A-DDPG: Attention Mechanism-based Deep Reinforcement Learning for NFV

He, Nan; Yang, S.; Li, Fan; Trajanovski, S.; Kuipers, F.A.; Fu, Xiaoming

DOI
10.1109/IWQOS52092.2021.9521285

Publication date
2021

Document Version
Accepted author manuscript

Published in
IWQoS 2021 - IEEE/ACM International Symposium on Quality of Service

Citation (APA)
He, N., Yang, S., Li, F., Trajanovski, S., Kuipers, F. A., & Fu, X. (2021). A-DDPG: Attention Mechanism-based Deep Reinforcement Learning for NFV. In *IWQoS 2021 - IEEE/ACM International Symposium on Quality of Service* Article 9521285 IEEE. https://doi.org/10.1109/IWQOS52092.2021.9521285

A-DDPG: Attention Mechanism-based Deep Reinforcement Learning for NFV

Nan He*, Song Yang*, Fan Li*, Stojan Trajanovski†, Fernando A. Kuipers‡, Xiaoming Fu§
* School of Computer Science and Technology, Beijing Institute of Technology, China
† Microsoft, London, United Kingdom
‡ Embedded and Networked Systems group, Delft University of Technology, The Netherlands
§ Institute of Computer Science, University of Göttingen, Germany
Email: {henan, S.Yang, fli}@bit.edu.cn, sttrajan@microsoft.com, F.A.Kuipers@tudelft.nl, Fu@cs.uni-goettingen.de

***Abstract*—The efficacy of Network Function Virtualization (NFV) depends critically on (1) where the virtual network functions (VNFs) are placed and (2) how the traffic is routed. Unfortunately, these aspects are not easily optimized, especially under time-varying network states with different quality of service (QoS) requirements. Given the importance of NFV, many approaches have been proposed to solve the VNF placement and traffic routing problem. However, those prior approaches mainly assume that the state of the network is static and known, disregarding real-time network variations. To bridge that gap, in this paper, we formulate the VNF placement and traffic routing problem as a Markov Decision Process model to capture the dynamic network state transitions. In order to jointly minimize the delay and cost of NFV providers and maximize the revenue, we devise a customized Deep Reinforcement Learning (DRL) algorithm, called A-DDPG, for VNF placement and traffic routing in a real-time network. A-DDPG uses the attention mechanism to ascertain smooth network behavior within the general framework of network utility maximization (NUM). The simulation results show that A-DDPG outperforms the state-of-the-art in terms of network utility, delay, and cost.**

***Index Terms*—Network function virtualization, deep reinforcement learning, placement, routing**

I. Introduction

Traditionally, Network Functions (NFs), such as firewalls and load balancers, are implemented on physical devices, called middleboxes, which are costly, lack flexibility, and are difficult to operate. Network Function Virtualization (NFV) has emerged as an innovative technique that can deal with these challenges by decoupling network functions from dedicated hardware and realizing them in the form of Virtual Network Functions (VNFs) [1], [2]. Because this technique shows great potential in promoting openness, innovation, flexibility, and scalability of networks, NFV attracts a good deal of interest from the networking community [3]–[5]. To build more complex services, the notion of Service Function Chaining (SFC) can be used, where a sequence of VNFs must be processed in a pre-defined order to collectively deliver a certain service. Therefore, an important problem is to determine the positions for placing VNFs and select the paths for routing traffic, such that the service requirement can be satisfied. The problem of VNF placement and traffic routing is referred to as VNF-PR in this paper. In solving the VNF-PR problem, service providers typically strive for network utility maximization (NUM). Therefore, jointly considering cost and QoS (e.g., delay) schemes is required, which will lead to a better user experience and higher profit.

Existing works on the VNF-PR problem is either based on linear programming [6], [7] or the VNF-PR problem is translated into some well-known NP-hard problems [8] such as the knapsack problem, and then a heuristic or approximation method is proposed to solve it [9]–[11], at the expense of ignoring the network state dynamics. For better performance, existing works formulate a one-shot optimization problem in a dynamic environment [12], and some works consider the VNF-PR problem over the entire system lifespan [13]. However, they only focus on the revenue for NFV operators and do not pay attention to the network utility consisting of revenue and cost.

Another line of work applies (Deep) Reinforcement Learning (DRL) [14] to solve the VNF-PR problem [15]–[17]. More specifically, to solve the VNF-PR problem, an RL agent interacts with the real-time NFV-enabled environment through the implementation of placement and routing strategies. Subsequently, the RL agent continuously optimizes the strategies according to the reward value of the environment feedback (e.g., delay, capacity, and overhead). However, given the large state space involved, RL methods become impractical and inefficient for large networks. On the contrary, deep neural networks can be applied to high-dimensional state space. Different from these approaches, in this paper, we leverage the feature of deep neural networks and introduce a Markov Decision Process (MDP) to capture the dynamic network state transitions and process them within a DRL architecture. We adopt a Deep Deterministic Policy Gradient (DDPG) [18] algorithm to deal with the high-dimensional and time-varying

Song Yang is the corresponding author.

The work of Song Yang is partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61802018 and the Beijing Institute of Technology Research Fund Program for Young Scholars. The work of Fan Li is partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62072040, 61772077, and Beijing Natural Science Foundation under Grant No. 4192051. The work of Xiaoming Fu is partially supported by the EU H2020 RISE COSAFE project (No. 824019).

network state and complex network environment.

Typically, a DRL agent will not pay equal attention to all the available placement nodes. The agent usually chooses the current action based on information of higher levels of cognitive skills and ignores other perceivable information. In this paper, we introduce the concept of attention mechanism, which is widely used in neural image caption generation [19] to simulate the agent's action for DDPG. We find that during the training process of DDPG, the attention mechanism will automatically focus on the feasible neighbor node that may affect the agent's selection behavior. It ultimately helps to reduce the attention to other unnecessary nodes and improve the training efficiency of the model. With this motivation, we design a customized attention mechanism-based DDPG to train our DRL model.

The main contributions of this paper are as follows:

- We formulate the VNF-PR problem as an optimization model and establish a utility function aiming to trade off between revenue and cost.
- We propose a novel Attention mechanism-based Deep Deterministic Policy Gradients (A-DDPG) framework, using the Actor-Critic network structure, in which both the Actor and Critic networks adopt double networks (namely the main network and the target network).
- Through extensive simulation experiments, we show that our A-DDPG framework outperforms the state-of-the-art in terms of network utility, delay, and cost.

The remainder of this paper is organized as follows. Section II defines the VNF-PR problem. In Section III, we devise the A-DDPG algorithm to solve the VNF-PR problem. Section IV provides the simulation results. Section V describes related work and we conclude in Section VI.

II. Model and Problem formulate

In this section, we begin with describing the network utility model in Section II-A and then we formulate the VNF-PR problem with the objective and constraints in Section II-B. For the convenience of reading, we summarize the notations used in this paper in Table I.

TABLE I
NOTATIONS.

