This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

Optimal VNF Placement via Deep Reinforcement Learning in SDN/NFV-Enabled Networks

Jianing Pei, *Student Member, IEEE*, Peilin Hong, *Member, IEEE*, Miao Pan, *Senior Member, IEEE*, Jianqing Liu, *Member, IEEE*, Jingsong Zhou, *Student Member, IEEE*

***Abstract*—The emerging paradigm - Software-Defined Networking (SDN) and Network Function Virtualization (NFV) - makes it feasible and scalable to run Virtual Network Functions (VNFs) in commercial-off-the-shelf devices, which provides a variety of network services with reduced cost. Benefitting from centralized network management, lots of information about network devices, traffic and resources can be collected in SDN/NFV-enabled networks. Using powerful machine learning tools, algorithms can be designed in a customized way according to the collected information to efficiently optimize network performance. In this paper, we study the VNF placement problem in SDN/NFV-enabled networks, which is naturally formulated as a Binary Integer Programming (BIP) problem. Using deep reinforcement learning, we propose a Double Deep Q Network-based VNF Placement Algorithm (DDQN-VNFPA). Specifically, DDQN determines the optimal solution from a prohibitively large solution space and DDQN-VNFPA then places/releases VNF Instances (VNFIs) following a threshold-based policy. We evaluate DDQN-VNFPA with trace-driven simulations on a real-world network topology. Evaluation results show that DDQN-VNFPA can get improved network performance in terms of the reject number and reject ratio of Service Function Chain Requests (SFCRs), throughput, end-to-end delay, VNFI running time and load balancing compared with the algorithms in existing literatures.**

***Index Terms*—Software-Defined Networking, Network Function Virtualization, VNF Placement, Deep Reinforcement Learning.**

I. Introduction

TRADITIONALLY, middleboxes are placed in the network to provide services for users. As middleboxes are generally implemented in dedicated hardwares, the placement of middleboxes is inflexible and always incurs high Capital Expenses (CAPEX) and Operating Expenses (OPEX) for Internet Service Providers (ISPs) during purchase, management and maintenance [1], [2]. Fortunately, Network Function Virtualization (NFV) and Software-Defined Networking (SDN) appear as a viable way to solve these problems. Using NFV, Virtual Network Functions (VNFs) can be realized in software and placed in commercial-off-the-shelf devices adaptively. Thus, middleboxes, such as Firewall (FW), Deep Package Inspection (DPI) and Intrusion Detection System (IDS), can be replaced by VNFs, which greatly enhances the network flexibility and scalability and reduces CAPEX/OPEX resulting from middleboxes [3]. As a new networking paradigm, SDN decouples control plane and data plane, and achieves central network management with SDN controllers [4]–[6]. Given the advantages above, in SDN/NFV-enabled networks, it is convenient for ISPs to monitor network devices (*e.g.,* routers, switches, *etc.*) and traffic, thus making it efficient to manage VNF Instances (VNFIs).

Though VNFIs can be flexibly placed in the network, there exist a few challenging problems. In SDN/NFV-enabled networks, Service Function Chains (SFCs) have become a popular networking service paradigm. According to the standardization of SFC by Internet Engineering Task Force (IETF), SFC defines a set of ordered or partially ordered VNFIs and ordering constraints that must be applied to packets, frames and/or flows selected as a result of classification [7], [8]. Noting that the traffic of an SFC often requests to be steered to traverse a series of specific VNFIs in a predefined order, we define such request as SFC Request (SFCR) in this paper. As for an SFCR, ingress $\rightarrow$ FW $\rightarrow$ DPI $\rightarrow$ IDS $\rightarrow$ egress, its traffic needs to be steered to traverse the instances of FW, DPI and IDS in order before reaching the egress node. However, each type of VNF always has many instances placed in different network locations, so it is an important problem how to select the optimal VNFIs and construct the routing path to steer the traffic of an SFCR. Moreover, in real network scenarios, network traffic changes drastically over time and places, so the placement of VNFIs should be dynamically adjusted to adapt to the change of network load [9]–[11]. For example, when network load increases, it is necessary to place more VNFIs to provide more available resources for users; and when network load decreases, it is helpful to release redundant placed VNFIs to save resources and reduce energy consumption. Nevertheless, a series of works prove that the VNF placement problem is NP-hard [12]–[14], which is nontrivial to address. Therefore, how to determine optimal placement of VNFIs is a critical yet challenging problem to study in SDN/NFV-enabled networks.

Today, Deep Reinforcement Learning (DRL) has made significant breakthroughs and impacted various domains like control theory and strategic game playing [15], [16], which is expected to enlighten the solution to our VNF placement problem as well. Generally, traditional Reinforcement Learning (RL) algorithms

This work of J. Pei, P. Hong, and J. Zhou was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61671420. The work of M. Pan was supported in part by the U.S. National Science Foundation under grants CNS-1350230, CNS-1646607, CNS-1702850, and CNS-1801925. (Corresponding Author: Peilin Hong)

J. Pei, P. Hong, and J. Zhou are with the department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, China. (jianingp@mail.ustc.edu.cn, plhong@ustc.edu.cn, zjskd@mail.ustc.edu.cn)

M. Pan is with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77004, USA. (mpan2@uh.edu)

J. Liu is with the Department of Electrical and Computer Engineering, University of Alabama in Huntsville, Huntsville, AL 35899, USA. (jianqing.liu@uah.edu)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

are tabular methods that evaluate the performance of an action under a state with a Q table. However, since the dimension of Q table is finite, traditional RL methods are limited to handle low-dimension problems with discrete states and actions. Compared with traditional RL methods, in DRL, deep neural networks are used to replace Q table. As deep neural networks can build the relationship among high-dimensional states, actions and Q-values, DRL has powerful learning capacity to be widely adopted to solve complex problems.

In SDN/NFV-enabled networks, it is difficult for traditional RL methods to solve the VNF placement problem because of the following challenges. Firstly, a network could include plenty of hardware devices, and the change of network resource state (*e.g.,* bandwidth, memory, CPU, *etc*) is complex resulting from varieties of network services and requirements of users and has influence on the placement of VNFIs. Moreover, in NFV environment, VNFIs can be flexibly placed in various network locations, which results in a large size of action space of the VNF placement and optimization. Benefitting from deep neural networks, DRL is capable to deal with such high-dimension network resource states and VNF placement actions in SDN/NFV-enabled environment. With the exploration of new knowledge and the exploitation of the acquired one, DRL can effectively evaluate the performance of VNF placement actions under network resource states, and learn to adjust and conduct better actions to optimize the placement of VNFIs given the feedback rewards from networks.

In this paper, we study the VNF placement problem in SDN/NFV-enabled networks. First, we formulate this problem as a Binary Integer Programming (BIP) model aiming to minimize a weighted cost including the VNF placement cost, VNFI running cost and penalty of reject SFCRs. The VNF placement and VNFI running costs are related to Operating Expenses (OPEX), Capital Expenses (CAPEX) and energy consumption in management, monitoring and maintenance of VNFIs. The penalty of reject SFCRs has a big impact on network performance and the usage of resources (*e.g.,* bandwidth, memory, CPU, *etc*). So ISPs can enhance their revenue and network performance by optimizing the above three costs. Next, a Double Deep Q Network-based VNF Placement Algorithm (DDQN-VNFPA) is proposed to intelligently and efficiently solve the problem. In order to design a time-efficiency approach, the latencies from VNF placement and resource provision for an SFC must be taken into account. Many papers have shown that initializing a VNFI can need tens of seconds and it always needs serval minutes to completely setup an SFC for a user, which could influence the Quality-of-Service (QoS) in the network and harm the Quality-of-Experience (QoE) of users [17], [18]. Fortunately, the forecasting technique has been widely studied and used to forecast the change of traffic in SDN/NFV-enabled networks [18]–[20]. In our paper, we assume that the SFCRs of a future time interval can be forecasted, then we can avoid the setup latency of VNFIs and SFCRs by pre-placing VNFIs according to forecasting results.

Our proposed DDQN-VNFPA consists of offline training process and online running process. In training process, we collect training data and train DDQN models offline. After training process, we run those trained DDQN models to output the optimization strategies of VNF placement according to network resource states. The running process conducts in three phases: i) use DDQN models to preliminarily evaluate VNF placement actions and optimize the solution space size; ii) conduct actions of optimized solution space considering forecasted SFCRs in simulation environment to get rewards, and record results into dataset to further update DDQN models; iii) choose the optimal action with the highest reward to optimize the placement of VNFIs with a threshold-based policy.

Generally, there are two kinds of schemes to solve the VNF placement problems [21], [22]. The first kind is horizontal scheme, which can adjust the number of VNFIs placed in the network, while the available resource per VNFI is fixed; the second kind is vertical scheme, which can optimize the available resource per VNFI, while do not adjust the number of VNFIs placed in the network. In this paper, we consider the horizontal way to solve the VNF placement problem.

The contributions of this paper are listed as follows:

- We give a detailed analysis of the VNF placement problem considering dynamic change of network load in SDN/NFV-enabled networks, and show the advantages by solving the problem using DRL.
- We formulate the VNF placement problem as a BIP model aiming to minimize a weighted cost where the VNF placement cost, VNFI running cost and penalty of reject SFCRs are all taken into account. Then, a novel horizontal scheme, DDQN-VNFPA, is proposed to solve the problem more efficiently.
- We conduct a theoretical analysis on the effectiveness of DDQN-VNFPA and further construct a Tensorflow-based environment to evaluate its performance. Simulation results show that, compared with existing algorithms, our DDQN-VNFPA can get high performance in terms of SFCR reject number and ratio, throughput and end-to-end delay of SFCRs, save VNFI running time, improve VNF utilization ratio and better balance network load.

The rest of the paper is organized as follows: we review related works and introduce the technical background about DDQN in Section II. In Section III, we present the system model, and formulate the VNF placement problem in Section IV. Then we propose our DDQN-VNFPA algorithm and analyze its effectiveness in Section V. The performance of DDQN-VNFPA is compared with existing algorithms in Section VI. Finally, Section VII concludes the paper.

II. Related Works and Technical Background

A. Related Works

Recently, the design of VNF placement has become a hot issue in academia, and many solutions have been proposed. Generally, most of existing literatures formulate it into Integer Linear Programming (ILP) models [12], [23]–[25], Binary Integer Programming (BIP) models [9], [13], [26] or Mixed Integer Linear Programming (MILP) models [19], [27], [28]. However, since the VNF placement problem is NP-hard, it is difficult to obtain the optimal solutions efficiently, especially in large scale networks. For example, Li *et al.* [23], [24] showed that, in the worst situation, it leaded to about four orders of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

magnitude time consumption more than heuristics to solve their models using existing optimization toolboxes.

Therefore, as for large scale networks, particularly in the environments of cloud computing [29], [30] and fog computing [31], [32], many researchers have proposed heuristics to solve the VNF placement problem for SFCRs. For example, Li *et al.* studied the VNF placement problem in cloud datacenter [24] and edge computing enabled networks [23], where the objective was to minimize used physical machines and the resource consumptions in nodes and links. Then, considering basic resource consumptions of VNFs, two polynomial-time heuristics are proposed to achieve the placement of VNFIs in large scale networks. In datacenter networks, Bari *et al.* [12] solved the VNF orchestration problem with a multi-stage graph, and Qi *et al.* [25] improved their algorithm using accessible scope which could achieve VNF placement efficiently. Considering traffic forecasting and the VNF placement problem in operator datacenters, first, Tang *et al.* [19] proposed a traffic forecasting method based on slip-window linear regression, then developed two heuristics via relaxing integer variables. In our previous work [9], we proposed a layered graph-based algorithm to solve dynamic VNF placement in geo-distributed cloud system, where the objective was to achieve load balancing and minimize the placement cost of VNFIs.

