Deep-NFVOrch: Leveraging Deep Reinforcement Learning to Achieve Adaptive vNF Service Chaining in DCI-EONs

Baojia Li, Wei Lu, Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—Due to the raising of cloud computing, how to realize adaptive and cost-effective virtual network function service chaining (vNF-SC) in a datacenter interconnection based on elastic optical network (DCI-EON) has become an interesting but challenging problem. In this work, we tackle this problem by optimizing the design of a deep reinforcement learning (DRL) based adaptive service framework, namely, Deep-NFVOrch. Specifically, Deep-NFVOrch works in service cycles, and tries to reduce the setup latency of vNF-SC by invoking request prediction and pre-deployment at the beginning of each service cycle. We introduce a DRL-based observer (DRL-Observer) to select the duration of each service cycle adaptively according to the network status. The DRL-Observer is designed based on the advantage actor critic (A2C), which can interact with the network environment constantly through its deep neural network (DNN) and learn how to make wise decisions based on the environment's feedback. Our simulation results demonstrate that DRL-Observer converges fast in online training with the help of a few asynchronous training threads, and the Deep-NFVOrch with it achieves better performance than several benchmarks, in terms of balancing the tradeoff among the overall resource utilization, the vNF-SC request blocking probability, and the number of network reconfigurations in a DCI-EON.**

***Index Terms*—Network function virtualization (NFV), Service chaining, Datacenter interconnection (DCI), Elastic optical networks (EONs), Deep reinforcement learning (DRL), Artificial intelligence (AI).**

I. Introduction

With the emerging of new network paradigms such as 5G and Internet-of-things (IoT), the demands for real-time, adaptive and inexpensive network services are increasing fast [1, 2]. Therefore, if service providers (SPs) still deploy dedicated middleboxes to cope with the demands, both the capital expenditures (CAPEX) and operational expenses (OPEX) would be skyrocketing. This dilemma pushes people to consider network function virtualization (NFV) [3] as the alternative, which replaces dedicated middleboxes with the virtual network functions (vNFs) realized on commodity servers [4]. Hence, an SP can instantiate vNFs in datacenters (DCs) dynamically and adaptively in response to time-variant demands for network services, significantly reducing both CAPEX and OPEX. Moreover, the service provisioning enabled by NFV can be even more flexible, if the SP defines atomic vNFs and composes relatively complex network services with a series of such vNFs. This is usually referred to as vNF service chaining (vNF-SC) [5, 6], *i.e.*, building a network service by steering the application traffic through a series of vNFs in sequence.

In addition to the innovations on the IT side, NFV also gains momentum from new optical networking architectures (*e.g.*, flexible-grid elastic optical networks (EONs) [7–11]). This is because vNF-SC can route high-throughput and bursty traffic across several DCs and a DC interconnection (DCI) has to use an optical network as its physical layer [12, 13]. With a spectrum allocation granularity at 12.5 GHz or even smaller, EONs remove the restriction due to the fixed-grids in conventional wavelength-division multiplexing (WDM) networks. Therefore, a DCI based on EON (DCI-EON) becomes promising for supporting NFV [14–16].

In order to achieve vNF-SC in a DCI-EON, an SP needs to deploy the required vNFs in proper DCs, establish lightpaths in the EON to interconnect the vNFs, and steer the application traffic over the lightpaths in sequence to go through the vNFs one by one. Previous studies have designed various algorithms to tackle the problem of provisioning vNF-SC in DCI-EONs [6, 16–18], including both integer linear programming (ILP) models and time-efficient heuristics. However, these approaches are still reactive, which means that the deployment of vNFs and lightpaths is only conducted when the SP actually sees a vNF-SC request. Hence, they can hardly get around the excessive latency from setting up lightpaths and instantiating vNFs, and this would hinder real-time and on-demand vNF-SC provisioning. Note that, the setup latencies of lightpaths and vNFs are usually several seconds [19] and tens of seconds [20], respectively, and they will make the total latency of provisioning a vNF-SC request in the order of minutes. Nevertheless, in today's Internet, there are numerous network services that have stringent quality-of-service (QoS) requirement on setup delay. For instance, it is known that video viewers start to close their browsers if the loading of a video content takes more than 2 seconds, and the give-up ratio increases 5.8% for each additional second spent on loading [21].

To address this latency issue, we proposed a new service framework that first predicts future vNF-SC requests based on historical information, then deploys the required vNFs and lightpaths in the DCI-EON in advance, and thus only needs to steer the application traffic through the vNFs in sequence when the requests actual come in [22]. The service framework with pre-deployment is shown in Fig. 1 (adapted from [23]), which operates based on service cycles and the duration of the n-th cycle is ΔT_n. Each service cycle includes a pre-

B. Li, W. Lu, and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

Manuscript received on June 30, 2019.

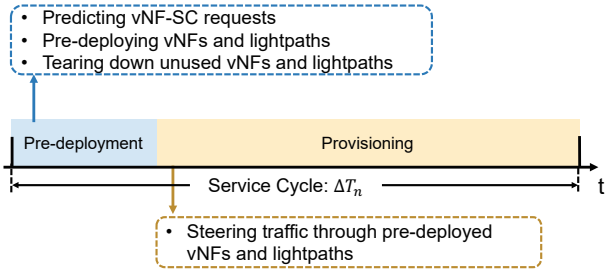

Fig. 1. Service framework for vNF-SC provisioning with pre-deployment.

deployment phase followed by a provisioning phase. In the pre-deployment phase[1], we use a deep learning (DL) module to predict the requests that might arrive in this service cycle, deploy the required vNFs and lightpaths in advance based on the prediction, and tear down the unused vNFs and lightpaths in the DCI-EON. Next, we proceed to the provisioning phase, and steer application traffic through the required vNFs to form a vNF-SC upon receiving each new vNF-SC request. To this end, the total latency of provisioning a vNF-SC request is shortened significantly, since the setup latencies of lightpaths and vNFs are removed from it and the traffic steering can usually be finished within hundreds of milliseconds in a software-defined networking (SDN) based environment [24].

Although the service framework proposed in [22] is promising in terms of real-time and on-demand vNF-SC provisioning, it still suffers from the inflexibility due to using a fixed service cycle duration (*i.e.*, the value of ΔT_n is predetermined and fixed throughout the service provisioning). The value of ΔT_n actually adjusts the tradeoff between the operation complexity and the resource utilization of vNF-SC provisioning. Decreasing ΔT_n can improve the accuracy of the prediction on new vNF-SC requests and reduce the idling time of pre-deployed vNFs and lightpaths, which will help to enhance resource utilization in the DCI-EON. However, a shorter ΔT_n also means that the network reconfigurations (*i.e.*, setting up and tearing down vNFs and lightpaths) would be more frequent to push up the operation complexity [25, 26]. On the other hand, even though increasing ΔT_n can avoid unnecessary network reconfigurations, it would degrade the framework's performance on resource utilization. Therefore, the selection of ΔT_n's value should be careful and adaptive to properly balance the tradeoff in a time-varying network environment. This, however, can hardly be achieved with a classic optimization algorithm due to the complexity of vNF-SC provisioning in DCI-EONs [22].