Variable	Definition
$\mathcal{G}$	Physical network
$\mathcal{N}$	The set of nodes of the network
$\mathcal{E}$	The set of links of the network
$\mathcal{R}$	The set of request. For each $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, ξ indicates the flow rate, $\mathcal{F}$ represents a set of requested VNFs, D denotes the requested delay
$\mathcal{F}_r$	The set of requested VNFs of $r \in \mathcal{R}$
$u^r_{s,a}, u^c_{s,a}$	The revenue function and cost function
T^r	The total delay of network request $r \in R$
C^f_n	Capacity demand for $f \in \mathcal{F}_r$ on node $n \in N$
$C_{u,v}$	The link capacity demand between u and v
d^f_n	Delay demand for $f \in \mathcal{F}_r$ on node n
$d_{u,v}$	Delay demand between u and v
$\delta_n,\ \eta_e$	Capacity of node $n \in \mathcal{N}$ and link $e \in \mathcal{E}$
Ψ	The expected service payment from consumers
$\Phi^{op}, \Phi^{de}, \Phi^{tr}$	The unit operation cost, unit deployment cost and unit transmission cost, respectively
S, A, R	The state space, action space, and reward, respectively
$x^{r,f}_n$	A Boolean variable. It is 1 if r's requested VNF f is placed on n; and 0 otherwise
$y^r_{u,v}$	A Boolean variable. It is 1 if path between u and v is used for delivering the requested task of r; and 0 otherwise
z_r	A Boolean variable. It is 1 if r is accepted; and 0 otherwise
k_i, v_i, q_i	The key, value and query for node $i \in \mathcal{N}$
sc^j_i	The compatibility of query q_i with key k_j
θ^μ *and* θ^Q	The weights of actor and critic networks
$\mathbb{N}_{\boldsymbol{\tau}}$	The distribution with a time τ for the exploration noise

A. Network Utility Model

Firstly, we consider a physical network which is presented as a graph $\mathcal{G} = \{\mathcal{N},\ \mathcal{E}\}$, where $\mathcal{N}$ and $\mathcal{E}$ stand for the node set and the link set, respectively. We mainly consider two kinds of resource constraints, including node and link resource constraints. Each node $n \in \mathcal{N}$ has a capacity of δ_n (i.e., CPU cycles per second) and a delay of d_n. Each link $e \in \mathcal{E}$ has a capacity of η_e and a delay d_e. We use $\mathcal{R}$ to represent a set of $|\mathcal{R}|$ requests, and each $r_l(\xi, \mathcal{F}, D) \in \mathcal{R}$ has multiple VNFs that are used in sequence, where ξ indicates the flow rate, $\mathcal{F}$ represents a set of requested VNFs, and D denotes the requested delay. We define the VNFs set as $\mathcal{F}_r = \{f_1, f_2, ..., f_K\}$. Each VNF $f \in \mathcal{F}_r$ on node $n \in \mathcal{N}$ requires time of D^f_n to process it. Before SFC, we define k as a constant in the range $(0, K)$. When all VNFs in SFC are successfully placed and routed, $k = K$. We use the high-order matrix $K * \mathcal{N}$ to represent the deployment status of VNF on a physical server.

The total network utility to serve a request r consists of revenue and cost. More specifically, we define the utility function U^r of request r as:

$$U^r = u^r_{s,a} - u^c_{s,a} \tag{1}$$

where $u^r_{s,a}$ is the revenue function, and $u^c_{s,a}$ is the total cost. We use the concept of Shannon's entropy [20] and define the revenue function as:

$$u^r_{s,a} = \sum_{r \in \mathcal{R}} z_r \cdot \xi \cdot \Psi - \sum_{r \in \mathcal{R}} (-\frac{1}{T^r} \log \frac{1}{T^r}) \tag{2}$$

where ξ represents the traffic of the request r, and Ψ is the expected service revenue from consumers according to the SLA [21]. T^r represents the total delay of network request r, and it is the sum of the processing delays of all nodes.

The purpose of using information entropy is to unify service revenue and delay that ensures the additivity between data. The transmission delay in the SFC is defined as:

$$T^r = \sum_{f_i \in \mathcal{F}_r} x_n^{r,f} \cdot d_n^f + \sum_{e^{u,v} \in \mathcal{E}} y_{u,v}^r \cdot d_{u,v} \quad \forall u, v, n \in \mathcal{N} \quad (3)$$

where d_n^f represents the delay demand for $f \in \mathcal{F}_r$ on node n, $d_{u,v}$ indicates the delay demand between nodes u and v.

The cost function $u_{s,a}^c$ includes three parts: operation cost, deployment cost and transmission cost.

1) Operation cost: Each physical node needs to complete the preparatory work before deploying VNFs, such as the pre-configuration of different types of VNFs. We define the unit operating cost as Φ^{op}, and then the total operation cost is defined as:

$$u_{s,a}^{op} = \sum_{n \in \mathcal{N}} x_n^{r,f} \cdot \Phi^{op} \quad \forall r \in \mathcal{R}, f \in \mathcal{F}_r \quad (4)$$

2) Deployment cost: The deployment cost of a server is directly proportional to the resources consumed. Therefore, we stipulate that VNF deployment cost is mainly generated by the server of the deployed function. If no VNF is placed on the server, the deployment cost is not considered. We define the unit deployment cost as Φ^{de}. The total deployment cost is defined as:

$$u_{s,a}^{de} = \sum_{n \in \mathcal{N}} x_n^{r,f} \cdot \Phi^{de} \quad \forall r \in \mathcal{R}, f \in \mathcal{F}_r \quad (5)$$

3) Transmission cost: The transmission cost is the communication cost for transferring traffic between nodes. In practice, the deployment cost is negatively correlated with the transmission cost [22]. When the number of VNFs is reduced to save on deployment cost, the average transmission cost of the network will increase. We define the transmission unit cost as Φ^{tr}. The total transmission cost is defined as:

$$u_{s,a}^{tr} = \sum_{e \in \mathcal{E}} y_{u,v}^r \cdot \xi_r^e \cdot \Phi^{tr} \quad \forall r \in \mathcal{R}, u, v \in \mathcal{N} \quad (6)$$

Finally, $u_{s,a}^c$ is a combination of above mentioned three kinds of cost:

$$u_{s,a}^c = u_{s,a}^{op} + u_{s,a}^{de} + u_{s,a}^{tr} \quad (7)$$

B. Problem Definition and Formulation

After the VNF is placed on a physical node in $\mathcal{G}$, the path for link mapping follows the order of the SFC, to reduce the dimension of the action space and solve the problem appropriately. Formally, the VNF Placement and traffic Routing (VNF-PR) problem can be defined as follows:

Definition 1: Given are a network $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ and a set of requests R, for each request $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, the VNF-PR problem is to place the VNFs on N and to route the traffic in a specific order, such that the network utility $\sum_{r \in R} U^r$ is maximized.

The VNF placement and traffic routing problem is known to be NP-hard [8]. We first formally present the VNF-PR problem with objectives and constraints. We begin with some necessary variables.

Boolean Variables:

$x_n^{r,f}$: It is 1 if r's requested VNF f is placed on n; and 0 otherwise.

$y_{u,v}^r$: It is 1 if a path between u and v is used for delivering the requested task of r; and 0 otherwise.