Hawilo *et al.* [27], Mechtri *et al.* [33] and Zeng *et al.* [28] considered the VNF placement problem in VNF Forwarding Graph (VNFFG), respectively. In order to minimize the communication delay between two dependent VNFs, Hawilo *et al.* [27] divided VNF types into different sub-groups based on their inherited dependency from VNFFG. Next, all the available servers are used to build a weighted graph where servers are connected with logical communication links. Then, based on sub-groups and a weighted graph, the betweenness centrality is computed for vertices to obtain the best VNF placement solution. Mechtri *et al.* [33] studied the VNF placement and chaining problem and proposed a heuristic named as eigendecomposition which aimed to obtain the optimal matching of a VNFFG in physical network according to Umeyama's eigendecomposition approach. Zeng *et al.* [28] considered the VNF placement problem in inter-datacenter elastic optical networks, and proposed three heuristics to minimize the total cost consisting of spectrum utilization on fiber links, resource consumption in datacenter and the cost of VNF deployment.

Moreover, the problem of VNF placement for SFCs is considered by Pham *et al.* [34] for the purpose of energy and traffic-aware cost minimization. Since the problem was NP-hard and solution space was very large, they proposed a novel two-step algorithm based on the combination of Markov approximation technique and matching approach to efficiently solve the problem, where the first step was to find the subset of nodes to place VNFIs and the second step was to place VNFIs to minimize the total system cost. Eramo *et al.* [22], [35] considered the migration of VNFIs for SFCRs in NFV-enabled networks based on Markov decision process theory, where the objective was to minimize the energy consumption and reconfiguration cost of VNFIs. Liu *et al.* [13] solved the middlebox placement problem based on simulated annealing, where the objective in their model was to minimize the bandwidth consumptions and end-to-end delay among switches and middleboxes. Xiao *et al.* [36] studied the SFC deployment problem, and proposed a DRL approach to deploy SFCs aiming to jointly optimize the operation cost of NFV providers and the total throughput of requests.

Most of these mentioned solutions for the VNF placement problem need to abstract problems with complex math models. These complex math models can only be solved when the network scale is small, while heuristics are more preferred in large scale networks. However, due to lack of strict theoretical proof, heuristics cannot always guarantee to obtain close-to-optimal results. Different from these mentioned methods, in this paper, our proposed DDQN-VNFPA is based on DRL technique which can effectively use collected network information as training data to improve its performance. Moreover, our proposed DDQN-VNFPA is not sensitive to network scale. To our best knowledge, this work is the first one to solve the VNF placement problem using DRL.

B. Technical Background

DRL is competent in sequential decision-making problems with high-dimensional states and actions [37]. As one of DRL approaches, Deep Q Network (DQN) has gained superhuman performance in many video games of Atari 2600 [16]. In DQN, a deep neural network is used to approximate Q function which is used to evaluate how good a state-action pair is under a policy. DQN uses experience reply to achieve efficient model training. As for experience reply, the sequences consisting of current state, action, reward and next state are recorded in memory and picked randomly to update Q function, which can effectively break the correlation between samples.

Nevertheless, studies show that DQN overestimates Q-values, then DDQN is proposed to solve the problem. DDQN includes two neural networks named as online neural network and target neural network. In DDQN, online neural network takes charge of sampling and action selection, and the Q value evaluation is conducted by target neural network. By decoupling action selection and Q value evaluation, DDQN can efficiently enhance stability and mitigate overestimation during model training.

In this paper, we propose DDQN-VNFPA to intelligently solve the VNF placement problem. As the change of network resource has influence on the placement of VNFIs, we take the bandwidth, memory, processor cores and CPU in links, nodes and VNFIs as network resource states, and actions represent different VNF placement strategies that can be conducted to adjust the placement of VNFIs. In SDN/NFV-enabled networks, network resource states cannot be all enumerated, and there exist plenty of VNF placement actions. Thus, the VNF placement belongs to a high-dimensional sequential decision-making problem, and it is difficult to use traditional RL algorithms to solve it. Fortunately, in DDQN, neural networks are naturally competent in the matching of high-dimensional states and actions. With the help of DDQN, we can efficiently evaluate different VNF placement actions under network resource states, which can be used to optimize the placement of VNFIs in the network.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

III. System Model

A. Physical Network and VNFIs

Physical network is presented as a graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{M})$. The parameters $\mathcal{V}$ and $\mathcal{E}$ stand for the sets of nodes and links, respectively. The parameters $u, v \in \mathcal{V}$ represent two nodes and $uv \in \mathcal{E}$ stands for a physical link connecting node u and v. The set of VNFIs is denoted as $\mathcal{M}$, and $m \in \mathcal{M}$ indicates VNFI m. In the network, $\mathcal{V}$ consists of two kinds of nodes. One is switch node which is in charge of forwarding packets to neighbour nodes. The other one is function node that can not only communicate with neighbour nodes, but also place VNFIs to handle traffic of SFCRs.

In the network, bandwidth, memory, processor cores and CPU are considered metrics for links, nodes and VNFIs. We use C_{uv}^{bw} to represent the bandwidth capacity of link uv, and C_u^{mem} denotes the memory capacity of node u. We use $\mathcal{P}$ to indicate the set of all the VNF types and $p \in \mathcal{P}$ stands for VNF type p. Since running VNFIs require processor cores from the corresponding function node and the number of available processor cores within each function node is finite, we use C_u^{core} to represent the number of available processor cores of node u, and n_p^{core} represents the number of processor cores that VNF type p demands. Then, C_m^{cpu} denotes the CPU capacity that VNFI m can provide to handle the traffic of SFCRs. We use ω_{uv}^{bw}, ω_u^{mem} and ω_m^{cpu} to indicate the available ratios of bandwidth, memory, and CPU of link uv, node u and VNFI m, respectively. Moreover, we use d_{uv}, d_u and d_m to represent the delay in link uv, node u and VNFI m.

B. Service Function Chain Request

We use service function graph $G_f = (\mathcal{V}_f, \mathcal{E}_f)$, which is a digraph and the edges are from the ingress node to egress node concatenating a series of VNF Requests (VNFRs) in a predefined order, to represent $SFCR_f$. The parameter $\mathcal{V}_f$ denotes ingress node, egress node and the set of VNFRs of $SFCR_f$, and $u_f, v_f \in \mathcal{V}_f$ represent two nodes in G_f. The parameter $\mathcal{E}_f$ indicates the set of links connecting adjacent VNFRs of $SFCR_f$, and $u_f v_f \in \mathcal{E}_f$ stands for a link connecting nodes u_f and v_f in G_f. For example, assuming that $SFCR_f$ is B $\rightarrow$ FW $\rightarrow$ DPI $\rightarrow$ I, B and I represent the ingress node and egress node in the network, and its traffic needs to traverse the instances of FW and DPI in order. For $SFCR_f$, we use φ_f^{bw} and φ_f^{mem} to represent the bandwidth and memory consumptions in physical links and nodes, respectively. And we use $\varphi_{f,p}^{cpu}$ to denote the CPU consumption in VNFIs belonging to VNF type p. In addition, the parameter φ_f^{delay} indicates the maximum tolerated delay of $SFCR_f$.

IV. Problem Statement

In this section, we give a detailed description to the VNF placement problem, then formulize it using BIP model.

A. Problem Description

Since network load changes dynamically and periodically, it is important for ISPs to improve network performance and reduce resource consumptions and extra cost by optimizing the placement of VNFIs over time in SDN/NFV-enabled networks. For example, in Fig. 1, we show the average change of network load in TOTEM project [38] which conducts a trace-driven emulation in a transit network. Moreover, we assume that the SFCRs in a future time interval $\triangle t$ can be forecasted. Thus, in order to adapt to dynamic change of network load, we optimize the placement of VNFIs with forecasted SFCRs per $\triangle t$ time interval.

In SDN/NFV-enabled networks, the number of SFCRs that fail to be served has a great influence on the QoS/QoE. VNFI running time denotes the total time that all the placed VNFIs occupy and it is related to the consumptions of energy and network resources. Additionally, VNF placement cost results from the costs of computing power, license fees and the utilization of network resources [9], [12]. So it is necessary to take reject SFCRs, VNFI running time and VNF placement cost into account, when solving the VNF placement problem. To sum up, in this paper, considering finite network resources including bandwidth, memory, processor cores and CPU in links, nodes and VNFIs, our objective is to make the optimal VNF placement action according to these forecasted SFCRs in a future time interval $\triangle t$, so as to minimize a weighted cost consisting of VNF placement cost, penalty of reject SFCRs and VNFI running cost.

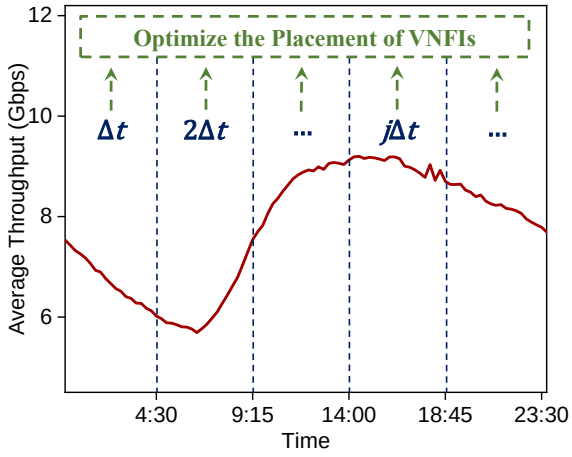


Fig. 1. Optimize VNF placement with dynamic change of network load.

B. Problem Formulation

In this subsection, we formulate the problem of VNF placement for SFCRs as a BIP model in detail.

As SFCRs will come and go as they run, in a future time interval, we use $\mathcal{O}$, $\mathcal{L}$ and $\mathcal{R}$ to represent the set of new arrival SFCRs, the set of expired SFCRs and the set of SFCRs that needs to be redirected because of VNF placement. Then, in a future time interval, the consumptions of bandwidth, memory and CPU cannot exceed the available resources in links ($\forall uv \in \mathcal{E}$), nodes ($\forall u \in \mathcal{V}$) and VNFIs ($\forall m \in \mathcal{M}$, $\forall p \in \mathcal{P}$) as:

$$\sum_{u_f v_f \in \mathcal{E}_f} \sum_{f \in \mathcal{O} \cup \mathcal{L} \cup \mathcal{R}} \varphi_f^{bw} \left(z_{uv}^{u_f v_f} - \hat{z}_{uv}^{u_f v_f} \right) \leq \omega_{uv}^{bw} C_{uv}^{bw}, \quad (1)$$

$$\sum_{u_f v_f \in \mathcal{E}_f} \sum_{f \in \mathcal{O} \cup \mathcal{L} \cup \mathcal{R}} \varphi_f^{mem} \left(z_u^{u_f v_f} - \hat{z}_u^{u_f v_f} \right) \leq \omega_u^{mem} C_u^{mem}, \quad (2)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

$$\sum_{u_f \in \mathcal{V}_f} \sum_{f \in \mathcal{O} \cup \mathcal{L} \cup \mathcal{R}} \varphi_{f,p}^{cpu} q_p^m \left(z_m^{u_f} - \hat{z}_m^{u_f} \right) \leq \omega_m^{cpu} C_m^{cpu}. \quad (3)$$

In Eqs. (1)-(2), $z_{uv}^{u_f v_f}$ and $z_u^{u_f v_f}$ are binary variables and indicate whether $u_f v_f \in \mathcal{E}_f$ traverses $uv \in \mathcal{E}$ and $u \in \mathcal{V}$, respectively. And $z_{uv}^{u_f v_f}$ and $z_u^{u_f v_f}$ equal 1, if $u_f v_f \in \mathcal{V}_f$ traverses $uv \in \mathcal{E}$ and $u \in \mathcal{V}$, and 0 otherwise. Similarly, in Eq. (3), binary variable $z_m^{u_f}$ denotes whether $u_f \in \mathcal{V}_f$ is served by VNFI $m \in \mathcal{M}$. And $z_m^{u_f}$ equals 1, if $u_f \in \mathcal{V}_f$ is served by VNFI $m \in \mathcal{M}$, and 0 otherwise. The parameter q_p^m is also a binary variable and it denotes whether VNFI m belongs to VNF type p. And q_p^m equals 1, if VNFI m belongs to VNF type p, and 0 otherwise. Here, we use $\hat{z}_{uv}^{u_f v_f}$, $\hat{z}_u^{u_f v_f}$ and $\hat{z}_m^{u_f}$ to represent the values of $z_{uv}^{u_f v_f}$, $z_u^{u_f v_f}$ and $z_m^{u_f}$ in the last time interval. Then, in Eqs. (1)-(3), the first parts represent the resource consumptions in future time interval, and the second parts represent the ones in the last time interval. Thus, the differences indicate the change of available resources in links, nodes and VNFIs, and Eqs. (1)-(3) guarantee that the resource consumptions in links, nodes and VNFIs after change cannot exceed their available resources.