Meanwhile, machine learning (ML) has recently attracted intensive interests, because the approaches based on it can potentially make intelligent decisions to handle complicated environments. Hence, people started to apply ML to solve complex optimizations in optical networks [27, 28]. Reinforcement learning is a type of ML that enables online training of neural networks, and thus it provides superior adaptivity to address sophisticated optimizations in dynamic network environments [29–32]. Therefore, to resolve the aforementioned issue with our service framework proposed in [22], we leveraged deep reinforcement learning (DRL) to design an adaptive service framework, *i.e.*, Deep-NFVOrch [23]. More specifically, to properly select the value of ΔT_n for each service cycle, we introduced a DRL based observer, namely, DRL-Observer. At the beginning of each service cycle, the DRL-Observer collects instant performance metrics regarding the service provisioning in the previous cycle, and feeds them into its deep neural network (DNN) to determine the duration of the current cycle and update the DL-based vNF-SC request predictor and its own DNN. Hence, the service framework can always be adapted to the current network status. The DRL-Observer is based on the advantage actor critic (A2C) [33], which can interact with the network environment constantly through its DNN and learn how to make wise decisions based on the environment's feedback (*i.e.*, the reward calculated with the instant performance metrics).

In this work, we expand our preliminary study in [23] to make it much more comprehensive, and the major improvements are summarized as follows:

- We describe the design of the DRL-Observer in more detail, and further optimize the asynchronous training scheme to expedite its convergence in online training.
- We optimize the operation to determine ΔT_n in the DRL-Observer. To verify the DRL-Observer's benefits, we design another adaptive framework without it and run extensive simulations to compare the two frameworks.
- We perform extensive simulations to check how the key parameters of the DRL-Observer affect its performance, and summarize the empirical way to select the key parameters' values based on the simulation results.

The rest of paper is organized as follows. Section II presents the design of Deep-NFVOrch. The detailed design and operation principle of the DRL-Observer are introduced in Section III. We conduct extensive simulations to evaluate the performance of Deep-NFVOrch in Section IV. Finally, Section V summaries the paper.

II. Proposed Adaptive Service Framework

A. Architecture of Deep-NFVOrch

Fig. 2 (adapted from [23]) shows the architecture and operation principle of our adaptive service framework for provisioning vNF-SC in an DCI-EON (*i.e.*, Deep-NFVOrch). It can be seen that Deep-NFVOrch is a periodic system whose service cycle is controlled by the system timer. The system timer also drives the operation of Deep-NFVOrch in each cycle, to divide it as a pre-deployment phase followed by a provisioning phase. Meanwhile, the duration of each cycle (*i.e.*, ΔT_n) is determined by the DRL-Observer to ensure adaptive service provisioning. At the beginning of a pre-deployment phase, the system timer wakes up the DL-based request predictor, which then forecasts the future vNF-SC requests that will come in this service cycle (*i.e.*, the next ΔT_n) based on historical information. Next, the vNF-SC pre-deployment module tries to establish lightpaths and instantiate vNFs based on the prediction, which is done by leveraging a

[1]We make the duration of the pre-deployment phase much shorter than ΔT_n, and thus we can assume that all the new vNF-SC requests in the service cycle arrive in its provisioning phase.

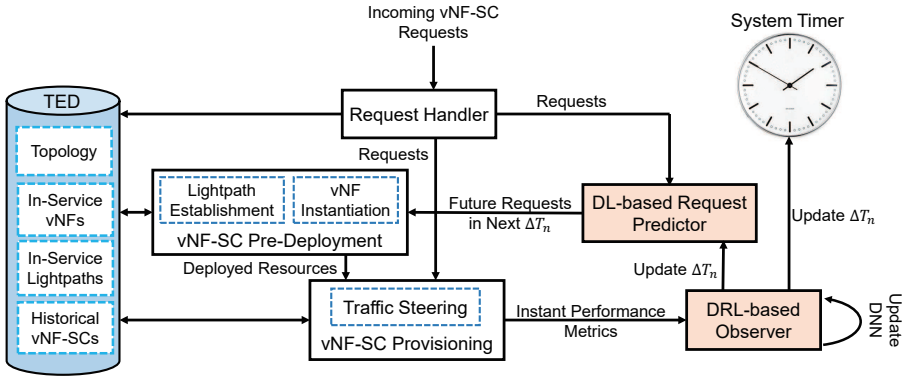

Fig. 2. Architecture and operation principle of Deep-NFVOrch, TED: traffic engineering database.

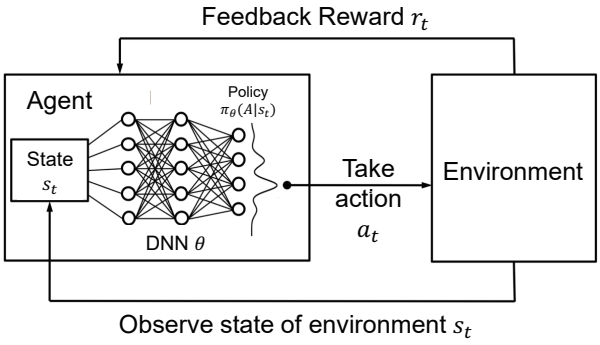

Fig. 3. Operation principle of DRL-Observer.

time-efficient heuristic algorithm to orchestrate the spectrum and IT resources in the DCI-EON for vNF-SC (*e.g.*, the LBA algorithm in [6]). More specifically, for each vNF-SC request, the LBA algorithm first reuses as many deployed vNFs as possible by finding the longest common subsequence between requested and deployed vNFs, then deploys the remaining vNFs with the spectrum-saving principle, and finally sets up lightpaths to connect the deployed vNFs with the routing and spectrum assignment (RSA) schemes that cause the smallest increase on the maximum frequency slot index (MFSI) [8]. In the meantime, the unused vNFs and lightpaths, which were set up for expired vNF-SC requests, are torn down to maintain high resource utilization.

Then, the system timer moves Deep-NFVOrch to the provisioning phase, and instructs the request handler to start to accept vNF-SC requests. The incoming requests are dispatched to the DL-based request predictor and the vNF-provisioning module. The predictor stores the requests as historical ones and prepares itself with them for the next prediction, while the vNF-SC provisioning module steers application traffic through the required vNFs in each vNF-SC request to accomplish the service provisioning quickly. The traffic engineering database (TED) is introduced to record the DCI-EON's status, which includes the network topology, in-service vNFs on DCs, in-service lightpaths in the EON, and historical vNF-SC requests.

As the heart of Deep-NFVOrch, DRL-Observer monitors the system's performance proactively and updates its key components/parameters accordingly. Specifically, at the beginning of each service cycle, DRL-Observer collects the instant performance metrics regarding the service provisioning in the previous cycle (*i.e.*, the blocking probability, the average utilization of pre-deployed vNFs and lightpaths, and the number of network reconfigurations), which can be used to evaluate previous cycle and update its DNN to get better performance. Meanwhile it leverages its DNN to determine the proper system configuration based on current environment, and then updates Deep-NFVOrch accordingly. For example, it determines the duration of the current service cycle (*i.e.*, ΔT_n) and feeds ΔT_n to the system timer and DL-based request predictor, and it also updates its DNN.