Objective:

$$\max \sum_{r \in R} U^r \quad (8)$$

Placement Constraints:

$$\sum_{n \in \mathcal{N}} x_n^{r,f} = 1 \quad \forall r \in \mathcal{R}, f \in \mathcal{F}_r \quad (9)$$

Node Capacity Constraints:

$$\sum_{r \in \mathcal{R}} \sum_{f \in r} x_n^{r,f} \cdot C_n^f \leq \delta_n \quad \forall n \in \mathcal{N} \quad (10)$$

Link Capacity Constraints:

$$\sum_{r \in \mathcal{R}} \sum_{e \in \mathcal{E}} y_{u,v}^r \cdot C_{u,v} \leq \eta_e \quad \forall u, v \in \mathcal{N} \quad (11)$$

Delay Constraints:

$$\sum_{f \in r} \sum_{n \in \mathcal{N}} x_n^{r,f_i} \cdot d_n^f + \sum_{u,v \in \mathcal{N}} \sum_{e^{u,v} \in \mathcal{E}} y_{u,v}^r \cdot d_{u,v} \leq D \quad \forall r \in \mathcal{R} \quad (12)$$

Eq. (8) maximizes the network utility. Eq. (9) ensures that for each requested VNF f, it must be placed on one node in the network. Eq. (10) indicates that each node's capacity is not violated. Eq. (11) ensures that each link's load is not greater than η_e. Eq. (12) ensures that the total delay for each request does not exceed D.

Without considering the variability of network states, the optimization problem can be solved by using ILP or heuristic algorithms [8]. However, it is non-trivial to use those techniques to model real-time metrics. DRL can capture the dynamic states of networks. Therefore, in Section III, we exploit DRL to solve the VNF placement and traffic routing problem.

III. Deep reinforcement learning

In this section, we begin with the DRL model design in Section III-A. This is a Markov decision process including state, action, and reward. Then we propose our A-DDPG framework to solve the VNF-PR problem in Section III-B.

A. DRL Model Design

In the DRL model, three elements, which are based on a Markov decision process, can be described by a tuple (S, A, R), referring to the state space, action space, and reward, respectively. To deal with the real-time network state changes caused by VNF-PR, we consider a discrete-time period T. From state S, after taking action A, the agent transfers to the next state S', and generates a return R (reward

or penalty) that guides the DRL. Then the agent makes new decisions and the procedure repeats. We define the 3-tuple (S, A, R) for the VNF-PR problem as follows:

State: The state space can be described by a vector $S = \{s_1, s_2, s_3, ..., s_T\}$, where each term $s_t \in S$ represents the remaining resources of the virtual links and nodes at time t. T represents a time period.

Action: The action of any agent is a vector A with each term $a \in A$ representing VNF placement and traffic routing. Therefore, we define an action as $a = \{x_n^{r,f},\ \xi_r^f\}\quad \forall r \in \mathcal{R}, f \in \mathcal{F}_r,\ n \in \mathcal{N}$.

Reward: The value of the reward is a value indicating correct action. Whether the action can bring profit and whether the user's demand is met is taken as the criteria to affect the reward value. The reward received at time-slot t is set as the objective of our utility function, defined as $R = U^r = u_{s,a}^r - u_{s,a}^c$ according to Eq. (1). If the action brings benefits to the network and saves cost, the reward will be a positive value to encourage the operation. However, if the cost increases or the constraint is violated, a negative reward is returned.

B. A-DDPG Framework

Incorporating the above definitions, we start to design our A-DDPG framework. We first present the attention model in Section III-B1. Then, we describe the Actor-Critic network in Section III-B2. Finally, we complete the algorithm in Section III-B3.

1) Attention Model: We argue that the neighbor nodes of each server node are of great significance to the performance of VNFs placement and routing. Therefore, we introduce an attention mechanism into the neural network, which allows the network to better obtain neighbor node information. The attention mechanism assumes that weights of nodes measure matching degree between neighbors in the attention layer.

Formally, we define h_i as the state of the node, and the key k_i, value v_i, and q_i can be calculated as follows:

$$k_i = W^K \cdot h_i, v_i = W^V \cdot h_i, q_i = W^Q \cdot h_i \quad \forall i \in \mathcal{N} \tag{13}$$

where W^Q, W^K, and W^V are the parameter matrices that can be learned, and h_i is equal to s_i defined in section III. The compatibility sc_i^j of the query q_i of node i with the key k_j of node j is calculated as the active function (e.g., dot-product):

$$sc_i^j = active(q_i, k_j) \quad \forall i, j \in \mathcal{N} \tag{14}$$

According to Eq. (14), we compute the weights a_i^j using a softmax function:

$$a_i^j = softmax(sc_i^j) = \frac{e^{sc_i^j}}{\sum_{i=1}^{N} e^{sc_i^j}} \quad \forall i, j \in \mathcal{N} \tag{15}$$

Then the attention value is equal to:

$$at((k_i, v_i), q_i) = \sum_{i=1}^{\mathcal{N}} a_i^j \cdot v_i = \sum_{i=1}^{\mathcal{N}} \frac{e^{sc_i^j}}{\sum_{i=1}^{\mathcal{N}} e^{sc_i^j}} \cdot v_i \quad \forall i, j \in \mathcal{N} \tag{16}$$

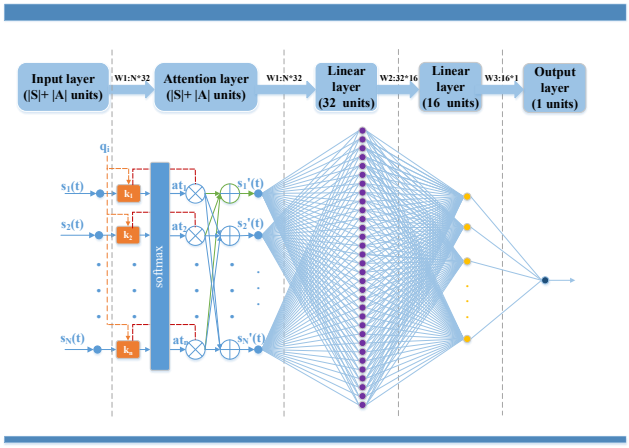


Fig. 1. Actor-Critic Network Design of our A-DDPG Framework.

2) Actor-Critic Network Design: Our A-DDPG algorithm constructs a deep reinforcement network fitting state-action value function to solve the state space explosion problem. We use the Actor-Critic network structure, in which the Actor and Critic networks both adopt double-networks, namely the main network and the target network. Therefore, our framework has four networks for DRL agent training to solve the VNF-PR problem. The four network structures are the same, as shown in Fig. 1. In the Actor-Critic network, the observation state and action are taken as the input, and two-layer hidden networks with 32 and 16 neurons are used to process the input to understand the deployment status of the physical server in the current network. Then the processing results are imported into a full connection layer with $Relu$ function. The $Relu$ function is a non-linear activation function in neural network. The network uses the cumulative reward as the target value and the expected cumulative reward as the predicted value. The purpose of training is to make the predicted value as close as possible to the target value. The equation to define the loss function is as follows:

$$L(\theta) = \frac{1}{B}\sum_t (y_t - Q(s_t, a_t|\theta^Q)^2) \tag{17}$$

where θ represents the parameter of actor for sampling and B indicates the size of the replay buffer. Then, the partial derivative of the loss function to the weight of the neural network can be calculated as:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{1}{B}\sum_t (y_t - Q(s_t, a_t|\theta^Q)^2) \frac{\partial Q(s_t, a_t|\theta^Q)}{\partial \theta} \tag{18}$$

where $Q(s_t, a_t|\theta^Q)$ refers to the long-term return of an action, taking a specific a_t under a specific policy from the current state s_t, and y_t denotes the predicted return. Through multiple iterations of the gradient descent method and the back-propagation mechanism, the $Q(s_t, a_t|\theta^Q)$ value can be obtained.