The total number of processor cores applied by the placed VNFIs in a function node cannot exceed the number of available processor cores as:

$$\sum_{m \in \mathcal{M}} n_p^{core} y_u^m q_p^m \leq C_u^{core}, \forall p \in \mathcal{P}, \forall u \in \mathcal{V}, \quad (4)$$

where binary variable y_u^m represents whether VNFI $m \in \mathcal{M}$ is placed in node $u \in \mathcal{V}$. And y_u^m equals 1, if VNFI $m \in \mathcal{M}$ is placed in node $u \in \mathcal{V}$, and 0 otherwise.

For each served or new arrival SFCR $f \in \mathcal{O} \cup \mathcal{R}$, the end-to-end delay cannot exceed its maximum tolerated delay as:

$$\sum_{uv \in \mathcal{E}} \sum_{u_f v_f \in \mathcal{E}_f} d_{uv} z_{uv}^{u_f v_f} + \sum_{u \in \mathcal{V}} \sum_{u_f v_f \in \mathcal{E}_f} d_u z_u^{u_f v_f} + \sum_{m \in \mathcal{M}} \sum_{u_f \in \mathcal{V}_f} d_m z_m^{u_f} \leq \varphi_f^{delay}. \quad (5)$$

For *SFCR*$_f$ ($f \in \mathcal{O} \cup \mathcal{L} \cup \mathcal{R}$), the physical links it traverses must be connected head-to-tail and cannot be split as:

$$\sum_{v \in \mathcal{V}} \sum_{u_f v_f \in \mathcal{E}_f} \left(z_{uv}^{u_f v_f} - z_{vu}^{u_f v_f} \right) = \begin{cases} 1, & u \text{ is ingress node}, \\ -1, & u \text{ is egress node}, \\ 0, & \text{otherwise}. \end{cases} \quad (6)$$

If a physical link is traversed, the nodes connected by this physical link should be traversed as well ($f \in \mathcal{O} \cup \mathcal{L} \cup \mathcal{R}$):

$$z_u^{u_f v_f} z_v^{u_f v_f} = \begin{cases} 1, & z_{uv}^{u_f v_f} = 1, \forall u, v \in \mathcal{V}, \forall uv \in \mathcal{E}, \forall u_f v_f \in \mathcal{E}_f, \\ 0, & \text{otherwise}. \end{cases} \quad (7)$$

Eq. (8) ensures that all the selected links, nodes and VNFIs must be traversed by the traffic of *SFCR*$_f$ as:

$$\sum_{m \in \mathcal{M}} \sum_{u_f \in \mathcal{V}_f} z_m^{u_f} y_u^m \leq z_{uv}^{u_f v_f}, \forall u \in \mathcal{V}, \forall u_f v_f \in \mathcal{E}_f, \forall f \in \mathcal{O} \cup \mathcal{R}. \quad (8)$$

For VNFI m, it can only belong to one VNF type as:

$$\sum_{p \in \mathcal{P}} q_p^m = 1, \forall m \in \mathcal{M}. \quad (9)$$

VNFI m can only be placed in one function node as:

$$\sum_{u \in \mathcal{V}} y_u^m = 1, \forall m \in \mathcal{M}. \quad (10)$$

For a VNFI to place, the VNF placement cost is c^{place}. Then we can compute the total VNF placement cost in future time interval $\triangle t$ as:

$$\mathbb{D} = c^{place} \sum_{u \in \mathcal{V}} \sum_{m \in \mathcal{M}} max \left\{ y_u^m - \hat{y}_u^m, 0 \right\}, \quad (11)$$

where $\hat{y}_u^m$ represents the value of y_u^m in the last time interval.

In a future time interval $\triangle t$, the VNFI running time can be calculated as:

$$\mathbb{F} = \triangle t \sum_{u \in \mathcal{V}} \sum_{m \in \mathcal{M}} y_u^m. \quad (12)$$

We suppose that each reject SFCR leads to a penalty $c^{penalty}$. Then the penalty of reject SFCRs in future time interval $\triangle t$ is calculated as:

$$\mathbb{U} = \sum_{f \in \mathcal{O} \cup \mathcal{R}} c^{penalty} \left(1 - \mathbb{I} \Big(\sum_{u \in \mathcal{V}} z_u^{u_f v_f} > 0 \Big) \right), \forall u_f v_f \in \mathcal{E}_f. \quad (13)$$

Here, $\mathbb{I}(\cdot)$ is an indicative function and its condition indicates whether SFCR$_f$ is successfully served or not. And $\mathbb{I}(\cdot)$ equals 1, if SFCR$_f$ is successfully served, and 0 otherwise.

In this paper, our objective is to minimize the weighted cost consisting of VNF placement cost, penalty of reject SFCRs and VNFI running cost in every time interval $\triangle t$ as follows:

$$\underset{z_{uv}^{u_f v_f}, z_u^{u_f v_f}, z_m^{u_f}}{\textit{Minimize}} \quad \eta_1 \mathbb{D} + \eta_2 \mathbb{F} + \eta_3 \mathbb{U}, \quad s.t.\ Eqs.\ (1) - (10), \quad (14)$$

where η_1, η_2 and η_3 are weighted parameters to get good trade off among the three parts.

V. VNF Placement with the Help of DDQN

A. Overview

We propose an intelligent method, DDQN-VNFPA, to optimize the placement of VNFIs with dynamic network load. In this paper, we suppose that network load changes periodically with a time cycle T, and SFCRs can be forecasted in future time interval $\triangle t$. In the network, since load changes dynamically and its change tendency may be different over time. For example, as shown in Fig. 1, network load increases during 6 am – 12 am; during 12 am – 4 pm, network load keeps stable, then continuously goes down from 4 pm to 6 am the next day. Additionally, for different time slots, though their network load changes with similar tendency (*e.g.,* in Fig. 1, the network load both increases during 7 am – 9 am and 9 am – 11 am), there are many differences among them because of complex network states and resource usage. In order to improve the performance of DDQN models considering the change of network load, we separate time cycle T with time interval $\triangle t$, and train a DDQN model to specifically take charge of the optimization of VNFI placement in each time interval $\triangle t$.

In the network, devices (*e.g.,* nodes and links) could join in and leave making the network topology change dynamically, so it is necessary to design a VNF placement algorithm that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

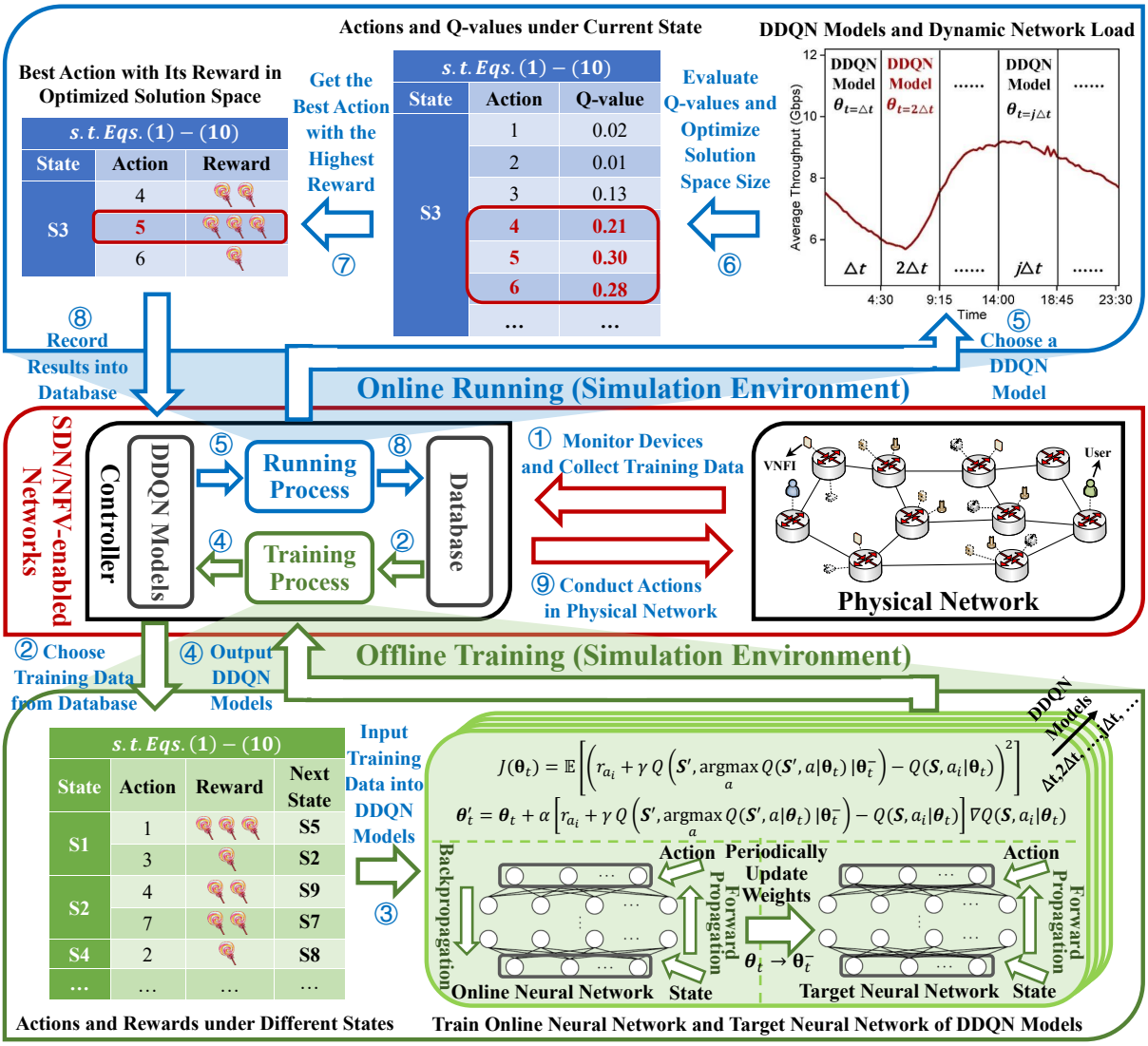


Fig. 2. Framework of DDQN-VNFPA.

adapts to the change of network topology. In this paper, the whole network topology is divided into N network regions, and each node or link is related to a network region. Since network regions do not change when nodes and links dynamically join in and leave, the proposed DDQN-VNFPA can adapt to the change of network topology by training DDQN models according to network regions.