B. DRL based on Advance Actor Critic (A2C)

In principle, DRL is about an intelligent agent interacting with a time-varying environment and learning how to act in different states to maximize its reward with a DNN [34]. Hence, the design of our DRL-Observer also follows this principle, as shown in Fig. 3. The operation of DRL-Observer involves four elements, *i.e.*, the agent, action, reward, and state. The agent is our DRL-Observer, and it constantly interacts with the environment (*i.e.*, the DCI-EON) to receive a state s_t at time t. Then, the agent leverages its DNN to generate an action policy $\pi_\theta(A|s_t)$, which denotes the distribution of the probabilities to take possible actions at state s_t. Here, A represents the set of all the possible actions that can be taken by the agent, and θ denotes the DNN's parameters.

Based on the action policy $\pi_\theta(A|s_t)$, the agent chooses an action a_t to take in response to the state s_t with the stochastic method. This means that the actions are selected randomly according to their probability distribution in $\pi_\theta(A|s_t)$. By doing so, we can encourage the agent to explore the action space thoroughly and prevent it from being trapped by local extremes. After having taken action a_t, the agent gets an instant reward r_t from the environment, which can be used to evaluate the performance of a_t, and in the meantime, the environment's state changes to s_{t+1}. Then, the procedure gets repeated as time goes on, and for each time t, the agent stores the tuple of $< s_t, a_t, r_t >$ as an entry of historical experience. The experiences are used to train the agent's DNN and guide it to act better. Here, the "better" means that the agent can obtain larger rewards by taking appropriate actions in different states.

Note that, the action a_t not only determines the next state s_{t+1} but also affects the subsequent states (*i.e.*, $\{s_{t+k}, k \in \mathbf{N}^+\}$). Therefore, we calculate the long-term "return" of action a_t as

$$R_t(s_t, a_t) = \sum_{k=0}^{\infty} \xi^k \cdot r_{t+k}, \tag{1}$$

where $\xi \in (0, 1]$ is the discounted factor. The training of the DNN in DRL-Observer needs to update its parameters θ such that return of each action can be maximized.

The advantage actor critic (A2C) is a stable model for DRL. Specifically, the A2C model involves two neural networks, *i.e.*, an actor neural network (A-NN) θ_a and a critic neural network (C-NN) θ_c. The A-NN chooses an action in a state, while the C-NN evaluate the performance of the selected action. At each time t, the A-NN takes the state s_t as its input to generate an action policy $\pi_{\theta_a}(A|s_t)$, and the C-NN obtains the "value" of state s_t as $V_{\theta_c}(s_t)$. Then, the return of action a_t can be decomposed into two parts, *i.e.*, the value of state s_t ($V_{\theta_c}(s_t)$) and the advantage of action a_t in state s_t ($\Omega(s_t, a_t)$), as

$$R_t(s_t, a_t) = V_{\theta_c}(s_t) + \Omega(s_t, a_t). \tag{2}$$

Here, $\Omega(s_t, a_t)$ not only tells the agent how good action a_t is in state s_t, but also indicates how much better the action turned out to be than expected.

The training of the A2C-based DRL-Observer uses sufficient entries of historical experiences, which cover the time period $[\tau, \tau + M]$. Then, the return of action a_t in Eq. (3) is approximated with

$$R_t(s_t, a_t) = \sum_{k=0}^{\tau+M-t} \xi^k \cdot r_{t+k}, \quad \forall t \in [\tau, \tau + M]. \tag{3}$$

In the training process, the objective of the C-NN is to learn how to evaluate $V_{\theta_c}(s_t)$ more accurately, and thus we define its loss function as

$$L_c = \left[R_t(s_t, a_t) - V_{\theta_c}(s_t)\right]^2. \tag{4}$$

Meanwhile, the objective of the A-NN is to learn how to act in different states to get higher long-term return. In other words, the A-NN's training process should try to make the probability of choosing action a_t higher, if the advantage of action a_t in state s_t is larger. Hence, we define its loss function as

$$L_a = -\log\left(\pi(a_t|s_t)\right) \cdot \Omega(s_t, a_t), \tag{5}$$

where $\Omega(s_t, a_t)$ is the advantage of action a_t in state s_t and $\pi(a_t|s_t)$ is the probability to take action a_t in state s_t. With the loss functions, the training process calculates their gradients and updates the parameters of A-NN and C-NN accordingly.

C. Network Model

We model the IDC-EON as a graph $G(V, E)$, where V and E denote the sets of DCs and fiber links, respectively. Each DC $v \in V$ contains C_v IT resources for instantiating vNFs, while each fiber link $e \in E$ contains F frequent slots (FS'). A lightpath can be established between any two DCs, as long as its RSA scheme complies with the spectrum contiguous and continuous constraints [35] and there are enough spectra in the related fiber links. We assume that there are Z types of vNFs available in the network, and each z-th type of vNF (*i.e.*, denoted as vNF-z) consumes c_z IT resources. A vNF-SC request demands for a series of vNFs, which can be model as $SC = \{f_1, f_2, \cdots, f_K\}$. Here, K is the total number of vNFs in the request, and f_k is the k-th vNF. Then, a vNF-SC request can be modeled as $R^i = \{s^i, d^i, SC^i, b^i, t_a^i, t_l^i\}$, where i is its index, s^i and d^i are the source and destination DCs, respectively, SC^i is the required vNF-SC, b^i is the bandwidth requirement, and t_a^i and t_l^i are the arrival time and hold-on time, respectively.

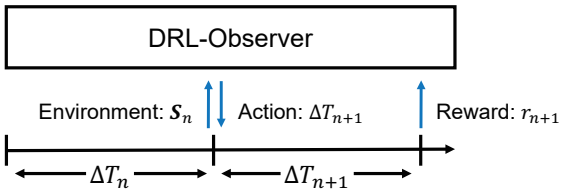

Fig. 4. Interaction between DRL-Observer and DCI-EON.

Based on the network model above, DRL-Observer monitors the performance of Deep-NFVOrch proactively and updates its key components/parameters accordingly. Here, the most important parameter is the duration of each service cycle, *i.e.*, ΔT_n. We denote the end-time of the n-th cycle as t_n. The four DRL-related elements are explained as follows.