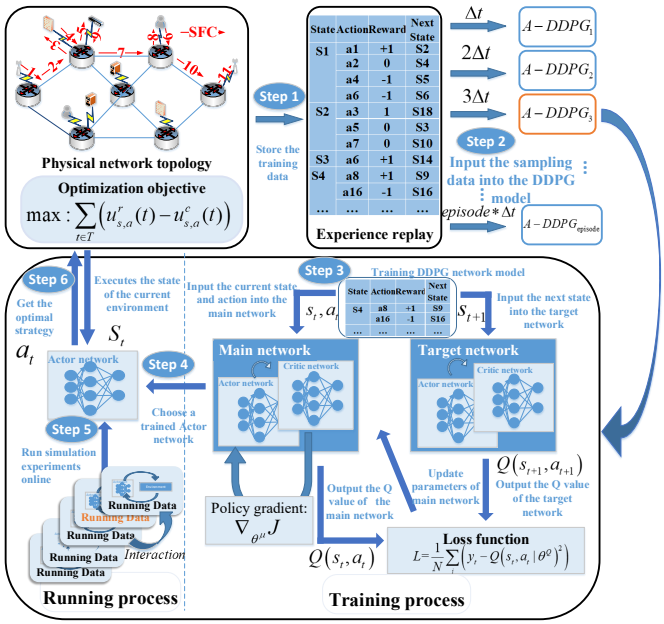


Fig. 2. The A-DDPG Framework.

3) Algorithm Design: The A-DDPG framework is divided into three main processes: observation process, training process, and running process, as shown in Fig. 2.

a) Observation process: The whole observation process mainly consists of two parts: Step 1 and Step 2 in Fig. 2.

Step 1, which is the initial phase of observation, begins with the agent interacting with the environment. The step is mainly used to obtain the initial state and store the historical samples. More specifically, the agent obtains the original state of the environment (including the routing status of the server, etc.) and collects environmental historical samples that need to be trained. These samples contain a sequence composed of the initial state s_t, action a_t, reward r_t, and the next state s_{t+1}. Then the samples are put into the replay memory (Step 1). Subsequently, the action is obtained according to the ϵ greedy strategy (since the neural network parameters are also randomly initialized, the parameters will not be updated at the step, and they are collectively called random actions). Next, ϵ is reduced according to the number of iterations. Afterwards, the simulator performs the selecting action and returns a new state and reward.

Step 2 of the observation process begins in the replay buffer. The samples of the replay buffer must be independent and identically distributed. However, the adjacent training samples of RL are related to each other. Therefore, an experience replay and target network are introduced into the network to break up the correlation. More specifically, the previous state s_t, action a_t, new state s_{t+1}, and reward r_t are assembled into (s_t, a_t, r_t, s_{t+1}) to enter the replay memory for parameter updating. Finally, the action to be executed next is selected according to the ϵ greedy strategy, and the cycle is repeated until the number of iterations reaches the limitation (depending on the size of the replay buffer).

Algorithm 1 A-DDPG Training Procedure

1: Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
2: Initialize target network $Q^{'}$ and $\mu^{'}$ with $\theta^{Q^{'}}$, $\theta^{\mu^{'}}$
3: Initialize replay buffer R
4: **for** $episode = 0, 1, ..., M$ **do**
5: Initialize a random process $\mathbb{N}$, choose a time $\boldsymbol{\tau}$ to get a distribution $\mathbb{N}_{\boldsymbol{\tau}}$ for action exploration
6: Receive initial observation state s_1
7: **for** $t = 0, 1, ..., T$ **do**
8: Select action $a_t = \mu(s_t|\theta^\mu) + \rho$ according to the current policy θ^μ and exploration noise ρ, where ρ is randomly chosen from $\mathbb{N}_{\boldsymbol{\tau}}$
9: Execute action a_t and observe reward r_t and observe new state s_{t+1}
10: Store a random minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from replay buffer
11: Set $y_t = r_t + \gamma Q^{'}(s_{t+1}, \mu^{'}(s_{t+1}|\theta^{\mu^{'}})|\theta^{Q^{'}})$
12: Update critic by minimizing the loss: $L = \frac{1}{B}\sum_t (y_t - Q(s_t, a_t|\theta^Q)^2)$
13: Update the actor policy using the sampled gradient: $\bigtriangledown_{\theta^\mu} J \approx \frac{1}{B}\sum \bigtriangledown_a Q\left(s_t, a_t|\theta^Q\right) \bigtriangledown_{\theta^\mu} \mu\left(s|\theta^\mu\right)|_{s_t}$
14: Update the target network: $\theta^{Q^{'}} \leftarrow \tau\theta^Q + (1-\tau)\,\theta^{Q^{'}}$, $\theta^{\mu^{'}} \leftarrow \tau\theta^\mu + (1-\tau)\,\theta^{\mu^{'}}$
15: **end for**
16: **end for**

b) Training process: After the observation process, the sufficient samples required for A-DDPG training are obtained. Algorithm 1 describes the training process of Step 3 in Fig. 2. More specifically, the agent first initializes the weights of the critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ as θ^Q and θ^μ, respectively (Line 1), and the target networks are cloned from the critic and actor networks (Line 2). Then a batch of data is sampled from the replay memory R as the input parameters of the A-DDPG model (Line 3). During the m-th episode (Line 4), in order to increase the randomness of the training process and the coverage of learning, the model adds random noise to the selected action, the noise is simulated by $\mathbb{N}_{\boldsymbol{\tau}}$ (Line 5), and the agent receives the first state s_0 of the current environment (Line 6). During the t-th time-slot (Line 7), A-DDPG adds exploration noise to the current policy (Line 8). Next, the agent executes action a_t and transfers to next state s_{t+1}, rewards r, and decides whether to terminate the state (Line 9). Subsequently, the agent stores the quadruple (s_t, a_t, r_t, s_{t+1}) into the experience replay B (Line 10). The next step is to update the actor network and the critic network. First we need to prepare significant training data: (1) calculate the predicted baseline y_t (Line 11) and (2) calculate the policy $\mu(s_t|\theta^\mu) + \rho$ and Q baseline $Q(s_t, a_t|\theta^Q)$. Drawing on the DDPG method, the value loss function is the mean square error (MSE) of the predicted baseline and the actual baseline according to (17)

(Line 12). Afterwards, gradient descent (18) is used to train the neural network (Line 13), and the weight parameters of the network are updated regularly (Line 14). If s_{t+1} is the terminal state, the current round of iteration is finished, otherwise, it goes to Line 8. The total number of episodes is denoted by M and each training episode contains T training rounds. The above procedures iterate until convergence or reaching the predefined episode bound.

c) Running process: The whole running process of the A-DDPG algorithm mainly consists of three parts: Step 4, Step 5, and Step 6 in Fig. 2.