In DDQN-VNFPA, we treat each combination of network regions as an action. If an action is chosen to be conducted, it means that we need to optimize the placement of VNFIs in function nodes of the network regions corresponding to this selected action. For example, assuming that there are two network regions A and B, then the actions ϕ, $\{A\}$, $\{B\}$, $\{A, B\}$ represent the corresponding combination of network regions. If action $\{A, B\}$ is selected, it means that it is necessary to optimize the VNF placement in the function nodes of network regions A and B. Thus, as for N network regions, there are 2^N actions. It is noted that, for briefness, we use state to indicate the network resource state in this paper.

DDQN-VNFPA includes offline training process and online running process. In the training process, we collect training data and train a DDQN model to take charge of the VNF placement for each time interval $\triangle t$. After training process, we can run those trained DDQN models online to achieve the optimization of VNF placement according to states. Additionally, once it is necessary to update DDQN models, the only thing we need to do is to make a copy of the old trained DDQN models and continue to train it using new collected training data. When the training process of new DDQN models is completed, we can replace old DDQN models with the new ones.

In the running process, DDQN-VNFPA optimizes the placement of VNFIs with three phases. In the first phase, a DDQN model is chosen according to current time to give a preliminary evaluation of each action under current state and optimize the solution space size by choosing the top k actions from total 2^N actions. In the second phase, we conduct the top k actions considering forecasted SFCRs in simulation environment to get rewards, and record results into database to further update DDQN models. Here, a simulation environment of physical network is constructed to evaluate the performance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

of actions. In simulation environment, all the processes do not influence physical network. In the third phase, the action with the highest reward in optimized solution space is conducted to optimize the placement of VNFIs in physical network based on a threshold-based policy. It is worthy to note that the SFC-MAP algorithm in our previous work [9], which achieves load balancing according to the costs of bandwidth, memory and CPU, is used to compute routing paths of SFCRs.

Fig. 2 shows the framework of DDQN-VNFPA. In step ①, SDN controller monitors devices to collect and record training data into database. Step ②-④ take charge of the training process of DDQN models. Step ② selects a batch of training data from database, then they are used to train DDQN models in step ③. The trained DDQN models are output into SDN controller in step ④.

The running process of DDQN-VNFPA includes steps ⑤-⑨. In step ⑤, the corresponding DDQN model is chosen according to current time $t = \triangle t, ..., j\triangle t, ...$ For example, assuming that current time is $t = 2\triangle t$, the DDQN model $\boldsymbol{\theta}_{t=2\triangle t}$ is chosen in step ⑤. Next, the chosen DDQN model evaluates all the actions in current state preliminarily, and get their Q-values in step ⑥. As Q-values can reflect the performance of actions, the actions with top k Q-values are selected in step ⑥ to optimize the solution space. For example, assuming that current state is $S3$ and $k = 3$, only the action 4 to 6 that have the top three Q-values are chosen in this step. Then, based on current state, step ⑦ conducts chosen actions considering forecasted SFCRs in simulation environment to get rewards. The action with the highest reward (*e.g.,* action 5 has the highest reward in step ⑦) is regarded as the best one. After that, in step ⑧, all the results including current states, actions, their rewards and the next states after conducting actions are recorded into database to further update DDQN models. Finally, in step ⑨, DDQN-VNFPA gets the network regions that the best action is corresponding to, and optimizes the placement of VNFIs in them according to a threshold-based policy.

B. Structure of Neural Network in DDQN Model

In DRL, neural network is used to replace Q table in conventional RL algorithm to achieve the mapping among states, actions and Q-values. DDQN includes two neural networks of the same structure named as online neural network and target neural network. Since the resource conditions of network can influence the computation of the optimal strategy of VNF placement, we define that the state of DDQN-VNFPA includes available resource ratios of links, nodes and VNFIs. Noting that the whole network topology is divided into network regions, we use $\mathcal{V}_n \subset \mathcal{V}$, $\mathcal{E}_n \subset \mathcal{E}$ and $\mathcal{M}_n \subset \mathcal{M}$ to represent the sets of nodes, links and VNFIs in the n^{th} network region and N represents the total number of network regions, $n = 1, 2, ...N$. Then, each state is denoted as a vector as:

$$\boldsymbol{S} = \left(\overline{\omega}_1^{bw}, \overline{\omega}_1^{mem}, \overline{\omega}_1^{core}, \overline{\omega}_{1,p}^{cpu}, ..., \overline{\omega}_n^{bw}, \overline{\omega}_n^{mem}, \overline{\omega}_n^{core}, \overline{\omega}_{n,p}^{cpu}, ...\right)^T. \quad (15)$$

In Eq. (15), $\overline{\omega}_n^{bw}$, $\overline{\omega}_n^{mem}$ and $\overline{\omega}_n^{core}$ represent mean available ratios of bandwidth, memory and processor cores in network region n, respectively, and they are calculated as follows:

$$\overline{\omega}_n^{bw} = \frac{1}{|\mathcal{E}_n|} \sum_{uv \in \mathcal{E}_n} \omega_{uv}^{bw}, \quad (16)$$

$$\overline{\omega}_n^{mem} = \frac{1}{|\mathcal{V}_n|} \sum_{u \in \mathcal{V}_n} \omega_u^{mem}, \quad (17)$$

$$\overline{\omega}_n^{core} = 1 - \frac{\sum_{m \in \mathcal{M}_n} n_p^{core} y_u^m q_p^m}{\sum_{u \in \mathcal{V}_n} C_u^{core}}. \quad (18)$$

The parameter $\overline{\omega}_{n,p}^{cpu}$ indicates mean available CPU ratio of VNF type p in network region n, and it can be computed as follows:

$$\overline{\omega}_{n,p}^{cpu} = \frac{\sum_{m \in \mathcal{M}_n} \sum_{u \in \mathcal{V}_n} \omega_m^{cpu} y_u^m q_p^m}{\sum_{m \in \mathcal{M}_n} \sum_{u \in \mathcal{V}_n} y_u^m q_p^m}. \quad (19)$$

Given the definition above, state $\boldsymbol{S}$ can be formatted and served as the input of neural network of DDQN. As for the output of neural network in DDQN model, each dimension represents an action. The value of a dimension indicates the Q-value of the corresponding action. In DDQN, after model training, neural networks can evaluate the performance of each action under state $\boldsymbol{S}$. Then, we denote the output of neural networks as follows:

$$\boldsymbol{y} = \left(Q\left(\boldsymbol{S}, a_1\right), ..., Q\left(\boldsymbol{S}, a_i\right), ...\right)^T, i = 1, 2, ..., 2^N, \quad (20)$$

where $Q(\boldsymbol{S}, a_i)$ stands for the evaluated Q-value by conducting action a_i under state $\boldsymbol{S}$. Since there are N network regions and each combination of network regions represents an action, the total number of actions equals 2^N.

C. Generation of Training Data

In DDQN-VNFPA, the reward conducting action a_i is denoted as r_{a_i} which is used to reflect the influence of action a_i in the network. In this paper, our objective is to minimize a weighted cost including the VNF placement cost, penalty of reject SFCRs and VNFI running cost in Eq. (14). Thus, if a_i can get good performance in the reduction of Eq. (14), r_{a_i} is associated a high value. On the contrary, r_{a_i} is set to a small value, if a_i leads to high cost in Eq. (14). Given the explanation above, we define the calculation of reward as follows:

$$r_{a_i} = -\eta_1 \mathbb{D} - \eta_2 \mathbb{F} - \eta_3 \mathbb{U}, i = 1, 2, ..., 2^N. \quad (21)$$

We generate training data according to the following steps. If the state at current time t is $\boldsymbol{S}$, we conduct each action a_i considering the forecasted SFCRs of this time interval in the simulation environment by placing/releasing VNFIs in the corresponding network regions, then get its reward r_{a_i} according to Eq. (21) and the next state $\boldsymbol{S}'$ which represents the state after VNF placement. And all the results including $\boldsymbol{S}, a_i, r_{a_i}$ and $\boldsymbol{S}'$ are recorded as training data. Afterwards, we conduct the best action with the highest reward in physical network, then we continue to optimize the VNF placement in the next time interval. Finally, the following processes are repeated until we generate enough training data.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

D. Training Process

In this paper, we assume that the network load changes with time cycle T and we can forecast the variation of SFCRs in future time interval $\triangle t$. In order to obtain good training and prediction performance given the change tendency of network load, we discretize time cycle T with time interval $\triangle t$, and train a DDQN model for each time interval $\triangle t$. So, in DDQN-VNFPA, there are $\left\lceil \frac{T}{\triangle t} \right\rceil$ DDQN models needed to be trained.

For the training process of a DDQN model, the loss function, which is used to indicate the estimation performance, is defined in Eq. (22) as follows:

$$J(\boldsymbol{\theta}_t) = \mathbb{E}\left[\left(r_{a_i} + \gamma Q\Big(\boldsymbol{S}', \underset{a}{argmax}\, Q\big(\boldsymbol{S}', a|\boldsymbol{\theta}_t\big)|\boldsymbol{\theta}_t^-\Big) - Q\big(\boldsymbol{S}, a_i|\boldsymbol{\theta}_t\big)\right)^2\right]. \tag{22}$$

In DDQN models, we use $\boldsymbol{\theta}_t$ to represent the online neural network at time t, and $\boldsymbol{\theta}_t^-$ denotes the target neural network at time t. Here, the first part $r_{a_i}+\gamma Q\Big(\boldsymbol{S}', \underset{a}{argmax}\, Q\big(\boldsymbol{S}', a|\boldsymbol{\theta}_t\big)|\boldsymbol{\theta}_t^-\Big)$ represents a target that the Q-value needs to move and the second part $Q\big(\boldsymbol{S}, a_i|\boldsymbol{\theta}_t\big)$ represents the estimation of Q-value [37]. In Eq. (22), $Q\Big(\boldsymbol{S}', \underset{a}{argmax}\, Q\big(\boldsymbol{S}', a|\boldsymbol{\theta}_t\big)|\boldsymbol{\theta}_t^-\Big)$ is solved in two steps. The first step is to find the action a with the highest Q-value according to online neural network $\boldsymbol{\theta}_t$ under state $\boldsymbol{S}'$. The second step is to evaluate the Q-value of action a under state $\boldsymbol{S}'$ using the target neural network $\boldsymbol{\theta}_t^-$. Therefore, the loss function indicates the estimation error of a DDQN model, and the smaller the loss function is, the better estimation performance a DDQN model will have. Given the statement above, the loss function of DDQN model at time t is symbolized as $J(\boldsymbol{\theta}_t)$, and it equals the expectation of the square of the summation among r_{a_i}, $\gamma Q\Big(\boldsymbol{S}', \underset{a}{argmax}\, Q\big(\boldsymbol{S}', a|\boldsymbol{\theta}_t\big)|\boldsymbol{\theta}_t^-\Big)$ and $-Q\left(\boldsymbol{S}, a_i|\boldsymbol{\theta}_t\right)$, where γ is discount-rate parameter. The gradient descent algorithm is used to update the weights of online neural network, which is helpful to optimize the loss function in Eq. (22), as follows:

$$\boldsymbol{\theta}_t^{'} = \boldsymbol{\theta}_t + \alpha\left[r_{a_i} + \gamma Q\Big(\boldsymbol{S}', \underset{a}{argmax}\, Q\big(\boldsymbol{S}', a|\boldsymbol{\theta}_t\big)|\boldsymbol{\theta}_t^-\Big) - Q(\boldsymbol{S}, a_i|\boldsymbol{\theta}_t)\right]\nabla Q\big(\boldsymbol{S}, a_i|\boldsymbol{\theta}_t\big), \tag{23}$$

where $\boldsymbol{\theta}_t^{'}$ and $\boldsymbol{\theta}_t$ indicate online neural network after and before update and α is the step-size parameter.