- **State**: For the n-th service cycle $[t_{n-1}, t_n]$, we define $\phi_{v,n}$ as the ratio of used to total IT resources on DC $v \in V$, $\varphi_{e,n}$ as the ratio of used to total FS' on fiber link $e \in E$, and $\psi_{\tau,n}$ as the number of requests arrived within $[t_n - \tau, t_n]$. The state of the DCI-EON at t_n (*i.e.*, $\mathbf{S}_n$) is represented with the following three sets: $\{\phi_{v,n},\ \forall v \in V\}$, $\{\varphi_{e,n},\ \forall e \in E\}$, and $\{\psi_{\tau,n},\ \forall \tau \in [1, t_n - t_{n-1}]\}$.
- **Agent**: The agent is just DRL-Observer, and it interacts with its environment (*i.e.*, the DCI-EON) as shown in Fig. 4. Specifically, DRL-Observer gets the state $\mathbf{S}_n$ when the n-th service cycle ends, and determines the value of ΔT_{n+1} based on it. Then, when the $(n+1)$-th cycle ends, DRL-Observer receives the reward r_{n+1}.
- **Reward**: We define the reward r_n as [23]

$$r_n = \alpha \cdot \bar{\phi} - \beta \cdot \log[\max(p_b, 10^{-6})] + \gamma \cdot \frac{1}{\eta}. \tag{6}$$

  Here, $\bar{\phi}$ is the average utilization of pre-deployed lightpaths and vNFs, p_b is the blocking probability, and η is the number of network reconfigurations (*i.e.*, the total number of newly-established lightpaths and vNFs), in the n-th service cycle. The number of network reconfigurations is considered because it directly determines the complexity of network control and management (NC&M) [36, 37]. Eq. (6) is formulated in the way that the three concerned metrics are normalized, *i.e.*, not only making their values be in the same magnitude but also ensuring that they will change in a similar scale. The weight coefficients α, β and γ are used to adjust the importance of the three metrics. Since the network operator might have its own preference when balancing the performance metrics, it determines the values of α, β and γ empirically, which

will be discussed in detail in Section IV.

- **Action**: The action of DRL-Observer is to determine the value of ΔT_n. We assume that ΔT_n can only take discrete values in $[\Delta T^{min}, \Delta T^{max}]$, *i.e.*, the size of A is finite.

III. Design of DRL-Observer

A. Operation Principle

We design DRL-Observer based on A2C, and merge the A-NN and C-NN as an actor-critic neural network (AC-NN). This means that we make the two neural networks share the first two layers. By doing so, we can reduce the number of parameters to adjust for the AC-NN and accelerate its training process. Meanwhile, since this would also let the training of A-NN and C-NN affect each other, we should set their parameter adjustment schemes carefully to avoid parameter oscillation during the training. Then, DRL-Observer consists of an AC-NN, an experience store, and a decision module. The AC-NN is trained to take the current state $\mathbf{S}_n$ as the input, and it outputs the action policy $\pi(\Delta T_{n+1}|\mathbf{S}_n)$, and the value of the state V_n. The decision module obtains the value of ΔT_{n+1} based on $\pi(\Delta T_{n+1}|\mathbf{S}_n)$, and updates the system timer and DL-based request predictor accordingly. At the end of the service cycle, DRL-Observer gets a reward r_{n+1} and transforms the environment state to $\mathbf{S}_{n+1}$ as explained in Section II-C. Then, it records the tuple of $< \mathbf{S}_n, \Delta T_{n+1}, r_{n+1}, \mathbf{S}_{n+1}, V_n >$ as an entry of experience in the experience store, which can be used as a training sample to update the AC-NN, before the whole system moving to the next service cycle.

To ensure stable operation, we do not just update the AC-NN when each service cycle ends. The training process will only be triggered when the system has run for M cycles and recorded M training samples in the experience store. Then, the online training operates as follows. Firstly, we calculate the long-term return of the action in each training sample as

$$R_m = \sum_{k=0}^{M+n-m} \xi^k \cdot r_{m+k}, \quad \forall m \in [n+1, n+M], \tag{7}$$

where we assume that the M training samples cover $(n+1)$-th to $(n+M)$-th service cycles. Hence, the loss function of the C-NN can be get as

$$L_c(\theta_c) = \sum_{m=1}^{M} (R_{n+m} - V_{n+m})^2. \tag{8}$$

Meanwhile, with the state value V_m provided by the C-NN, we can get the advantage of action a_m in state $\mathbf{S}_m$ (*i.e.*, $\Omega_m(\mathbf{S}_m, a_m)$) with Eq. (2). Here, action a_m is essentially to determine the duration of the m-th service cycle as ΔT_m. The loss function of the A-NN becomes

$$L_a(\theta_a) = -\sum_{m=1}^{M} \Omega(\mathbf{S}_{n+m}, a_{n+m}) \cdot \log[\pi_{\theta_a}(\mathbf{S}_{n+m}) \cdot \zeta(a_{n+m})] - \epsilon \sum_{m=1}^{M} \sum_{a \in A} \pi_{\theta_a}(\mathbf{S}_{n+m}) \cdot \log[\pi_{\theta_a}(\mathbf{S}_{n+m})], \tag{9}$$

where $\zeta(a_{n+m})$ is the one-hot encoding [38] of action a_{n+m}. Note that, compared with the loss function in Eq. (5), we add the second term in Eq. (9), which is the policy entropy to encourage exploring the solution space more thoroughly, and ϵ is the hyper-parameter to denote the extending of exploration.

Algorithm 1 provides the details regarding the online training of DRL-Observer. *Lines* 1-2 are for the initialization. The actual training process is explained with *Lines* 3-18. In the while-loop covering *Lines* 5-13 DRL-Observer interacts with the DCI-EON constantly to accumulate training samples in its experience store. When it has recorded M training samples, DRL-Observer leverages the operations in *Lines* 14-16 to update the AC-NN's parameters with the adam optimization algorithm [39] that utilizes stochastic gradient descent (SGD). After each parameter update is completed, experience store is emptied for the next update in *Line* 17.

Algorithm 1: Online Training of DRL-Observer

```
1  initialize parameters of the AC-NN (θ_a and θ_c) randomly;
2  set the initial service cycle as 1;
3  repeat
4      n = 0;
5      while n < M do
6          get current state S_n;
7          use the AC-NN to generate action policy π_{θ_a}(S_n)
           and value V_{θ_c}(S_n);
8          use the decision module to determine ΔT_{n+1}
           based on π_{θ_a}(S_n);
9          run the DCI-EON for ΔT_{n+1} to predict,
           pre-deploy, and provision vNF-SC requests;
10         calculate reward r_n with Eq. (6);
11         record < S_n, ΔT_{n+1}, r_{n+1}, S_{n+1}, V_n > as a
           training sample in the experience store;
12         n = n + 1;
13     end
14     get the advantage of M training samples with Eq. (2);
15     calculate the gradients of the loss functions (∇_{θ_a}(L_a)
       and ∇_{θ_c}(L_c)) based on Eqs. (8) and (9);
16     use the adam optimization algorithm [39] to update
       θ_a and θ_c with the gradients;
17     empty the experience store;
18 until convergence;
```

B. Design of Neural Networks

To realize accurate mapping from environment state to action (*i.e.*, $\mathbf{S}_n \rightarrow \Delta T_{n+1}$), we build the AC-NN with a 3-layer structure, as shown in Fig. 5. Here, the A-NN and C-NN in the AC-NN share the first and second layers to extract information from environment state, while the third layer reshapes the processed information to generate the action policy and state value. After obtaining the outputs, the A-NN and C-NN use the loss functions in Eqs. (9) and (8), respectively, to evaluate them. Since environment state $\mathbf{S}_n$ contains three sets, *i.e.*, $\{\phi_{v,n},\ \forall v \in V\}$, $\{\varphi_{e,n},\ \forall e \in E\}$, and $\{\psi_{\tau,n},\ \forall \tau \in [1, t_n - t_{n-1}]\}$, the elements in $\mathbf{S}_n$ can take different variable types (*e.g.*, real and integer) and ranges.