In Step 4, the well-trained A-DDPG model is selected, and the long-term cumulative reward of the action is preliminarily evaluated by inputting the current state. The purpose is to avoid operations with poor performance and statistically select operations that may achieve good performance to optimize the solution space size. In Step 5, the performance of each action in the optimized solution space, based on the predicted value in the simulated environment, is evaluated to obtain rewards, and the results are recorded in the database to further update the A-DDPG model. In Step 6, the action with greatest reward is performed in the physical network.

IV. Performance evaluation

A. Simulation Settings

The simulation experiments are all implemented on an Intel (R) Core (TM) i7 Windows 10 64-bit system. We conduct simulations on a 50-node network: We generate 50 nodes in the network and draw a link for each node pair. Moreover, the network parameters, computing capabilities, and traffic requests are randomly generated similar to existing works [23], [24]. The node capacity is randomly distributed in $[1, 100]$. For each link, its capacity is randomly assigned from the range $[2, 4]$ Gb/s and its delay takes value in $[30, 50]$ ms. We simulate $[10, 100]$ requests and each request requires an SFC consisting of 3 to 6 different VNFs (e.g., firewall, NAT, IDS, load balancer, WAN optimizer and flow monitor) according to [25]. For the unit cost of Eqs. (4), (5), (6), we set $\Phi^{op} = 0.2$, $\Phi^{de} = 0.4$, and $\Phi^{tr} = 0.1$.

We set our attention-based deep neural network structure with an input layer, an output layer, and 3 hidden layers. The 3 hidden layers comprise an attention layer and 2 fully connected layers. The number of hidden nodes of the 2 fully connected layers is 32 and 16, respectively. The hyperparameters for DRL are shown in Table II, and the target network parameters are updated once every 200 steps.

TABLE II
HYPERPARAMETERS FOR DRL.

Replay buffer size	10000	Learning rate	$0.1, 0.01, 0.001$
Hidden nodes	$32, 64$	Number of episodes	3000
Discounted factor	0.8	Hidden layer	3

The implementation of the A-DDPG algorithm is divided into three modules. The first is the construction of the underlying network environment, including the simulation of network topology nodes and link resources. Next, the request generation module. Each request contains an SFC and each SFC contains 3 to 6 VNFs. Finally, the DRL algorithm module runs the A-DDPG algorithm. Once the agent is well-trained after convergence, it can make the right decision for the VNF-PR problem.

We compare our A-DDPG method with three counterpart algorithms: DDPG, NFVdeep [26], and Q-learning.

DDPG: DDPG is a model-free DRL algorithm to solve the VNF-PR problem. The difference with A-DDPG is that it does not add an attention mechanism. For each request $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, the DDPG algorithm tries to place $f \in \mathcal{F}_r$ on node $n \in \mathcal{N}$ only considering its current remaining resource capacity, regardless of the neighbors' states. Moreover, it considers actor and critic networks with two fully connected layers, in which the number of nodes is 32 and 16, respectively. Meanwhile, we set $\alpha = 0.01$, the batch size is 64 and $\gamma = 0.8$, which is consistent with the parameter settings of A-DDPG.

NFVdeep: NFVdeep is a state-of-the-art method for VNF-PR problems. NFVdeep methods are a type of RL technique that relies upon optimizing parameterized policies concerning the expected return (long-term cumulative reward) by gradient descent. The policy gradient of the parameter θ is defined as:

$$\bigtriangledown_\theta J(\theta) = \left(\frac{\partial \mathcal{J}(\theta)}{\partial \theta_1}, ..., \frac{\partial \mathcal{J}(\theta)}{\partial \theta_{\mathfrak{n}}} \right) \tag{19}$$

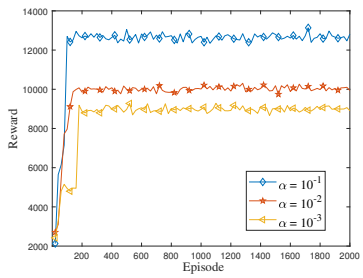
where the parameter θ is updated as $\theta_{i+1} = \theta_i + \alpha \bigtriangledown_{\theta_i} \mathcal{J}(\theta_i)$, α is the learning rate and $\mathfrak{n}$ is the number of neurons. During the training process, the agent processes one VNF of SFC in each MDP state transition. Then the reward for each state s is calculated, and the physical network gives the reward to the NFVdeep agent. Subsequently, the NFVdeep agent is trained for updating the policy circularly until the reward converges.

Q-learning: Q-learning is an off-policy RL method where the agent queries the Q-value table to make decisions. For each request $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, the Q-learning algorithm tries to place $f \in \mathcal{F}_r$ on a well-resourced node $n \in \mathcal{N}$, such that the total delay of r shall be less than D. Afterwards, the agent calculates the cumulative reward by the utility function of the network. We set the learning rate to 10^{-2} and halved it every 200 episodes. Meanwhile, we set $\gamma = 0.8$ to ensure the best performance of the algorithm. The Q-value is updated as follows:

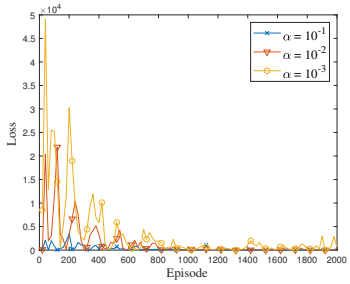
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{20}$$

where s represents the state at a certain moment, a_t indicates the action taken at that moment, $Q(s_t, a_t)$ denotes the Q-value corresponding to the (state, action) pair, r reflects the reward function, $Q(s_{t+1}, a_{t+1})$ represents the state transition function, and a_{t+1} denotes the action corresponding to the next state.

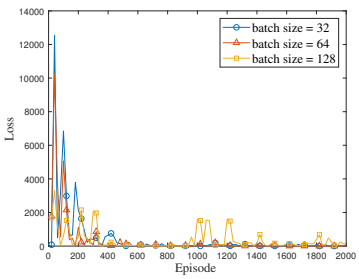
For the above three methods, we compare their network utility, delay, and running time. Among them, the network



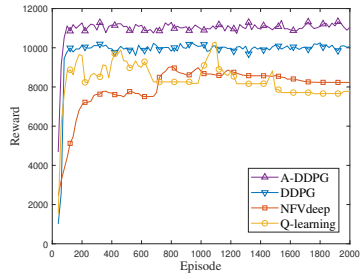
(a) The reward returned by A-DDPG under different learning rates.