The pseudocode of training process is listed in **Algorithm 1**. The weights of DDQN models are initialized in line 2. In line 4, we choose a batch of training data from database, and get the outputs which represent Q-values of the corresponding actions under current state in line 5. Then, the loss function is calculated in line 6, and we decide whether it is convergent in line 7-12. In line 7-8, if the value change of loss function is smaller than a threshold ϵ, the model is decided to be convergent, then we stop the training process and train a DDQN model at the next time interval in line 14. And if the model is not convergent, we update online neural network in line 10 and periodically update target neural network in line 11.

Algorithm 1: Training Process

1: **for** $t = \triangle t, ..., j\triangle t, ...$ **do**
2: $\quad\boldsymbol{\theta}_t, \boldsymbol{\theta}_t^- \leftarrow$ Initialize the weights of online and target neural networks of the DDQN model at time t;
3: $\quad$**while** *True* **do**
4: $\quad\quad(\boldsymbol{S}, a_i, r_{a_i}, \boldsymbol{S'}) \leftarrow$ Choose a batch of training data randomly from database;
5: $\quad\quad\boldsymbol{y} \leftarrow \left(Q\left(\boldsymbol{S}, a_1\right), ..., Q\left(\boldsymbol{S}, a_i\right), ...\right)^T, i = 1, ..., 2^N$;
6: $\quad\quad J(\boldsymbol{\theta}_t) \leftarrow \mathbb{E}\Big[\Big(r_{a_i} + \gamma Q\big(\boldsymbol{S}', \underset{a}{argmax}\, Q(\boldsymbol{S}', a|\boldsymbol{\theta}_t)|\boldsymbol{\theta}_t^-\big) - Q(\boldsymbol{S}, a_i|\boldsymbol{\theta}_t)\Big)^2\Big]$;
7: $\quad\quad$**if** $\Big|J(\boldsymbol{\theta}_t^{'}) - J(\boldsymbol{\theta}_t)\Big| <= \epsilon$ **then**
8: $\quad\quad\quad$break;
9: $\quad\quad$**else**
10: $\quad\quad\quad\boldsymbol{\theta}_t \leftarrow$ Update the weights of online neural network with Eq. (23);
11: $\quad\quad\quad\boldsymbol{\theta}_t^- \leftarrow$ Periodically update the weights of target neural network with $\boldsymbol{\theta}_t$;
12: $\quad\quad$**end**
13: $\quad$**end**
14: $\quad\boldsymbol{\Theta}(t) \leftarrow$ Record DDQN model $\boldsymbol{\theta}_t$;
15: **end**

E. Running Process

After model training, DDQN-VNFPA can solve the VNF placement problem in SDN/NFV-enabled networks. As the number, type and resource consumptions of SFCRs change dynamically in different time intervals, it is difficult to format forecasted SFCRs as the input of DDQN models. Thus, the optimal VNF placement for forecasted SFCRs in a time interval cannot be obtained directly. Fortunately, based on collected historical network information, we can evaluate the performance of an action statistically. If an action always gets good performance in the past, it is also likely to perform well in the future. On the contrary, if an action often performs badly in the past, it is unlikely to obtain good performance in the future.

The running process of DDQN-VNFPA includes three phases. The first phase chooses the corresponding DDQN model, then preliminarily evaluates Q-values of actions by inputting current state into it. We avoid actions with bad performance and select the ones that are likely to obtain good performance statistically to optimize solution space size. In the second phase, we evaluate the performance of each action in optimized solution space by conducting actions according to forecasted SFCRs in simulation environment to get rewards, and record results into database to further update DDQN models. In the third phase, the action with the highest reward is conducted in physical network according to a threshold-based policy.

In DDQN-VNFPA, the threshold-based policy is related to available resources and load variation. In papers [9], [39] and [40], CPU thresholds are used to identify overloaded VNFIs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

Algorithm 2: Running Process

1:**for** $n = 1, 2, ..., N$ **do**
2: $\overline{\omega}_n^{bw} \leftarrow \frac{1}{|\mathcal{E}_n|} \sum_{uv \in \mathcal{E}_n} \omega_{uv}^{bw}$;
3: $\overline{\omega}_n^{mem} \leftarrow \frac{1}{|\mathcal{V}_n|} \sum_{u \in \mathcal{V}_n} \omega_u^{mem}$;
4: $\overline{\omega}_n^{core} \leftarrow 1 - \frac{\sum_{m \in \mathcal{M}_n} n_p^{core} y_u^m q_p^m}{\sum_{u \in \mathcal{V}_n} C_u^{core}}$;
5: **for** $p \in \mathcal{P}$ **do**
6: $\overline{\omega}_{n,p}^{cpu} = \frac{\sum_{m \in \mathcal{M}_n} \sum_{u \in \mathcal{V}_n} \omega_m^{cpu} y_u^m q_p^m}{\sum_{m \in \mathcal{M}_n} \sum_{u \in \mathcal{V}_n} y_u^m q_p^m}$;
7: **end**
8:**end**
9:$\boldsymbol{S} \leftarrow \left(\overline{\omega}_1^{bw}, \overline{\omega}_1^{mem}, \overline{\omega}_1^{core}, \overline{\omega}_{1,p}^{cpu}, ..., \overline{\omega}_n^{bw}, \overline{\omega}_n^{mem}, \overline{\omega}_n^{core}, \overline{\omega}_{n,p}^{cpu}, ...\right)^T$;
10:$\boldsymbol{\theta}_t \leftarrow$ Select the corresponding DDQN model from $\boldsymbol{\Theta}(t)$ according to time t;
11:$\boldsymbol{y} \leftarrow$ Put $\boldsymbol{S}$ into DDQN model $\boldsymbol{\theta}_t$ to get the output;
12:$\mathcal{A} \leftarrow$ Select the top k actions in $\boldsymbol{y}$ to optimize solution space size;
13:**for** $a_i \in \mathcal{A}$ *in simulation environment* **do**
14: Optimize the VNF placement of the corresponding network regions according to action a_i and state $\boldsymbol{S}$;
15: Steer $SFCR_f, f \in \mathcal{O} \cup \mathcal{L} \cup \mathcal{R}$;
16: $r_{a_i} \leftarrow -\eta_1 \mathbb{D} - \eta_2 \mathbb{F} - \eta_3 \mathbb{U}$;
17: $\boldsymbol{S'} \leftarrow$Get the next state by conducting a_i;
18: Record $(\boldsymbol{S}, a_i, r_{a_i}, \boldsymbol{S'})$ into database.
19:**end**
20:$a^* \leftarrow$ Get the best action with the highest reward from $\mathcal{A}$;
21:Conduct a^* in physical network; $\Rightarrow$ **Function 1**

Function 1: Threshold-based Policy

1:**for** $n = 1, 2, ..., N$ **do**
2: **if** a^* *needs to optimize the VNF placement in network region* n **then**
3: **for** $p \in \mathcal{P}$ **do**
4: $\zeta_{p,n}^{cpu} \leftarrow \sum_{u \in \mathcal{V}_n} \sum_{m \in \mathcal{M}_n} \omega_m^{cpu} C_m^{cpu} y_u^m q_p^m$;
5: **if** *network load is decreasing* **then**
6: **if** $\zeta_{p,n}^{cpu} < \sigma_{down}^{place}$ **then**
7: Place a new instance of VNF type p;
8: **else if** $\zeta_{p,n}^{cpu} \geq \sigma_{down}^{release}$ **then**
9: Release the instance with the lowest utilization of VNF type p;
10: **end**;
11: **else**
12: **if** $\zeta_{p,n}^{cpu} < \sigma_{up}^{place}$ **then**
13: Place a new instance of VNF type p;
14: **else if** $\zeta_{p,n}^{cpu} \geq \sigma_{up}^{release}$ **then**
15: Release the instance with the lowest utilization of VNF type p;
16: **end**;
17: **end**
18: **end**
19: **end**
20:**end**

In this paper, we also use CPU thresholds to determine whether it is necessary to optimize the number of placed VNFIs of a network region. The CPU thresholds σ_{up}^{place} and $\sigma_{up}^{release}$ are defined to determine whether it is necessary to place or release VNFIs, when network load is increasing. CPU thresholds σ_{down}^{place} and $\sigma_{down}^{release}$ indicate whether it is necessary to place or release VNFIs, when network load is decreasing. Additionally, in network region n, we use $\zeta_{p,n}^{cpu}$ to represent the available CPU of VNF type $p \in \mathcal{P}$ and it is computed as:

$$\zeta_{p,n}^{cpu} = \sum_{u \in \mathcal{V}_n} \sum_{m \in \mathcal{M}_n} \omega_m^{cpu} C_m^{cpu} y_u^m q_p^m. \tag{24}$$

When network load increases, if $\zeta_{p,n}^{cpu} < \sigma_{up}^{place}$, a new instance of VNF type p is placed in network region n. And if $\zeta_{p,n}^{cpu} \geq \sigma_{up}^{release}$, we release the instance with the lowest utilization of VNF type p in network region n. When network load decreases, if $\zeta_{p,n}^{cpu} < \sigma_{down}^{place}$, we place a new instance of VNF type p in network region n, and if $\zeta_{p,n}^{cpu} \geq \sigma_{down}^{release}$, the instance with the lowest utilization of VNF type p in network region n is released. After the placement of VNFIs, we use SFC-MAP algorithm referring to our previous work [9] to construct routing paths for SFCRs. Finally, in optimized solution space, we choose the action with the highest reward calculated according to Eq. (21) and conduct it in physical network.

The pseudocode of running process of DDQN-VNFPA is shown in **Algorithm 2**. Line 1-8 of **Algorithm 2** compute mean available resources in links, nodes and VNFIs, and a state is formatted in line 9 of **Algorithm 2**. Line 10-11 of **Algorithm 2** input the state into selected DDQN model and get the corresponding Q-value of each action. In line 12 of **Algorithm 2**, the actions with top k Q-values are chosen to optimize the size of solution space. We obtain the reward of each action in optimized solution space and record results into database in line 13-19 of **Algorithm 2**. We conduct the best action a^* in physical network in line 20-21 of **Algorithm 2** and optimize the VNF placement according to the threshold-based policy in **Function 1**. For each network region n that are chosen by a^*, we check available CPU for each type of VNF in line 3-18 of **Function 1**. If the network load is decreasing, we optimize the VNF placement in this network region in line 5-10 of **Function 1**, otherwise the VNF placement will be optimized in line 11-17 of **Function 1**.

F. Complexity Analysis

As for DDQN-VNFPA, the training process runs offline and its time complexity is proportional to the number of training data and training time. Thus, we only pay attention to the running process.

In DDQN-VNFPA, the time complexity of running process is related to the structure of neural networks and the size of optimized solution space. Assuming that there are N network regions and $|\mathcal{P}|$ types of VNFIs, as the input vector includes the mean available ratios of bandwidth, memory, processor cores and CPU of all the VNF types, the number of neurons in the input layer of each neural network is $(3 + |\mathcal{P}|) N$. The output of a neural network represents Q-values of actions, and there are 2^N neurons in the output layer of each neural network. In addition, we set that the number of hidden layer of each neural network of a DDQN model is H and the number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

neurons in each hidden layer is I. Then, for state $\boldsymbol{S}$, the time complexity to evaluate the performance of each action with forward propagation is $O\left(I\left(|\mathcal{P}|\,N + IH + 2^N\right)\right)$.