Therefore, we use different types of neural networks in the first layer to process the elements. The IT resource and

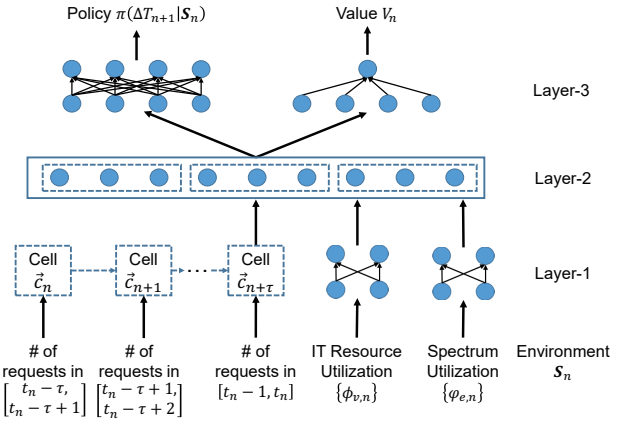

Fig. 5. Structure of AC-NN.

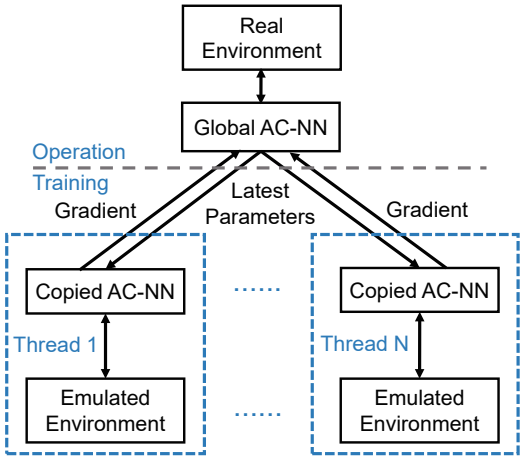

Fig. 6. Asynchronous training scheme.

spectrum utilizations ($\{\phi_{v,n},\ \forall v \in V\}$ and $\{\varphi_{e,n},\ \forall e \in E\}$) can be represented with one-dimensional vectors, and the value of their elements is within $[0, 1]$. Hence, we use a 2-layer fully connected neural network (FC-NN) with activation function based on rectified linear unit (ReLU) to process the vectors, where the ReLU activation function has the expression of

$$g(x) = \max(x, 0). \tag{10}$$

To extract the temporal information from historical vNF-SC requests, we use the long short term memory (LSTM) to process $\{\psi_{\tau,n},\ \forall \tau \in [1, t_n - t_{n-1}]\}$. Specifically, as shown in the Fig.5, LSTM mainly consists of an memory cell, which takes the number of vNF-SC requests over one time unit (*e.g.*, $[t_n - \tau, t_n - \tau + 1]$) as input in each time step and updates its status vector $\vec{c}_n$ accordingly.

The second layer of the AC-NN is the concatenation layer, which concatenates the outputs of the first layer as a vector. The third layer is the output layer and it reshapes the vector generated by the second layer to obtain the action policy (A-NN) and state value (C-NN). Since the actions (*i.e.*, possible values for ΔT_n) are discrete and limited and the A-NN needs to generate the action policy whose elements are all within $[0, 1]$, we use a 2-layer FC-NN as its output layer, with the sigmoid activation function

$$g(\vec{x}, i) = \frac{e^{x_i}}{\sum_{k=1}^{|A|} e^{x_k}}, \tag{11}$$

and the number of neurons in the last layer of the FC-NN is set as the size of the action space, *i.e.*, $\Delta T^{max} - \Delta T^{min} + 1$. The state value from the C-NN is a scalar within $(0, +\infty)$, and thus we use a 2-layer FC-NN with ReLU activation function as its output layer and the number of neurons in the last layer of the FC-NN is set as 1.

Note that, the three layers in Fig. 5 are defined according to their functionalities, while the actual structure of the AC-NN is much more complicated than a simple three-layer neural network. This is because its input contains multiple types of data (*i.e.*, time series of requests, and IT and spectrum resource vectors), and each of its three layers includes multiple neural networks or sub-layers for feature extraction. Therefore, the depth and number of parameters of the AC-NN make it qualified for a DNN [40], which is also the reason why we refer to the reinforcement learning that uses it as DRL.

C. Asynchronous Training

To improve the efficiency of the online training, we design an asynchronous training scheme that uses multiple threads to make copies of the AC-NN in DRL-Observer and let them interact with emulated environments simultaneously. Specifically, there is a global agent to determine ΔT_n for the real DCI-EON, while the remaining agents get trained in the background and send the gradients back to the global agent, as shown in Fig. 6 (adapted from [23]). In other words, the asynchronous training scheme separates the AC-NN's online operation from its training process. Since the global agent will not be updated frequently as those in the training process, the stability of this scheme is ensured. Meanwhile, the copied agents that are being trained in the background with the procedure in *Algorithm* 1 use emulated environments, which are randomly generated by taking snapshots of the historical status in the TED. Hence, the copied agents actually interact with different environments, which diversifies their training samples to explore the solution space more thoroughly.

IV. Performance Evaluation and Analysis

A. Simulation Setup

We evaluate our proposed Deep-NFVOrch with a DCI-EON that uses the NSFNET topology [41], which contains 14 nodes and 44 fiber links. We assume that the IT resource capacity of each DC is $C_v = 100$ and each fiber link can accommodate $F = 358$ FS', each of which has a bandwidth of 12.5 GHz. The DCI-EON supports 5 types of vNFs, which can form 10 types of vNF-SCs. The number of vNFs in a vNF-SC ranges within $[2, 4]$. Since we cannot find any reference traces to model the dynamic vNF-SC requests in a practical IDC, we just generate the dynamic requests based on the traces for real wide-area TCP connections in [42]. Specifically, for the TCP connections, we map their source and destination

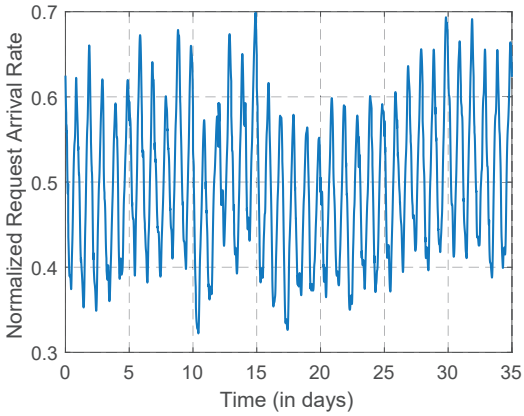

Fig. 7. Arrival rate of vNF-SC requests in simulations.