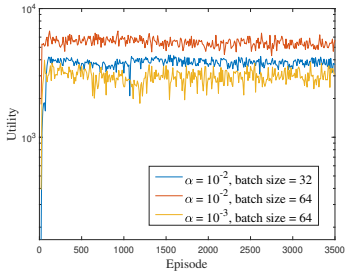
(b) The loss value of the A-DDPG under different learning rates.



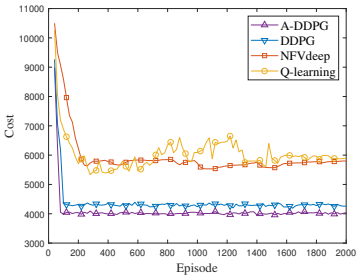
(c) The loss value of A-DDPG with different batch sizes.



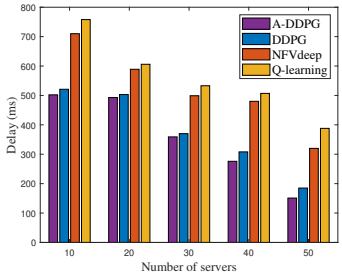
(d) The reward returned by all the algorithms.



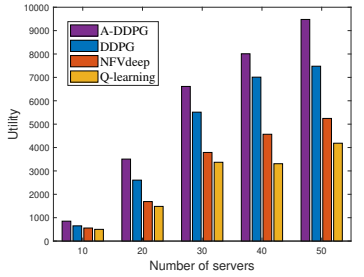
(e) The influence of learning rate and batch size on the utility for A-DDPG.



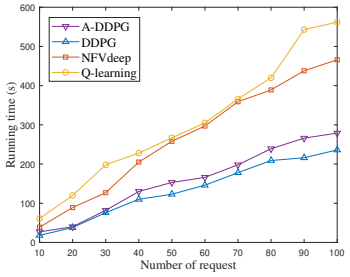
(f) The cost returned by all the algorithms.



(g) The influence of the number of servers on the delay for all the algorithms.



(h) The influence of the number of servers on the utility for all the algorithms.



(i) The influence of the number of servers on the running time for all the algorithms.

Fig. 3. Performance comparison of A-DDPG, DDPG, NFVdeep, and Q-learning.

utility reflects the resource occupancy of the nodes and links of the network according to Eq. (1), and the total delay is calculated by the sum of each path and node processing delay to reflect the network utility of VNF placement and traffic routing.

B. Simulation Results

Fig. 3(a) shows reward returned by A-DDPG under different learning rates (0.1, 0.01 and 0.001). The placement and routing of all VNFs are completed in each training episode. It can be seen from Fig. 3(a) that the learning rate affects the value of reward in the algorithm's training progress. The reason is that the learning rate represents the amount by which the weights are updated (a.k.a. the step size) during training. Smaller learning rates may lead to a slower weight update, so more training episodes are needed to achieve convergence of reward, whereas larger learning rates cause rapid changes and require fewer training episodes. According to our simulation results, when the learning rate is 0.01, A-DDPG achieves the best performance in terms of reward. Therefore, we will take the best learning rate for comparison with other algorithms. Its learning speed is acceptable, and it leads to faster convergence of reward function.

Fig. 3(b) shows the loss value of the A-DDPG method under different learning rates (0.1, 0.01 and 0.001). It can be seen from Fig. 3(b) that the learning rate affects the loss value in the algorithm's training step. The reason is that if the learning rate is too large, the loss function may directly exceed the global optimization of the learning process, whereas a small learning rate can cause the process to get stuck. When the learning rate is small, the changing speed of the loss function is slow. In this context, it will greatly increase the convergence complexity of the network, and the loss function is easy to be trapped in a local minimum. Our simulation shows that the learning rate of 0.01 provides the best performance for A-DDPG.

Fig. 3(c) shows the loss of A-DDPG with different batch sizes, where the batch sizes are 32, 64, and 128, respectively. As can be seen from Fig. 3(c), as the episodes increase, the

batch size will affect the value of the loss. We use batch gradient descent in the simulation to complete the iteration, which processes a portion of the samples at a time. A small portion of the samples will bring a large variance, which will cause the loss function to oscillate and slow down the convergence speed of the algorithm, especially when the network is complex. If the sample size is too large, the gradient estimation will be more accurate and stable, which may cause the neural network to converge to a poor local optimal solution point. Therefore, the batch size cannot be set too small or too large. According to our simulation results, the batch size can be set to 64. In addition, we find that after a series of shocks, the loss value in Fig. 3(c) can always be stable near 0 within a certain range. This indicates that our proposed algorithm can achieve convergence in VNF-PR.

Fig. 3(d) shows the reward returned by all the algorithms. As the episodes increase, the value of the reward gradually converges. In particular, we find that the A-DDPG algorithm is stable after being trained for 200 episodes. Subsequently, the reward value of A-DDPG somewhat fluctuates. On the one hand, this is due to the random generation of network requests, and the reward value is related to the completion of the network request. On the other hand, when poor samples are selected, one may end up in a local optimum, which results in a low reward value. It can be observed from Fig. 3(d) that Q-learning and NFVdeep always return the lowest reward because they do not directly use the deep network to select actions. DDPG performs better due to its capability to select actions directly. However, it is not as good as A-DDPG, since it fails to capture the neighbors' states. The A-DDPG algorithm can always achieve the highest reward after 200 episodes of training among the simulated algorithms. The reason is that the A-DDPG agent takes action by additionally paying attention to the states of neighbors, and the correct behavior enables the agent to obtain positive rewards faster during training, which accelerates the learning process. The results from the training process imply that the A-DDPG agent is more intelligent than other agents.

Fig. 3(e) depicts the influence of learning rate and batch size on the utility for A-DDPG. As the A-DDPG model iterates, the utility gradually converges towards a maximum where the model optimizes the weights. The utility value of $\alpha = 0.01$ is higher than that of $\alpha = 0.001$. We analyze that this is because the higher learning rate may miss the global optimization of the learning process, so it will cause the network to converge to a local optimum and obtain a low utility. Moreover, the speed of convergence when $batch\ size = 64$ is higher than the value when $batch\ size = 32$. This is because, as the batch size increases, the data processing speed becomes faster, which can reduce training time and enhance system stability.

Fig. 3(f) shows the cost returned by all the algorithms. As can be seen from Fig. 3(f), the training efficiency of Q-learning is lower than that of A-DDPG. More specifically, the cost of NFVdeep fluctuates at 5800 after 200 episodes. The cost of Q-learning fluctuates at 6000 after 1800 episodes. The cost of A-DDPG stabilizes after 100 episodes with slight fluctuations at 4000. Given these points, A-DDPG converges faster than NFVdeep and Q-learning. This is because, during initial training, VNFs are randomly placed on the different servers, which incurs large operation and transmission costs. Through the training of Q-table and neural networks, Q-learning and A-DDPG agents can reduce unnecessary costs through training results. However, due to the use of neural networks, the A-DDPG agent is conducive to expressing complex network states. Compared with the discrete strategy in NFVdeep, A-DDPG can directly optimize the strategy (e.g., request rate) to meet the time-varying network states. Under the same conditions, it can process more network requests and reduce cost, thereby improving the processing capacity of the network. The result shows that an A-DDPG agent is more intelligent since it can achieve lower costs.