After the preliminary evaluation of actions, the DDQN-VNFPA will further evaluate each of the top k actions according to forecasted SFCRs to get rewards. According to paper [9], in the worst situation, computing the routing paths for *S-FCR*$_f$ runs in $O\left(|\mathcal{M}| + \Gamma\,|\mathcal{V}_f|\,(|\mathcal{E}| + |\mathcal{V}|\,log\,|\mathcal{V}_f|\,|\mathcal{V}|)\right)$, where Γ represents iteration times during routing path computation. In DDQN-VNFPA, since all the new arrival SFCRs and re-directed SFCRs needs to recompute routing paths, then the total time complexity to obtain the rewards of k actions runs in $O\left(k\,(|\mathcal{O}| + |\mathcal{L}|)\,[|\mathcal{M}| + \Gamma\,|\mathcal{V}_f|\,(|\mathcal{E}| + |\mathcal{V}|\,log\,|\mathcal{V}_f|\,|\mathcal{V}|)]\right)$. Additionally, in the worst situation, the time complexity of conducting the best action in physical network is $O\left(|\mathcal{P}|\,N\right)$. Thus, the total time complexity of the running process of DDQN-VNFPA is $O(I\left(|\mathcal{P}|\,N + IH + 2^N\right) + k\,(|\mathcal{O}| + |\mathcal{L}|)\,[|\mathcal{M}| + \Gamma\,|\mathcal{V}_f|\,(|\mathcal{E}| + |\mathcal{V}|\,log\,|\mathcal{V}_f|\,|\mathcal{V}|)])$.

VI. Performance Evaluation

This section demonstrates the performance evaluation of DDQN-VNFPA. We construct a Tensorflow-based environment to evaluate DDQN-VNFPA using Tensorflow-gpu 1.13.1 version [41]. All the simulations are conducted in a computer with an Intel(R) Core(TM) i5-6500 CPU 3.20 @ GHz and a Nvidia GeForce GTX 1080Ti GPU.

A. Simulation Setup

1) Workload of SFCRs: In the simulation, we use the data of Google cluster-usage traces [42] to simulate SFCRs in the network. In Google cluster-usage traces, works are recorded with the format of jobs. A job is comprised of one or more tasks, each of which is accompanied by a set of resource consumptions used for scheduling (packing) tasks onto machines. Since each SFCR consists of one or more VNFRs, the format of an SFCR is similar to the format of a job. Thus, we can regard a job in Google cluster-usage traces as an SFCR, and take tasks as VNFRs. In the simulation, the value of $\varphi_{f,p}^{cpu}$ of *SFCR*$_f$ equals the corresponding CPU consumption of a task. And we select one of tasks randomly to set the memory consumption φ_f^{mem}. Since there is no bandwidth consumptions between tasks, for *SFCR*$_f$, φ_f^{bw} is set to equal a weighted average of memory and CPU consumptions [24].

Additionally, we set that the arrival rate of SFCRs refers to TOTEM project [38]. Through a trace-driven emulation, TOTEM project records traffic matrixes of a transit network per 15 *min* for a period of about 4 months, and its average network throughput over time is shown in Fig. 1.

2) Network Topology and Simulation Settings: The network topology we use in the simulation comes from TOTEM project, and it consists of 23 nodes and 37 links. We divided the whole network topology into 8 network regions each of which has a function node. Therefore, there are 8 function nodes to place VNFIs and the rest 15 nodes are switch nodes. There are 4 types of VNFIs that can be placed in the network. And we set that each function node can place 20 VNFIs at most. In Google cluster-usage traces, the resource consumptions of SFCRs are normalized. Thus, in the simulation, the capacities of bandwidth and memory of each physical link and node are set to 1 Gbps and 1 GB, respectively. The CPU capacity of each VNFI is set to 0.05 MIPS (the total CPU capacity of 20 VNFIs in a function node equals 1 MIPS). We set the weighted factors η_1, η_2 and η_3 in Eq. (14) to $1, 0.01$ and 10, respectively. In the threshold-based policy, the CPU thresholds for VNF placement are $\sigma_{up}^{place} = 0.05$ MIPS, $\sigma_{up}^{release} = 0.15$ MIPS, $\sigma_{down}^{place} = 0.01$ MIPS, $\sigma_{down}^{release} = 0.07$ MIPS which are set according to the CPU load of VNFIs in simulations referring to papers [39], [40]. Furthermore, in training process, if the value change of loss function is smaller than 10^{-5}, a DDQN model is regarded to be convergent.

In the simulation, each SFCR consists of three VNFRs, and the lifetime of each SFCR satisfies the exponential distribution with an average of 250 *min*. The maximum tolerated delay is set between [50, 100) *ms* [43], [44]. In addition, we run DDQN-VNFPA with a time interval $\triangle t = 15$ *min* [19], [38], and each experiment is repeated 20 times.

The queuing delay, propagation delay, processing delay and transmission delay are all considered in the simulation. We use M/M/1 queuing model as our hypothesis to capture the change of delay with network load. The delays of nodes, links and VNFIs in Eq. (5) are computed as follows [9], [45]:

$$d_{uv} = d_{uv}^{prop} + d_{uv}^{tx} + \frac{1 - \omega_{uv}^{bw}}{\omega_{uv}^{bw}} d_{uv}^{tx}, \forall uv \in \mathcal{E}, \tag{25}$$

$$d_u = \frac{1 - \omega_u^{mem}}{\omega_u^{mem}} t_u^{proc}, \forall u \in \mathcal{V}, \tag{26}$$

$$d_m = \frac{1 - \omega_m^{cpu}}{\omega_m^{cpu}} t_m^{proc}, \forall m \in \mathcal{M}. \tag{27}$$

In Eq. (25), we use d_{uv}^{prop} to indicate the propagation delay which is computed by the ratio of the length of link uv to the propagation speed of signals in that medium. The transmission delay d_{uv}^{tx} is computed in the second part and it equals the result by dividing the bandwidth capacity of link uv by the packet size. The third part denotes the queuing delay, and it is related to the load and transmission delay. The processing delays in node u and VNFI m are computed in Eqs. (26)-(27). The parameters t_u^{proc} and t_m^{proc} indicate the per-packet processing delay of node u and VNFI m, and we set them to 10 μs and 1 *ms*, respectively [46]. According to M/M/1 queuing model, for low load, the queuing delay and processing delay grow nearly linear, and they are associated high costs near their capacity, which is desirable to balance load and avoid to utilize systems at their maximum capacity.

3) Introduction of Compared Algorithms: We compare DDQN-VNFPA with MSGAS [25] and Eigendecomposition [33]. Before presenting the evaluation results, we give a brief description to these compared algorithms.

- MSGAS: achieves efficient VNF placement with accessible scope. First, for an SFCR, MSGAS chooses all the VNFIs according to the accessible scope which is used to optimize solution space size. Next, for all the chosen VNFIs, MSGAS arranges and chains them to satisfy the predefined order of this SFCR. Then, MSGAS computes link costs between VNFIs, where VNF deployment cost,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

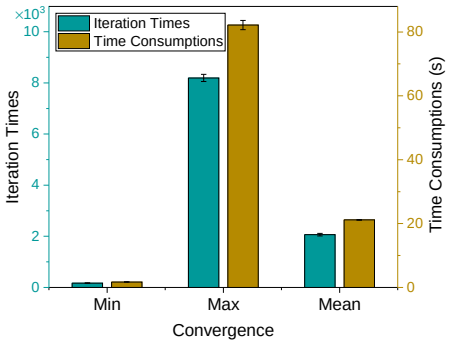


Fig. 3. Iteration times and time consumptions in training process (with 95% confidence intervals).

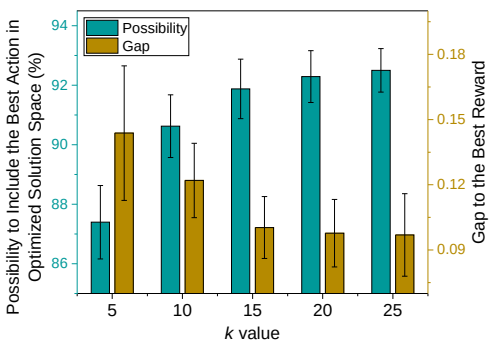


Fig. 4. Influence of optimized solution space size (with 95% confidence intervals).

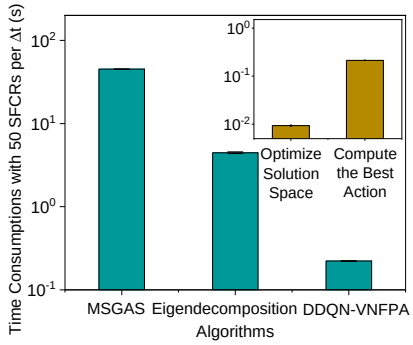


Fig. 5. Time consumptions of running process (with 95% confidence intervals).

energy cost, cost of forwarding traffic, penalty of SLO violation and resource fragmentation are all considered. Finally, the Multi-Stage Graph algorithm [12] is conducted to find the best VNF placement and chaining result.

- Eigendecomposition: uses Umeyama's eigendecomposition approach to achieve the optimal matching of a VNFFG in network topology. First, Eigendecomposition computes an adjacent matrix for network topology where the weight of each element is calculated using the widest-shortest path algorithm. Next, based on the demand of resource consumption, Eigendecomposition also computes an adjacent matrix for each SFCR. Then, the adjacent matrix of SFCR is extended to be with the same size of the network's. After that, Eigendecomposition computes the eigenvector matrixes of these two adjacent matrixes, then computes the conjugate matrixes and multiplies them together. Finally, Eigendecomposition chooses the locations with the maximum value in each row of the product to place VNFIs and construct the routing paths of SFCRs.

B. Simulation Results

*1) **Convergence of DDQN Models in Training Process**:* Fig. 3 shows the iteration times and time consumptions of DDQN models in training process. As stated, the time cycle of the trace-driven emulation of TOTEM project is 24 *h* and the time interval $\triangle t$ is set to 15 *min* [19], so there are 96 DDQN models trained in our simulation. And, we train each DDQN model using about 5×10^4 samples. Here, the iteration times mean how many times we sample training data from database to train a DDQN model, and the time consumptions denote the corresponding training time. In the simulation, all these DDQN models can converge, which means that DDQN models have found the hidden rules behind the training data to evaluate actions under current states. In these DDQN models, the minimum and maximum iteration times a DDQN model trained to be convergent are about 170 and 8,200, and the time consumptions are about 1.7 *s* and 82 *s*, respectively. The mean iteration times and the time consumption a DDQN model trained to be convergent in the simulation are about 2,100 and 21 *s*, respectively. Since the training time per DDQN model is short enough to be neglected, DDQN-VNFPA can achieve efficient model training and update.

*2) **Influence of Optimized Solution Space Size**:* This simulation shows the influence of the optimized solution space size in Fig.4. In DDQN-VNFPA, DDQN models are trained to preliminarily evaluate the performance of each action under current state, and the actions with top k Q-values are chosen to construct a optimized solution space. Then, we compute the best action according to forecasted SFCRs in the optimized solution space. In the figure, when k equals 5, the possibility to include the best action in the optimized solution space is about 87.5%, and the gap between the best reward and the reward of the action got from DDQN models is about 0.13. When k increases to 25, the possibility to include the best action in the optimized solution space becomes about 92.5%, and the gap to the best reward is reduced to about 0.1. This is because, when we increase optimized solution space size, there are more possibility to include the best action. Nevertheless, large optimized solution space size leads to more computation. So there exists a trade off between the optimized solution space size and computation complexity. As the performance improvement between $k = 15$ and $k = 25$ is little, all the next simulations are conducted with $k = 15$.