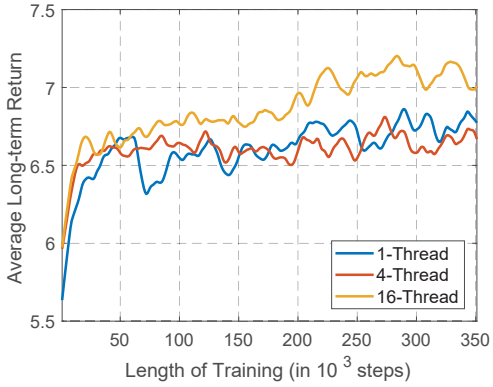

Fig. 8. Performance of asynchronous training.

IP addresses to the source and destination DCs of vNF-SC requests, map their application protocols to the types of vNF-SCs, and scale their arrival time, hold-on time and bandwidth requirements to get the corresponding parameters for the vNF-SC requests. For emulating the fluctuation of the workload in the DCI-EON, we make the arrival rate of the vNF-SC requests change according to a realistic traffic trace measured in a wide-area network [43], as shown in Fig. 7. The trace captures the multi-hour fluctuation of vNF-SC requests and each sample on it represents the average request load per hour. In the simulations, we set the range of ΔT_n as $[1, 10]$ hours.

We implement our DRL-Observer with TensorFlow 1.14.0, and program the DCI-EON environment and vNF-SC provisioning in it with Python. The simulations run on a personal computer that equips with Intel Xeon E5-2650 CPU, 128 GB RAM and four GTX 1080ti GPU cards. In each simulation, we treat the period that the DCI-EON handles the first 60% dynamic vNF-SC requests as the training phase, while the provisioning of the remaining 40% requests is the testing phase. The hyper-parameters are set as $\epsilon = 0.01$ and $\xi = 0.95$. Unless specifically stated, the weight coefficients in Eq. (6) have the same value as $\alpha = \beta = \gamma = 1$.

B. Training Performance

We first evaluate the performance of the asynchronous training of DRL-Observer. We set the batch size of the training samples as $M = 4$, *i.e.*, each training sample contains the information regarding four service cycles. We incorporate 1, 4 and 16 threads in the asynchronous training, and plot the average long-term return from DRL-Observer (calculated with Eq. (7)) in Fig. 8. We can see that for all the cases, the average long-term return has a sharp increase within $50,000$ training steps, while the multi-thread cases (*i.e.*, with 4 and 16 threads) provide much steeper slopes than the single-thread one. This confirms the benefit of our asynchronous training scheme. Meanwhile, it is interesting to notice that the 1-thread and 4-thread cases have comparable performance after $100,000$ training steps, *i.e.*, both of their average long-term returns converge and oscillate within $[6.5, 6.8]$ approximately. Nevertheless, the average long-term return from the 16-thread case still increases after $100,000$ training steps until it converges after around $250,000$ steps. This is because with more threads, the asynchronous training scheme makes DRL-Observer explore the solution space more thoroughly. In the simulations, the first $50,000$ training steps take $1,008$ seconds in a thread on average. After this, DRL-Observer can be considered as a trained one and be incorporated in Deep-NFVOrch for online operation and training.

C. Performance of Deep-NFVOrch

We then evaluate the performance of Deep-NFVOrch on provisioning dynamic vNF-SC requests in the DCI-EON. Here, we design three benchmarks. The first two benchmarks are based on fixed service cycles, which means that they fix ΔT_n as 1 (*i.e.*, the smallest possible value) and 10 (*i.e.*, the largest possible value), respectively, and keep the remaining design of Deep-NFVOrch. The third benchmark also determines ΔT_n adaptively (namely, the adaptive benchmark), but it does not do it with DRL-Observer. Specifically, it first chooses the durations of the first two service cycles (ΔT_1 and ΔT_2) randomly, and then determines $\Delta T_n (n \geq 3)$ as follows.

$$\Delta T_n = \begin{cases} \Delta T_{n-1} + 1, & \psi_{n-1} < \psi_{n-2} \text{ and } \Delta T_{n-1} \neq 10, \\ \Delta T_{n-1} - 1, & \psi_{n-1} > \psi_{n-2} \text{ and } \Delta T_{n-1} \neq 1, \\ \Delta T_{n-1}, & \Delta T_{n-1} = 1 \text{ or } \Delta T_{n-1} = 10, \end{cases} \quad (12)$$

where ψ_n denotes the number of requests arriving in the n-th service cycle. The main idea of this adaptive benchmark is to achieve load balancing among service cycles, *i.e.*, preventing the traffic load in each service cycle from becoming too high or too low. To measure the performance of the adaptive benchmark accurately, we run it with 10 different initial states (*i.e.*, initializing the values of ΔT_1 and ΔT_2 randomly for 10 times), and average the results as its performance metrics.

Fig. 9 shows the results on overall resource utilization in the DCI-EON at different workloads. Specifically, the simulations change the normalized arrival rate of vNF-SC requests over time as that in Fig. 7, while the maximum number of new vNF-SC requests per hour gets increased from 50 to 200. We observe that DRL-Observer provides the second-highest overall resource utilization among the four schemes. The

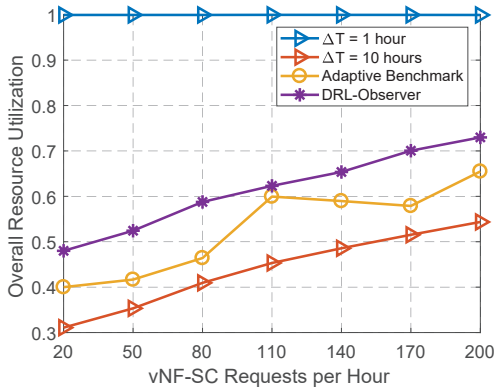

Fig. 9. Results on overall resource utilization.

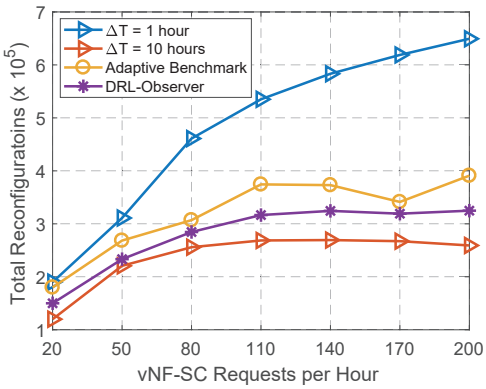

Fig. 10. Results on total reconfigurations.

reason why DRL-Observer can outperform the adaptive benchmark is that the adaptive benchmark adjusts ΔT_n only based on the fluctuation of request arrival rate, while the decision module in DRL-Observer determines ΔT_n based on both the request fluctuation and the network status. As Deep-NFVOrch with ΔT_n fixed as one hour reconfigures the DCI-EON most frequently, it achieves the highest overall resource utilization. However, this will actually cause tremendous overheads, which can be verified with the results in Fig. 10.

