A Dynamic and Collaborative Multi-Layer Virtual Network Embedding Algorithm in SDN Based on Reinforcement Learning

Meilian Lu, *Member, IEEE*, Yun Gu, and Dongliang Xie, *Member, IEEE*

***Abstract*—Most of existing virtual network embedding (VNE) algorithms only consider how to construct virtual networks more efficiently on a physical infrastructure, without considering the possibility that the constructed virtual networks may be further virtualized to multiple smaller ones. We define the former scenario as single-layer VNE and the later as multi-layer VNE. As the increasing popularity of deploying large datacenter networks and wide area networks with Software Defined Network (SDN) architectures, it becomes a new requirement and possibility to provide multi-layer encapsulated network services for large tenants who have hierarchical organizational structures or need fine-grained service isolation. However, existing VNE algorithm are not specifically designed for the above requirement and not flexible enough to deal with mapping virtual network requirements (VNRs) to a physical network and smaller VNRs to a mapped virtual network. In this paper, we aim to propose a unified and flexible multi-layer VNE algorithm combining with reinforcement learning to solve the embedding of multi-layer VNRs, which can better distinguish the differences between VNRs and physical networks. Simulation results show that our algorithm achieves good performance both in single-layer and multi-layer VNE scenarios.**

***Index Terms*—Dynamic and collaborative embedding, multi-layer virtual network embedding, multi-dimensional attributes, reinforcement learning.**

I. Introduction

With the rapid development of cloud computing, Internet of Things and 5G, it is increasingly difficult to deploy new network technologies and protocols in the traditional IP network structures.

The combination of Software Defined Network (SDN) and Network Virtualization (NV) is considered as an effective way to overcome current network ossification and promote future network innovation. SDN [1] is a new network architecture which separates control plane and data plane of network elements. NV [2]–[3] introduces virtualization technology into the network environment, allowing multiple virtual networks to be built and run on a shared physical network. NV decouples software-based virtual networks from hardware-based physical networks, so it is much easier to be implemented in SDN.

A key issue for NV is how the resources of substrate network should be allocated for multiple virtual network requests (VNRs) with topology and resource constraints. It is also called Virtual Network Embedding (VNE) or Virtual Network Mapping (VNM) problem.

Fischer *et al.* [4] made a comprehensively classification on existing algorithms for mapping single-layer VNRs onto a single-layer physical network, while Zhang *et al.* [5] and Chowdhury *et al.* [6] studied the VNE problem on multi-layer physical networks. Although Nguyen *et al.* [7] focused on multi-layer VNRs, their algorithm only solves the virtual link mapping by assuming that all the locations of virtual nodes are fixed. Besides, they simplified the multi-layer VNE problem as multiple single-layer VNE problems that can be solved from bottom to top because higher layer VNRs are already known in advance.

However, the arrival and departure of VNRs are unpredictable because tenants apply and destroy VNRs on each layer dynamically according to their business requirements in real scenarios. Besides, the topology and total resources of the physical network are static, but those of the accepted VNRs are dynamic, so the mapping results on the physical network can be adjusted when an under VNR fails to accept its upper requests. Therefore, multi-layer VNE algorithm should be flexible enough to adopt different mapping policies for different substrate networks and update existing policies for different VNRs.

Aiming at the above requirements, we study the multi-layer VNE problem and propose a dynamic and collaborative multi-layer VNE algorithm in SDN based on reinforcement learning. The main contributions of this paper are listed as follows.

1) We proposed a multi-layer VNE framework in SDN, which can provide the mapping from VNRs to underlying physical network and upper VNRs to under VNRs.
2) We build a reinforcement learning model named MLRL-Model with the goal of increasing mapping revenues and reducing mapping costs.
3) We further propose a dynamic and collaborative mapping mechanism that gives priority to increasing the resource capacity of particular under virtual nodes/links or migrating them to other physical ones with remapping the entire network as the complementary operation.