*3) **Comparison in Time Consumptions of Running Process**:* We compare the time consumptions of three algorithms in Fig. 5. Time consumption shows how long an algorithm optimizes the placement of VNFIs considering new arrival SFCRs. In the simulation, we assume that there are 50 new arrival SFCRs in a future time interval $\triangle t$. In Fig. 5, DDQN-VNFPA performs the best and its time consumption is only about 0.22 *s*. The time consumption of MSGAS is the longest which is about 45 *s*, and Eigendecomposition takes about 4.5 *s* to optimize the placement of VNFIs. In DDQN-VNFPA, the main time computation is to compute the best action which is about 0.21 *s*, and the solution space optimization only takes about 0.01 *s*. Compared with the future time interval $\triangle t$ ($\triangle t = 15$ *min* in the simulation), the time consumption of our proposed DDQN-VNFPA is small enough to achieve online VNF placement in SDN/NFV-enabled networks.

*4) **Comparison in SFCR Reject Number and Reject Ratio, Network Throughput and End-to-end Delay of SFCRs**:* The comparison of SFCR reject number and reject ratio among these three algorithms are shown in Fig. 6. In this simulation, our proposed DDQN-VNFPA gets the highest performance, and the performance of MSGAS does better than

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

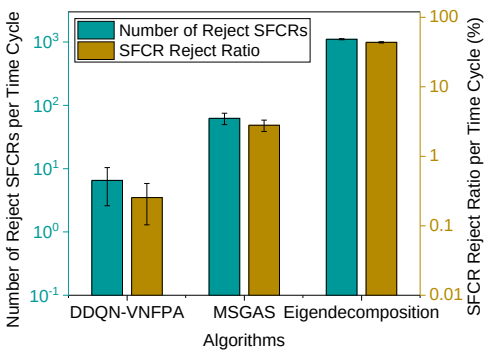


Fig. 6. Comparison in SFCR reject number and reject ratio (with 95% confidence intervals).

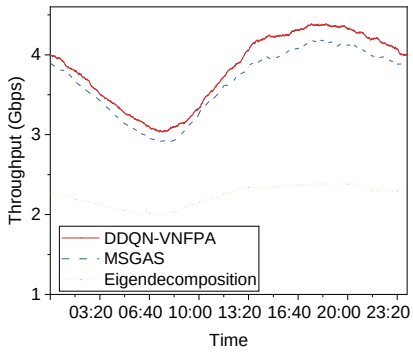


Fig. 7. Comparison in network throughput.

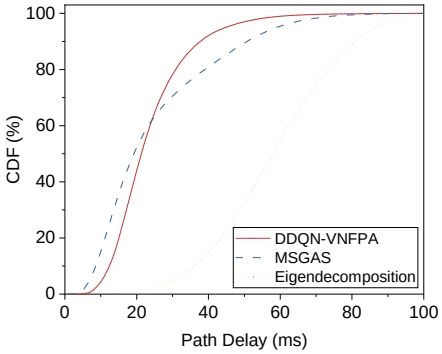


Fig. 8. Comparison in CDF of end-to-end delay.

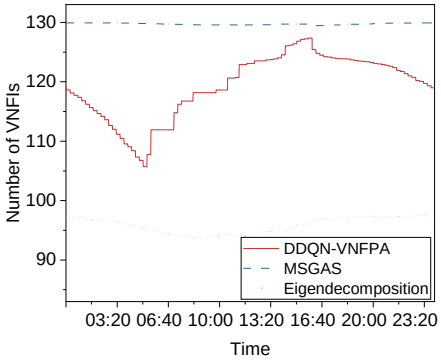


Fig. 9. Comparison in the number of VNFIs.

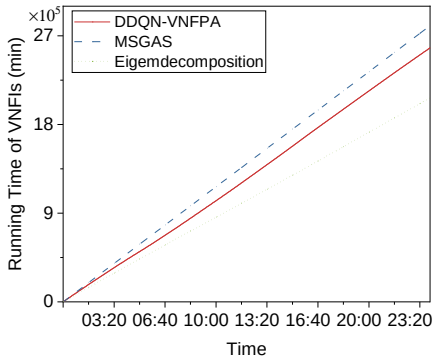


Fig. 10. Comparison in running time of VNFIs.

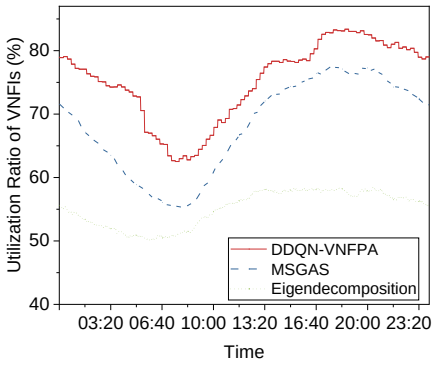


Fig. 11. Comparison in utilization ratio of VNFIs.

Eigendecomposition. In DDQN-VNFPA, the penalty of reject SFCRs are considered as the reward when training DDQN models, so the number of reject SFCRs per time cycle T is only about 6 which only accounts for about 0.25% of all the SFCRs. For MSGAS, the number of reject SFCRs per time cycle is about 70 and the SFCR reject ratio is about 2.8%. Eigendecomposition performs the worst in the simulation. The reason is that Eigendecomposition cannot guarantee to get the optimal matching of SFCRs in the network topology. Moreover, Eigendecomposition also neglects to minimize the number of reject SFCRs, and the widest-shortest path algorithm prefers to produce long routing paths for SFCRs, which leads to more resource consumptions. Thus, for Eigendecomposition, there are about 1,000 SFCRs cannot be served per time cycle, which accounts for about 40% of all the SFCRs.

Fig. 7 presents the network throughput during a time cycle. According to Fig. 6, since DDQN-VNFPA can efficiently reduce the number of reject SFCRs, it gets the best performance in network throughput as well. Compared with MSGAS, DDQN-VNFPA gets about 4% performance improvement in this simulation. Since the number of reject SFCRs of Eigendecomposition is much larger than that of DDQN and MSGAS, its network throughput is only about half of the other algorithms.

The path delay of the three algorithms are described in Fig. 8. Since the widest-shortest routing algorithm is used in Eigendecomposition, which prefers to generate long routing paths for traffic of SFCRs, the end-to-end delay of Eigendecomposition is much longer than that of the other algorithms. Compared with DDQN-VNFPA, in MSGAS, the end-to-end delay is considered in its objective, while DDQN-VNFPA does not consider to optimize the end-to-end delay of SFCRs. Therefore, MSGAS can generate more routing paths with shorter end-to-end delay for SFCRs. For example, in the figure, the number of paths with end-to-end delay shorter than 20 *ms* accounts for about 50%, while it is only about 40% in DDQN-VNFPA.

*5) **Comparison in the Number, Running Time and Utilization Ratio of VNFIs**:* We illustrate the number of VNFIs placed in the network over time in Fig. 9. In the simulation, as DDQN-VNFPA considers the change of network load and can dynamically place/release VNFIs in the network, the VNF placement can adapt to dynamic network load better than that of the other two algorithms. For example, according to the trace-driven emulation of TOTEM project in Fig. 1, the network load decreases between 4 pm – 6 am the next day and increases between 6 am – 12 am. When network load decreases, DDQN-VNFPA can release redundant placed VNFIs to reduce VNF running time. When network load increases, it places more VNFIs to guarantee that there are enough resources to provide for users. However, compared with DDQN-VNFPA, MSGAS and Eigendecomposition neglect to release VNFIs when network load decreases, so the number of placed VNFIs of these two algorithms cannot dynamically adapt to the change of network load. Additionally, though the number of VNFIs with Eigendecomposition is the smallest in the simulation, it leads to the worst network performance according to Fig. 6-8.

Fig. 10 shows the total VNFI running time per time cycle of these three algorithms. In the figure, we can find that, DDQN

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

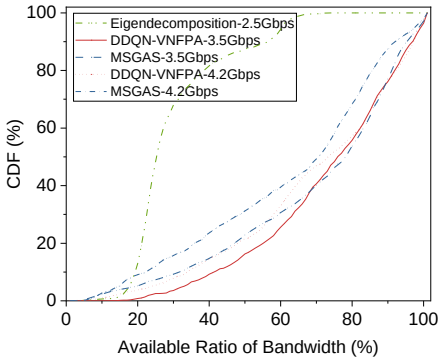


Fig. 12. Comparison in CDF of available bandwidth.

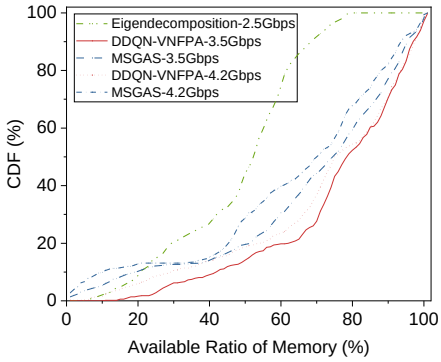


Fig. 13. Comparison in CDF of available memory.

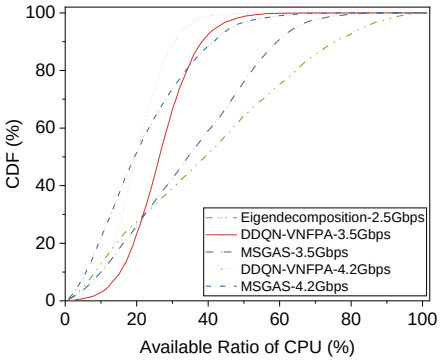


Fig. 14. Comparison in CDF of available CPU.

can reduce about 9% VNFI running time than MSGAS in a time cycle. As for Eigendecomposition, the number of placed VNFIs is the smallest according to Fig. 9, so it gets the lowest VNFI running time in this simulation.

In Fig. 11, the utilization ratio of VNFIs is compared among three algorithms. Since DDQN-VNFPA takes the VNF placement cost and running time of VNFIs into consideration, DDQN-VNFPA performs the best in this simulation. However, compared with DDQN-VNFPA, MSGAS and Eigendecomposition do not consider to improve the utilization ratio of VNFIs by releasing redundant VNFIs, thus in Fig. 11, when network load decreases between 4 pm – 6 am the next day, the utilization ratios of VNFIs with MSGAS and Eigendecomposition are about 8% and 25% lower than that of DDQN-VNFPA, respectively. And when network load increases between 6 am – 12 am, the utilization ratios of VNFI with MSGAS and Eigendecomposition are about 5% and 20% lower than that of DDQN-VNFPA, respectively.

*6) **Comparison in Resource Consumptions and Load Balancing**:* Fig. 12 shows the CDF of available bandwidth ratios of three algorithms. As the bandwidth cost is well considered using the SFC-MAP algorithm proposed in our previous work, when network throughput is 3.5 Gbps, there is no bottleneck link with available bandwidth ratio smaller than 10% in DDQN-VNFPA, while the bottleneck links account for about 2.3% in MSGAS. And the number of bottleneck links in DDQN-VNFPA is also smaller than MSGAS's, when network throughput is 4.2 Gbps. Additionally, as for available bandwidth ratio between 40% and 80%, the curve slope of DDQN-VNFPA is huger than that of MSGAS, meaning that DDQN-VNFPA is more bandwidth-efficient and does well in load balancing. As for Eigendecomposition, though the load in links is balancing, the bandwidth consumption is the highest leading to very low network performance compared with the other algorithms.