In the simulations, one reconfiguration means instantiating

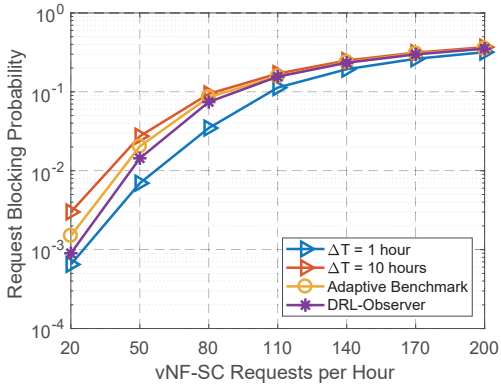

Fig. 11. Results on request blocking probability.

TABLE I
NETWORK PERFORMANCE UNDER DIFFERENT WEIGHT COEFFICIENTS

Group Index	Parameters (α, β, γ)	Resource Utilization	Blocking Probability $(\times 10^{-2})$	Reconfigurations $(\times 10^5)$
1	(1,1,1)	43.6%	1.44	2.28
2	(10,1,1)	65.4%	6.15	2.38
3	(1,10,1)	34.9%	0.97	2.19
4	(1,1,10)	39.84%	1.29	2.21
5	(10,10,10)	35%	2.82	2.20
6	$\Delta T = 1$	99.9%	0.7	3.11
7	$\Delta T = 10$	35.3%	2.74	2.20

a new vNF or establishing a new lightpath. In Fig. 10, the Deep-NFVOrch with $\Delta T_n = 1$ hour invokes the most reconfigurations, which are significantly more than those invoked by the one with DRL-Observer. The performance of DRL-Observer on total reconfigurations is also better than that of the adaptive benchmark, and it is just worse than the Deep-NFVOrch with $\Delta T_n = 10$ hours, which is actually expected. This is because the DCI-EON is reconfigured the least frequently, if Deep-NFVOrch fixes ΔT_n as 10 hours. To this end, by combining the results in Figs. 9 and 10, we can conclude that the Deep-NFVOrch with DRL-Observer achieves the best tradeoff between resource utilization and reconfiguration overheads among the four schemes. Finally, the request blocking probability in Fig. 11 also confirms the superiority of DRL-Observer, *i.e.*, its adaptivity to changes in the DCI-EON is better than that of the adaptive benchmark.

Although our DRL-Observer based approach balances the tradeoff among the performance metrics well, it can still sacrifice much on the blocking probability and resource utilization when being compared to the Deep-NFVOrch with $\Delta T_n = 1$ hour, especially when the request load is relatively low. The reason for this could be multi-fold, such as the used request traces, the design of the request predictor, the design of the DRL-Observer, and the offline and online training procedures. We will further optimize the system design in our future work to address these issues. Meanwhile, we hope to point out that reducing the number of network reconfigurations is actually very important in NC&M. This is because network reconfigurations not only shorten the life time of network elements but also cause most of the issues/errors/failures in a network [44]. From this perspective, we can see that the DRL-Observer based approach does have noticeable advantages.

D. Effects of Weight Coefficients

By designing the reward as that in Eq. (6), we enable DRL-Observer to balance the importance of multiple performance metrics of the DCI-EON (*i.e.*, the overall resource utilization, the request blocking probability, and the number of reconfigurations) with three weight coefficients α, β and γ. The simulations also investigate their effects, and the results can be leveraged to determine their values empirically. Table I shows the simulation results. Here, we choose 5 typical settings of the weight coefficients, and evaluate the performance of Deep-NFVOrch after the training of DRL-Observer has converged. Table I compares the network performance when the request arrival rate is set as 50 per hour. This is because the blocking

probability at this traffic load is around 10^{-2}, which mimics the situation in practical network operation. We also include the results from the fixed benchmarks with $\Delta T = 1$ and $\Delta T = 10$ as performance baselines.

The simulation results in Table I indicate that we can successfully make Deep-NFVOrch more favorable to one of the three performance metrics by increasing the corresponding weight coefficient. Meanwhile, since there are tradeoff among the performance metrics, the action would also degrade the rest of the performance metrics. For instance, *Group* 2 uses a higher α than that in *Group* 1, to make Deep-NFVOrch more favorable to the resource utilization. This results in that the resource utilization from *Group* 2 is significantly higher than that from *Group* 1, which is achieved by sacrificing certain performance on blocking probability and number of reconfigurations. Therefore, in real network operation, we can increase the corresponding weight coefficient to guide the optimization direction of Deep-NFVOrch, but in the meantime, we should also watch all the performance metrics and make sure that their values always fall into the acceptable regions.

The results in Table I also suggest that Deep-NFVOrch is more sensitive to the weight coefficient of resource utilization (*i.e.*, α). More specifically, when we change the value of a coefficient from 1 to 10, the effect of increasing α is the most significant. Hence, the value of α should be adjusted with more caution. The difference between *Groups* 1 and 5 indicates that both the relative and absolute values of the weight coefficients can affect the performance of Deep-NFVOrch. This is because applying coefficients with larger values increases the reward's absolute value, which would get DRL-Observer trapped in local optimums more easily during the training process.

V. Conclusion

In this work, we discussed how to improve the DRL-based adaptive service framework (*i.e.*, Deep-NFVOrch) for realizing high-performance vNF-SC in EON-DCIs. Specifically, Deep-NFVOrch works in service cycles, and tries to reduce the setup latency of vNF-SC by invoking request prediction and pre-deployment at the beginning of each service cycle. Therefore, we introduced a DRL-Observer to select the duration of each service cycle adaptively, which was designed based on A2C with an asynchronous training scheme. Our simulation results demonstrated that DRL-Observer converges quickly in online training with the help of a few asynchronous training threads, and the Deep-NFVOrch with it outperforms several benchmarks, in terms of balancing the tradeoff among overall resource utilization, vNF-SC request blocking probability, and number of network reconfigurations. Moreover, we also performed extensive simulations to discuss how to determine the values of weight coefficients in the reward formulation.

Acknowledgments

This work was supported in part by the NSFC projects 61871357 and 61701472, and CAS Key Project (QYZDY-SSW-JSC003).

References

[1] Cisco visual networking index: Forecast and methodology, 2017-2022. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html#_Toc529314186

[2] P. Lu, L. Zhang, X. Liu, J. Yao, and Z. Zhu, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[3] "Network functions virtualization (NFV)," Jan. 2012. [Online]. Available: https://portal.etsi.org/portal/server.pt/community/NFV/367

[4] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[5] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. of CloudNet 2014*, pp. 7–13, Oct. 2014.

[6] W. Fang, M. Zeng, X. Liu, W. Lu, and Z. Zhu, "Joint spectrum and IT resource allocation for efficient vNF service chaining in inter-datacenter elastic optical networks," *IEEE Commun. Lett.*, vol. 20, pp. 1539–1542, Aug. 2016.

[7] O. Gerstel, M. Jinno, A. Lord, and S. Yoo, "Elastic optical networking: A new dawn for the optical layer?" *IEEE Commu. Mag.*, vol. 50, pp. s12–s20, Feb. 2012.

