansour Richard S.Sutton, David McAllester. 2012. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems* 7 (2012), 1057–1063.

[30] Yu Sang, Bo Ji, et al. 2017. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *IEEE INFOCOM 2017*. https://doi.org/10.1109/INFOCOM.2017.8057036

[31] R. Shi, J. Zhang, et al. 2015. MDP and Machine Learning-Based Cost-Optimization of Dynamic Resource Allocation for Network Function Virtualization. In *IEEE SCC*. 65–73. https://doi.org/10.1109/SCC.2015.19

[32] David Silver, Aja Huang, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489. https://doi.org/10.1038/nature16961

[33] Chen Sun, Jun Bi, Zhilong Zheng, and Hongxin Hu. 2016. SLA-NFV: an SLA-aware High Performance Framework for Network Function Virtualization. In *ACM SIGCOMM 2016*. 581–582. https://doi.org/10.1145/2934872.2959058

[34] Quanying Sun, Ping Lu, Wei Lu, and Zuqing Zhu. 2016. Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement. In *IEEE GLOBECOM 2016*. 1–6. https://doi.org/10.1109/GLOCOM.2016.7841846

[35] Yantao Sun, Jing Chen, et al. 2014. Diamond: An Improved Fat-tree Architecture for Large-scale Data Centers. *Journal of Communications* 9, 1 (2014), 91–98.

[36] Tarik Taleb, Badr Mada, et al. 2017. PERMIT: Network Slicing for Personalized 5G Mobile Telecommunications. *IEEE Communications Magazine* 55, 5 (2017), 88–93. https://doi.org/10.1109/MCOM.2017.1600947

[37] Tao Wang, Hong Xu, and Fangming Liu. 2017. Multi-resource load balancing for virtual network functions. In *IEEE ICDCS 2017*. https://doi.org/10.1109/ICDCS.2017.233

[38] Ming Xia, Meral Shirazipour, Ying Zhang, Howard Green, and Attila Takacs. 2015. Network function placement for NFV chaining in packet/optical datacenters. *Journal of Lightwave Technology* 33, 8 (2015), 1565–1570. https://doi.org/10.1109/JLT.2015.2388585

[39] Zhifeng Xu, Fangming Liu, Tao Wang, and Hong Xu. 2016. Demystifying the energy efficiency of network function virtualization. In *2016 IEEE/ACM 24th IWQoS*. https://doi.org/10.1109/IWQoS.2016.7590429

[40] Min Sang Yoon and Ahmed E Kamal. 2016. NFV Resource Allocation using Mixed Queuing Network Model. In *IEEE GLOBECOM 2016*. 1–6. https://doi.org/10.1109/GLOCOM.2016.7842023

[41] Chaobing Zeng, Fangming Liu, Shutong Chen, Weixiang Jiang, and Miao Li. 2018. Demystifying the performance interference of co-located virtual network functions. In *IEEE INFOCOM 2018*. 765–773. https://doi.org/10.1109/infocom.2018.8486246

[42] Qixia Zhang, Fangming Liu, and Chaobing Zeng. 2019. Adaptive Interference-Aware VNF Placement for Service-Customized 5G Network Slices. In *IEEE INFOCOM 2019*.

[43] Qixia Zhang, Yikai Xiao, Fangming Liu, John CS Lui, Jian Guo, and Tao Wang. 2017. Joint optimization of chain placement and request scheduling for network function virtualization. In *IEEE ICDCS 2017*. 731–741. https://doi.org/10.1109/ICDCS.2017.232

[44] Sai Qian Zhang, Ali Tizghadam, Byungchul Park, Hadi Bannazadeh, and Alberto Leon-Garcia. 2016. Joint NFV placement and routing for multicast service on SDN. In *IEEE/IFIP NOMS 2016*. 333–341. https://doi.org/10.1109/NOMS.2016.7502829