Fig. 13 presents the CDF of available memory ratios of three algorithms. When network throughput is 3.5 Gbps, the bottleneck nodes with available memory ratio less than 10% accounts for about 5.3% in MSGAS, while there is almost no nodes becoming bottleneck in DDQN-VNFPA. And the number of bottleneck nodes of MSGAS is about sixfold than DDQN-VNFPA's, when network throughput is 4.2 Gbps. Moreover, the curve of DDQN-VNFPA is lower than that of MSGAS, meaning that DDQN-VNFPA is more efficient in memory consumption than MSGAS. As for Eigendecomposition, its curve is the highest, which indicates that it leads to the highest memory consumption in nodes than the other algorithms

The CDF of available ratio of CPU in VNFIs is described in Fig. 14. When network throughput is 2.5 Gbps, the number of bottleneck VNFIs with available CPU ratio less than 10% in Eigendecomposition is about 13%. When the network throughput is 3.5 Gbps, the number of bottleneck VNFIs in DDQN-VNFPA is about 3%, which is about 10% in MSGAS. And when network throughput comes to 4.2 Gbps, there is about 9% of VNFIs becoming bottlenecks in DDQN-VNFPA, while the number is about 22% in MSGAS. Additionally, as for DDQN-VNFPA, when network throughput is 4.2 Gbps, there are about half of VNFIs with available CPU utilization ratio between 20% and 40%, however, there are only about 37% VNFIs for MSGAS. Thus, according to Fig. 12-14, DDQN-VNFPA gets the best performance in resource efficiency and load balancing compared with the other two algorithms.

VII. Conclusion

This paper studies the VNF placement problem considering dynamic network load in SDN/NFV-enabled networks. In order to solve the problem, we first formulate it as a BIP model aiming to minimize the total cost consisting of VNF placement cost, VNFI running cost and penalty of reject SFCRs. Next, a novel horizontal scheme, DDQN-VNFPA, is proposed to solve this problem in an intelligent manner. DDQN-VNFPA includes offline training and online running processes. In the training process, we collect training data and train DDQN models. Then, we conduct the running process of DDQN-VNFPA in three phases. The first phase uses these trained DDQN models to preliminarily evaluate actions with Q-values and achieve solution space optimization. The second phase is to further evaluate actions of optimized solution space by calculating their rewards considering forecasted SFCRs in simulation environment, and record results to further update DDQN models. In the third phase, we compute the best action with the highest reward in the optimized solution space, then optimize the VNF placement in the corresponding network regions according to a threshold-based policy. We have given a detailed analysis of DDQN-VNFPA, then constructed a Tensorflow-based environment and conducted a trace-driven

simulation to evaluate its performance. Evaluation results show that DDQN-VNFPA can get high performance in terms of reject number and reject ratio of SFCRs, throughput and end-to-end delay, save VNFI running time, improve VNF utilization ratio and achieve better load balancing compared with the algorithms in existing literatures.

As a future work, we plan to extend our work in a number of ways. We plan to construct a SDN/NFV-enabled networks, and run our DDQN-VNFPA to achieve the VNF placement in it. We plan to test DDQN-VNFPA with different network topologies and network characteristics to get more comprehensive performance evaluation results. We also plan to implement traffic forecasting methods in SDN/NFV-enabled networks and further evaluate the influence of the forecasting accuracy on DDQN-VNFPA.

REFERENCES

[1] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[2] D. Li, P. Hong, K. Xue, and J. Pei, "Availability aware vnf deployment in datacenter through shared redundancy and multi-tenancy," *IEEE Transactions on Network and Service Management*, 2019.

[3] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-openflow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.

[5] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, 2017.

[6] J. Qin, Y. Wu, Y. Chen, K. Xue, and D. S. Wei, "Online user distribution-aware virtual machine re-deployment and live migration in SDN-based data centers," *IEEE Access*, vol. 7, pp. 11152–11164, 2019.

[7] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[8] P. Quinn and T. Nadeau, "Problem statement for service function chaining," in *Informational RFC, RFC 7498*, 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7498.txt, accessed Sep. 20, 2019.

[9] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2179–2192, 2018.

[10] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proc. IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9, 2014.

[11] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Proc. IEEE International Conference on Cloud Networking*, pp. 255–260, 2015.

[12] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[13] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve service chaining performance with optimized middlebox placement," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 560–573, 2017.

[14] T. Kuo, B. Liou, K. C. J. Lin, and M. J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *Proc. IEEE International Conference on Computer Communications*, pp. 1–9, 2016.

[15] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[17] W. Cerroni and F. Callegati, "Live migration of virtual network functions in cloud-based edge networks," in *Proc. IEEE International Conference on Communications (ICC)*, pp. 2963–2968, 2014.

[18] B. Li, W. Lu, S. Liu, and Z. Zhu, "Deep-learning-assisted network orchestration for on-demand and cost-effective vnf service chaining in inter-dc elastic optical networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D29–D41, 2018.

[19] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 530–543, 2019.

[20] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 106–120, 2017.

[21] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[22] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, 2017.

[23] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual network function placement and resource optimization in nfv and edge computing enabled networks," *Computer Networks*, vol. 152, pp. 12–24, 2019.

[24] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual network function placement considering resource optimization and sfc requests in cloud datacenter," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1664–1677, 2018.

[25] D. Qi, S. Shen, and G. Wang, "Towards an efficient vnf placement in network function virtualization," *Computer Communications*, vol. 138, pp. 81–89, 2019.

[26] Y. Liu, J. Pei, P. Hong, and D. Li, "Cost-efficient virtual network function placement and traffic steering," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2019.

[27] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, 2019.

[28] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type vnf forwarding graphs in inter-dc elastic optical networks," *Journal of Lightwave Technology*, vol. 34, no. 14, pp. 3330–3341, 2016.

[29] K. Xue, Y. Xue, J. Hong, W. Li, H. Yue, D. S. Wei, and P. Hong, "RAAC: Robust and auditable access control with multiple attribute authorities for public cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 953–967, 2017.

[30] W. Li, K. Xue, Y. Xue, and J. Hong, "TMACS: A robust and verifiable threshold multi-authority access control system in public cloud storage," *IEEE Transactions on parallel and distributed systems*, vol. 27, no. 5, pp. 1484–1496, 2015.

[31] K. Xue, J. Hong, Y. Ma, D. S. Wei, P. Hong, and N. Yu, "Fog-aided verifiable privacy preserving access control for latency-sensitive data sharing in vehicular cloud computing," *IEEE Network*, vol. 32, no. 3, pp. 7–13, 2018.

[32] J. Hong, K. Xue, N. Gai, D. Wei, and P. Hong, "Service outsourcing in F2C architecture with attribute-based anonymous access control and bounded service number," *IEEE Transactions on Dependable and Secure Computing*, 2018.

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

[34] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach," *IEEE Transactions on Services Computing*, 2017. [Online]. Available: http://doi.org/10.1109/TSC.2017.2671867, accessed Sep. 20, 2019.

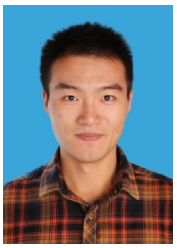
[35] V. Eramo, M. Ammar, and F. G. Lavacca, "Migration energy aware reconfigurations of virtual network function instances in nfv architectures," *IEEE Access*, vol. 5, pp. 4927–4938, 2017.

[36] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "Nfvdeep: adaptive online service function chain deployment with deep reinforcement learning," in *Proc. ACM International Symposium on Quality of Service*, 2019. [Online]. Available: http://doi.org/10.1145/3326285.3329056, accessed Sep. 20, 2019.

[37] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2959181, IEEE Journal on Selected Areas in Communications

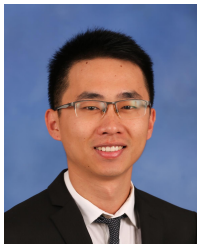
[38] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.
[39] P. Naik, D. K. Shaw, and M. Vutukuru, "Nfvperf: Online performance monitoring and bottleneck detection for nfv," in *Proc. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 154–160, 2016.
[40] F. H. Tseng, M. S. Tsai, C. W. Tseng, Y. T. Yang, C. C. Liu, and L. D. Chou, "A lightweight autoscaling mechanism for fog computing in industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4529–4537, 2018.
[41] "Tensorflow." [Online]. Available: https://www.tensorflow.org/, accessed Sep. 20, 2019.
[42] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," *Google Inc., White Paper*, pp. 1–14, 2011.
[43] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Proc. ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 23–29, 2002.
[44] J. Zhou, P. Hong, and J. Pei, "Multi-task deep learning based dynamic service function chains routing in sdn/nfv-enabled networks," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2019.
[45] J. Pei, P. Hong, K. Xue, and D. Li, "Resource aware routing for service function chains in sdn and nfv-enabled network," *IEEE Transactions on Services Computing, Online*, 2018. [Online]. Available: http://doi.org/10.1109/TSC.2018.2849712, accessed Sep. 20, 2019.
[46] R. Ramaswamy, N. Weng, and T. Wolf, "Characterizing network processing delay," in *Proc. IEEE Global Telecommunications Conference (Globecom)*, vol. 3, pp. 1629–1634, 2004.



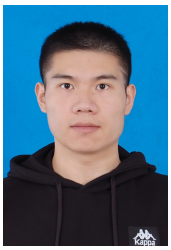
Jianing Pei was born in 1992. He received his B.S. degree from the Department of Information and Electrical Engineering, China University of Mining and Technology, in 2015. He is currently pursuing for his Ph.D in the Department of Electronic Engineering and Information Science, University of Science and Technology of China with his advisor Peilin Hong. His research interests include software-defined networks, network function virtualization, network resource orchestration and management and machine learning algorithms.

Peilin Hong was born in 1961. She received her B.S. and M.S. degrees from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 1983 and 1986. Currently, she is a Professor and Advisor for Ph.D. candidates in the Department of EEIS, USTC. Her research interests include next-generation Internet, policy control, IP QoS, and information security. She has published 2 books and over 100 academic papers in several journals and conference proceedings.

Miao Pan (S07-M12-SM18) received his BSc degree in Electrical Engineering from Dalian University of Technology, China, in 2004, MASc degree in electrical and computer engineering from Beijing University of Posts and Telecommunications, China, in 2007 and Ph.D. degree in Electrical and Computer Engineering from the University of Florida in 2012, respectively. He is now an Associate Professor in the Department of Electrical and Computer Engineering at University of Houston. He was a recipient of NSF CAREER Award in 2014. His research interests include cybersecurity, deep learning privacy, big data privacy, cyber-physical systems, and cognitive radio networks. His work won IEEE TCGCC (Technical Committee on Green Communications and Computing) Best Conference Paper Awards 2019, and Best Paper Awards in ICC 2019, VTC 2018, Globecom 2017 and Globecom 2015, respectively. Dr. Pan is an Associate Editor for IEEE Internet of Things (IoT) Journal from 2015 to 2018. He has also been serving as a Technical Organizing Committee for several conferences such as TPC Co-Chair for Mobiquitous 2019, ACM WUWNet 2019. He is a member of ACM and a senior member of IEEE.



Jiangqing Liu received the Ph.D. degree from University of Florida in 2018 and the B.Eng. degree from University of Electronic Science and Technology of China in 2013. He is currently a tenure-track assistant professor in the Department of Electrical and Computer Engineering at University of Alabama in Huntsville. His research interest is to apply cryptography, differential privacy and convex optimization to design secure and efficient protocols for various IoT systems. He is the recipient of the 2018 Best Journal Paper Award from IEEE Technical Committee on Green Communications & Computing (TCGCC) and the Best Paper Award from 2012 IEEE Workshop on Microwave and Millimeter-Wave Circuits and Systems (MMWCST).



Jingsong Zhou was born in 1995. He received his B.S. degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2018 and he is presently working on his Master's Degree in USTC with his advisor Peilin Hong. His research interests are involving SDN, NFV, and quality of service, particularly with applications of machine intelligence and deep learning.