[8] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[9] L. Gong, X. Zhou, X. Liu, W. Zhao, W. Lu, and Z. Zhu, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[10] Y. Yin, H. Zhang, M. Zhang, M. Xia, Z. Zhu, S. Dahlfort, and S. Yoo, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[11] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[12] C. Develder, M. De Leenheer, B. Dhoedt, M. Pickavet, D. Colle, F. De Turck, and P. Demeester, "Optical networks for grid and cloud computing applications," *Proc. IEEE*, vol. 100, pp. 1149–1167, May 2012.

[13] W. Fang, M. Lu, X. Liu, L. Gong, and Z. Zhu, "Joint defragmentation of optical spectrum and IT resources in elastic optical datacenter interconnections," *J. Opt. Commun. Netw.*, vol. 7, pp. 314–324, Mar. 2015.

[14] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

[15] Y. Wang, P. Lu, W. Lu, and Z. Zhu, "Cost-efficient virtual network function graph (vNFG) provisioning in multidomain elastic optical networks," *J. Lightw. Technol.*, vol. 35, pp. 2712–2723, Jul. 2017.

[16] X. Chen, Z. Zhu, J. Guo, S. Kang, R. Proietti, A. Castro, and B. Yoo, "Leveraging mixed-strategy gaming to realize incentive-driven VNF service chain provisioning in broker-based elastic optical inter-datacenter networks," *J. Opt. Commun. Netw.*, vol. 10, pp. A232–A240, Feb. 2018.

[17] W. Lu, L. Liang, and Z. Zhu, "Orchestrating data-intensive vNF service chains in inter-DC elastic optical networks," in *Proc. of ONDM 2017*, pp. 1–6, May 2017.

[18] X. Chen, Z. Zhu, R. Proietti, and B. Yoo, "On incentive-driven VNF service chaining in inter-datacenter elastic optical networks: A hierarchical game-theoretic mechanism," *IEEE Trans. Netw. Serv. Manag.*, vol. 16, pp. 1–12, Mar. 2019.

[19] J. Yin, J. Guo, B. Kong, H. Yin, and Z. Zhu, "Experimental demonstration of building and operating QoS-aware survivable vSD-EONs with transparent resiliency," *Opt. Express*, vol. 25, pp. 15 468–15 480, 2017.

[20] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proc. of NOMS 2014*, pp. 1–9, May 2014.

[21] S. Krishnan and R. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," *IEEE/ACM Trans. Netw.*, vol. 21, pp. 2001–2014, Dec. 2013.

[22] 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," *J. Opt. Commun. Netw.*, vol. 10, pp. D29–D41, Oct. 2018.

[23] B. Li, W. Lu, and Z. Zhu, "Deep-NFVOrch: Deep reinforcement learning based service framework for adaptive vNF service chaining in IDC-EONs," in *Proc. of OFC 2019*, pp. 1–3, Mar. 2019.

[24] Z. Zhu, C. Chen, S. Ma, L. Liu, X. Feng, and S. Yoo, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[25] S. Liu, W. Lu, and Z. Zhu, "On the cross-layer orchestration to address IP router outages with cost-efficient multilayer restoration in IP-over-EONs," *J. Opt. Commun. Netw.*, vol. 10, pp. A122–A132, Jan. 2018.

[26] W. Lu, X. Yin, X. Cheng, and Z. Zhu, "On cost-efficient integrated multilayer protection planning in IP-over-EONs," *J. Lightw. Technol.*, vol. 35, pp. 5335–5346, Dec. 2017.

[27] X. Chen, B. Li, R. Proietti, Z. Zhu, and B. Yoo, "Self-taught anomaly detection with hybrid unsupervised/supervised machine learning in optical networks," *J. Lightw. Technol.*, vol. 37, pp. 1742–1749, Apr. 2019.

[28] L. Velasco, B. Shariati, F. Boitier, P. Layec, and M. Ruiz, "A learning life-cycle to speed-up autonomic optical transmission and networking adoption," *J. Opt. Commun. Netw.*, vol. 11, pp. 226–237, May 2019.

[29] X. Chen, R. Proietti, H. Lu, A. Castro, and B. Yoo, "Knowledge-based autonomous service provisioning in multi-domain elastic optical networks," *IEEE Commun. Mag.*, vol. 56, pp. 152–158, Aug. 2018.

[30] T. Panayiotou, K. Manousakis, S. Chatzis, and G. Ellinas, "A data-driven bandwidth allocation framework with QoS considerations for EONs," *J. Lightw. Technol.*, vol. 37, pp. 1853–1864, May 2019.

[31] X. Chen, B. Li, R. Proietti, H. Lu, Z. Zhu, and B. Yoo, "DeepRMSA: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," *J. Lightw. Technol.*, vol. 37, pp. 4155–4163, Aug. 2019.

[32] R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5G flexible RAN," *J. Lightw. Technol., in Press*, 2019.

[33] V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. of ICML 2016*, pp. 1928–1937, Jun. 2016.

[34] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/first/the-book.html

[35] L. Gong, X. Zhou, W. Lu, and Z. Zhu, "A two-population based evolutionary approach for optimizing routing, modulation and spectrum assignments (RMSA) in O-OFDM networks," *IEEE Commun. Lett.*, vol. 16, pp. 1520–1523, Sept. 2012.

[36] H. Fang, W. Lu, Q. Li, J. Kong, L. Liang, B. Kong, and Z. Zhu, "Predictive analytics based knowledge-defined orchestration in a hybrid optical/electrical datacenter network testbed," *J. Lightw. Technol., in Press*, 2019.

[37] S. Liu, B. Niu, D. Li, M. Wang, S. Tang, J. Kong, B. Li, X. Xie, and Z. Zhu, "DL-assisted cross-layer orchestration in software-defined IP-over-EONs: From algorithm design to system prototype," *J. Lightw. Technol.*, vol. 37, pp. 4426–4438, Sept. 2019.

[38] A. Coates and A. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *Proc. of ICML 2011*, pp. 921–928, Jul. 2011.

[39] D. Kingma and J. Ba, "Adam: A method for stochastic optimization (v9)," Jan. 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[41] W. Lu and Z. Zhu, "Dynamic service provisioning of advance reservation requests in elastic optical networks," *J. Lightw. Technol.*, vol. 31, pp. 1621–1627, May 2013.

[42] V. Paxson, "Empirically derived analytic models of wide-area TCP connections," *IEEE/ACM Trans. Netw.*, vol. 2, pp. 316–336, Aug. 1994.

[43] S. Liu and Z. Zhu, "Generating data sets to emulate dynamic traffic in a backbone IP over optical network," *Tech. Rep.*, 2019. [Online]. Available: https://github.com/lsq93325/Traffic-creation/blob/master/README.md?tdsourcetag=s_pctim_aiomsg

[44] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or die: High-availability design principles drawn from Google's network infrastructure," in *Proc. of ACM SIGCOMM 2016*, pp. 58–72, Aug. 2016.