Fig. 3(g) shows the influence of the number of servers on the delay for all the algorithms. As shown in Fig. 3(g), the delay shows a downward trend as the number of servers increases. With the increase in the number of servers, it guarantees enough resources and applicable paths to accommodate requests. Our A-DDPG algorithm achieves a lower delay compared with the other three approaches. This is because, as the number of servers increases, the random topology becomes more complicated. The placement and routing of VNFs using NFVdeep and Q-learning can easily cause the server to fall into a local bottleneck due to performance degradation, whereas A-DDPG adds incentives for delay optimization in the reward function. The value of the reward increases more and more as delay decreases. Therefore, the node with the smaller delay will be selected to deploy the VNF in the strategic choice.

Fig. 3(h) shows the influence of the number of servers on the utility for all the algorithms. As shown in Fig. 3(h), A-DDPG achieves higher utility than the others. It reflects the superiority of A-DDPG in expressing decision-making in complex network environments. A-DDPG can obtain better utility under time-varying network states compared with the DDPG algorithm. NFVdeep performs well in an environment with 30 servers. Q-learning is not sensitive to the number of servers. In short, the A-DDPG method can attain greater utility by adopting an adaptive selection policy in a complex network environment.

Finally, we compare the running time performance of all the algorithms. As seen in Fig. 3(i), the running time of Q-learning is significantly higher than those of A-DDPG, DDPG, and NFVdeep. In addition, when there are more than 80 requests, the running time of Q-learning increases rapidly. This is because when there are fewer requests, the servers have numbers of capacity available on-demand, and there exist more feasible solutions for the VNF-PR problem to achieve the best performance. However, with an increase in resource demands, the state space and action space in the Q-table greatly increase. In that case, finding the optimal strategy by looking up the table becomes difficult, and considering the complex calculations of nodes and links in large NFV networks, it takes more time to find the optimal strategy in

large networks. Although A-DDPG and DDPG spend some more time to train the neural networks, after the training is completed and deployed, it only needs to use the well-trained neural networks for reasoning. A-DDPG has more running time than DDPG. This is because A-DDPG's neural network architecture has one more attention layer than DDPG, which causes a slight increase in running time. Therefore, A-DDPG consumes reasonable running times due to the powerful representation capabilities of the neural networks.

V. Related Work

Combinatorial optimization theory for NFV: The VNF-PR problem has been studied for different objectives [27], such as cost minimization [28], [29], performance improvement [30], [31], and utility maximization [32]–[34]. In most studies, VNF placement, either alone or jointly with traffic routing, is investigated by using combinatorial optimization theory (e.g., primal-dual, rounding, Markov approximation). For instance, Ma *et al.* [35] target the VNF placement problem to load balance the traffic at base stations. They subsequently solve the problem when the flow path is predetermined, and propose a traffic and space aware routing heuristic for a non-ordered or ordered middlebox set. Feng *et al.* [36] present a VNF placement and traffic routing model to minimize the network cost. The authors formulate the problem as an ILP and then devise an approximate algorithm to effectively consolidate flows into a limited number of active resources. A Minimum-Residue heuristic is presented in [37] for VNF placement in a multi-cloud scenario with constraints of deployment cost. Sampaio *et al.* [38] study how to achieve load balancing in networks to reduce the number of overloaded links in NFV/SDN-enabled networks. However, the aforementioned studies do not consider QoS, such as the delay of the data flow in SFC. Gao *et al.* [28] propose a cost-efficient scheme to address the VNFs placement problem in public cloud networks with the goal of low cost and latency. Cziva *et al.* [39] study how to use the optimal stopping theory to place VNFs under the edge cloud to minimize the total delay expectation. However, the authors in [39] assume that one VNF is sufficient to meet the users' requirements. Nevertheless, applying combinatorial optimization theory cannot work well with the real-time dynamic network variations [26].

DRL for NFV: Some studies solve the VNF-PR problem by using DRL. For instance, Quang *et al.* [40] solve the VNF-PR or VNF forwarding graph embedding problem in multiple non-cooperative domains by jointly considering the delay and underlying infrastructure constraints. They first introduce a DRL framework in which each domain determines the bidding price of using its resources selfishly. After that, the final decision is made by the owner of VNF-PR by executing a Cost-based First Fit (CFF)-based heuristic algorithm. Xiao *et al.* [26] present an adaptive deep reinforcement learning approach to automatically deploy SFCs for the optimization of throughput and operation cost. Tong *et al.* [41] propose a Gated Recurrent Units (GRU)-based traffic prediction model and place VNF instances in advance based on the prediction result. They apply a DRL algorithm called Asynchronous Advantage Actor-Critic (A3C) to train the agent and then obtain the optimal strategy. Manabu *et al.* [42] propose an accelerated reinforcement learning method to shorten the delivery time of services. According to [42], the reinforcement learning agent learns the optimal placement strategy of VNFs according to the state value function and simulates the model in various environments. Sun *et al.* [43] combine the DRL and GNN to solve the VNF placement problem with the minimum deployment cost. However, the methods in [26], [40]–[43] ignore the end-to-end delay, especially the processing delay. Gu *et al.* [44] minimize the deployment cost based on geographic location and the SFC processing delay, and jointly solve the VNF placement and routing problem using a DRL algorithm. However, the authors in [44] suppose that all VNF instances have already been placed on network nodes, so the VNF-PR problem is simplified to the deployment of paths and allocation of traffic load on the links. None of the aforementioned works considers the impact of surrounding nodes' resources on network states. In fact, the importance of neighbors to the learning agent is distinguishable according to their remaining resources in the DRL model. The attention mechanism enables to focus on neighbor nodes with sufficient resources and contributes to the generation of neighbor interaction behaviors. Li *et al.* [45] make a preliminary attempt to solve the VNF placement problem with the combination of attention based sequence model and RL algorithm, where the RL agent incorporates an entropy maximization strategy and the goal is formalized as optimizing the power consumption of the service chain. However, [45] assumes that the problem model is only for a star topology with 10 nodes, which is not always the case in practice. Our proposed A-DDPG solves the VNF-PR problem by applying the attention mechanism to the DRL architecture, using the Actor-Critic network structure.

VI. Conclusion

The main focus of this paper is the VNF-PR problem in NVF-enabled networks, which is to place each requested VNF on network nodes and effectively route traffic through these nodes. To solve the VNF-PR problem, we present a DRL framework with an attention mechanism, called A-DDPG, which consists of three processes: observation process, training process, and online running process. In the observation process, the agent first obtains historical samples through observation for training. Second, the agent updates the network parameters according to the historical samples and reduces the random exploration rate in the training process. Finally, the agent selects the trained network model and then executes the action of the maximum reward to get the optimized VNF-PR policy. By introducing an attention mechanism, we can focus on more critical information, such as the state of neighbors, to reduce the attention to other unnecessary nodes and improve the training efficiency of the model. Via extensive simulations, we find that our proposed algorithm A-DDPG can outperform the state-of-the-art in terms of network utility, delay, and cost.

REFERENCES

[1] M. S. Bonfim, K. L. Dias, and S. F. Fernandes, "Integrated NFV/SDN architectures: A systematic literature review," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–39, 2019.

[2] H. Ren, Z. Xu, W. Liang, Q. Xia, P. Zhou, O. F. Rana, A. Galis, and G. Wu, "Efficient algorithms for delay-aware NFV-enabled multicasting in mobile edge clouds with resource sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2050–2066, 2020.

[3] M. Scazzariello, L. Ariemma, G. Di Battista, and M. Patrignani, "Megalos: A scalable architecture for the virtualization of network scenarios," in *IEEE NOMS*, 2020, pp. 1–7.

[4] L. Ruiz, R. Duran, I. de Miguel, N. Merayo, J. Aguado, P. Fernandez, R. Lorenzo, and E. Abril, "Comparison of different protection schemes in the design of VNF-mapping with VNF resiliency," in *IEEE ICTON*, 2020, pp. 1–4.

[5] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295–314, 2021.

[6] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability driven VNF placement in a MEC-NFV environment," in *IEEE GLOBECOM*, 2018, pp. 1–7.

[7] D. T. Nguyen, C. Pham, K. K. Nguyen, and M. Cheriet, "Placement and chaining for run-time IoT service deployment in edge-cloud," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 459–472, 2019.

[8] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *IEEE ICDCS*, 2017, pp. 731–741.

[9] Z. Allybokus, N. Perrot, J. Leguay, L. Maggi, and E. Gourdin, "Virtual function placement for service chaining with partial orders and anti-affinity rules," *Networks*, vol. 71, no. 2, pp. 97–106, 2018.

[10] 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.

[11] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," in *IEEE INFOCOM*, 2019, pp. 523–531.

[12] M. Golkarifard, C. F. Chiasserini, F. Malandrino, and A. Movaghar, "Dynamic VNF placement, resource allocation and traffic routing in 5G," *Computer Networks*, vol. 188, p. 107830, 2021.

[13] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "Online and batch algorithms for VNFs placement and chaining," *Computer Networks*, vol. 158, pp. 98–113, 2019.

[14] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI Conference on Artificial Intelligence*, 2018.

[15] S. Guo, Y. Dai, S. Xu, X. Qiu, and F. Qi, "Trusted cloud-edge network resource management: DRL-driven service function chain orchestration for IoT," *IEEE IoT*, 2019.

[16] H. R. Khezri, P. A. Moghadam, M. K. Farshbafan, V. Shah-Mansouri, H. Kebriaei, and D. Niyato, "Deep reinforcement learning for dynamic reliability aware NFV-based service provisioning," in *IEEE GLOBECOM*, 2019, pp. 1–6.

[17] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "On using deep reinforcement learning for VNF forwarding graphs placement," in *IEEE NoF*, 2020, pp. 126–128.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[19] R. Z. Courville, Ruslan Salakhudinov and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *International Conference on Machine Learning*, 2015.

[20] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE*, vol. 5, no. 1, pp. 3–55, 2001.

[21] J. S. Chase, R. P. Doyle, and S. D. Ims, "Multi-tier service level agreement method and system," 2006.

[22] S. Verbrugge, D. Colle, M. Pickavet, P. Demeester, S. Pasqualini, A. Iselt, A. Kirstadter, R. Hu lsermann, F.-J. Westphal, and M. Jager, "Methodology and input availability parameters for calculating opex and capex costs for realistic network scenarios," *Journal of Optical Networking*, vol. 5, no. 6, pp. 509–520, 2006.

[23] T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, and H. Wang, "On efficient offloading control in cloud radio access network with mobile edge computing," in *IEEE ICDCS*, 2017, pp. 2258–2263.

[24] K. Gardner, S. Borst, and M. Harchol-Balter, "Optimal scheduling for jobs with progressive deadlines," in *IEEE INFOCOM*, 2015, pp. 1113–1121.

[25] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing costs on service chain placement in network functions virtualization," in *IEEE NFV-SDN*, 2015, pp. 191–197.

[26] 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 *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.

[27] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet, "VNF-AAPC: Accelerator-aware vnf placement and chaining," *Elsevier Computer Networks*, p. 107329, 2020.

[28] T. Gao, X. Li, Y. Wu, W. Zou, S. Huang, M. Tornatore, and B. Mukherjee, "Cost-efficient VNF placement and scheduling in public cloud networks," *IEEE Transactions on Communications*, 2020.

[29] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-end performance-based autonomous VNF placement with adopted reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, 2020.

[30] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks," *IEEE J-SAC*, vol. 38, no. 2, pp. 263–278, 2019.

[31] L. Dinh-Xuan, C. Popp, V. Burger, F. Wamser, and T. Hoßfeld, "Impact of VNF placements on qoe monitoring in the cloud," *International Journal of Network Management*, vol. 30, no. 3, p. e2053, 2020.

[32] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture," in *IEEE INFOCOM*, 2017, pp. 1–9.

[33] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE J-SAC*, vol. 38, no. 2, pp. 279–291, 2019.

[34] H. A. Shah and L. Zhao, "Multi-agent deep reinforcement learning based virtual resource allocation through network function virtualization in internet of things," *IEEE IoT*, 2020.

[35] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *IEEE INFOCOM*, 2017, pp. 1–9.

[36] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," in *IEEE INFOCOM*, 2017, pp. 1–9.

[37] D. Bhamare, R. Jain, M. Samaka, G. Vaszkun, and A. Erbad, "Multi-cloud distribution of virtual functions and dynamic service deployment: Open ADN perspective," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 299–304.

[38] L. S. Sampaio, P. H. Faustini, A. S. Silva, L. Z. Granville, and A. Schaeffer-Filho, "Using NFV and reinforcement learning for anomalies detection and mitigation in SDN," in *IEEE ISCC*, 2018, pp. 00 432–00 437.

[39] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal VNF placement at the network edge," in *IEEE INFOCOM*, 2018, pp. 693–701.

[40] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, "Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach," in *IEEE INFOCOM WKSHPS*, 2019, pp. 886–891.

[41] R. Tong, S. Xu, B. Hu, J. Zhao, L. Jin, S. Guo, and W. Li, "VNF dynamic scaling and deployment algorithm based on traffic prediction," in *IEEE IWCMC*, 2020, pp. 789–794.

[42] M. Nakanoya, Y. Sato, and H. Shimonishi, "Environment-adaptive sizing and placement of nfv service chains with accelerated reinforcement learning," in *IEEE IM*, 2019, pp. 36–44.

[43] P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, "Combining deep reinforcement learning with graph neural networks for optimal vnf placement," *IEEE Communications Letters*, 2020.

[44] L. Gu, D. Zeng, W. Li, S. Guo, A. Zomaya, and H. Jin, "Deep reinforcement learning based vnf management in Geo-distributed edge computing," in *IEEE ICDCS*, 2019, pp. 934–943.

[45] S. Li, S. Zhang, L. Chen, H. Chen, X. Liu, and S. Lin, "An attention based deep reinforcement learning method for virtual network function placement," in *IEEE ICCC*, 2020, pp. 1005–1009.