Manuscript received June 24, 2019; revised January 23, 2020 and July 1, 2020; accepted July 23, 2020. Date of publication July 29, 2020; date of current version December 9, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61602048. The associate editor coordinating the review of this article and approving it for publication was H. De Meer. *(Corresponding author: Meilian Lu.)*

The authors are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: mllu@bupt.edu.cn; gooyunn@qq.com; xiedl@bupt.edu.cn).

Digital Object Identifier 10.1109/TNSM.2020.3012588

4) We design and implement a VNE algorithm simulator which is used to compare our proposed VNE algorithm with three chosen single-layer VNE algorithms [8]–[10] for performance evaluation.

The rest of this paper is organized as follows: Section II introduces the related work. Section III describes the multi-layer VNE problem. Section IV elaborates our proposed multi-layer VNE framework and the corresponding algorithms. Section V reports the simulation experiments and results. Section VI concludes this paper.

II. Related Work

Existing single-layer VNE algorithms can roughly be divided into exact, heuristic and meta-heuristic algorithms according to their optimization strategies. Exact algorithms [11]–[12] are based on Integer Linear Programming (ILP) or Mix Integer Programming (MIP) models and can get optimal solutions in a reasonable time when the problem size is small. However, when the network scale becomes larger, the above algorithms cannot obtain mapping solutions in a reasonable time and space any more. Therefore, many heuristic algorithms [8], [13]–[15] were proposed to get an approximate optimal solution. Furthermore, considering that heuristic algorithms are easy to fall into local optimal solutions, meta-heuristic algorithms were proposed to improve the situation by combining random search with local search. Current meta-heuristic VNE algorithms include particle swarm optimization [16]–[18], ant colony optimization [19]–[21], artificial bee colony [22]–[23], Monte Carlo Tree Search [9], artificial neural network [10] and so on.

Different from above algorithms, some researchers further explored the VNE problem in more complex network scenarios. Both Zhang *et al.* [5] and Chowdhury *et al.* [6] studied on mapping single-layer VNRs onto multi-layer physical networks such as IP-over optical networks, which is different from our objective. Demirci and Ammar [24] and Mijumbi *et al.* [25] respectively proposed their VNE algorithms in SDN by considering the exclusive network properties of SDN. Nguyen *et al.* [7] tried to solve the mapping problem of multi-layer VNRs on a physical network. However, they supposed the locations of virtual nodes are already pre-defined and cannot be mapped arbitrarily, so they only focused on mapping virtual links. In addition, they assumed that the Service Level Agreement of a VNR is expressed by a traffic demand matrix between different pre-defined virtual nodes, and the traffic matrix of a lower layer VNR is calculated based on that of higher layer VNRs. Once traffic matrices of VNRs in all levels are known, the multi-path mapping algorithm is performed starting from layer 1 towards to higher layers. Thus, the multi-layer mapping problem is simplified as mapping VNRs onto the underlying substrate network.

Our algorithm proposed in this paper differs from existing ones in three ways. Firstly, we use a policy network based on reinforcement learning to obtain an optimal mapping policy once a new VNR arrives. Secondly, when there is a failure of mapping an upper VNR to its corresponding under VNR, we can dynamically adjust the under VNR and try its best to accept the upper VNR, which coordinates the mappings between different layers and improves the flexibility and efficiency of multi-layer VNE. At last, we consider more kinds of virtualized SDN resources including switch CPU, switch flow entry space, switch queue storage, and link bandwidth.

III. Problem Description and Modeling

A. Problem Description

In the scenario of multi-layer VNE problem, VNRs constitute a multi-layered architecture and can be divided into two types: under VNRs and upper VNRs. The under VNRs are directly mapped onto the physical network, while upper VNRs are mapped onto an accepted under VNR.

As mentioned in Section II, current VNE algorithms are mostly designed for single-layer VNRs, only few for multi-layer VNRs such as the method proposed by Nguyen *et al.* [7]. However, they assumed that all upper VNRs are known in advance and the locations of virtual nodes in each VNR are pre-defined, which may not match the actual situation. Besides, although the multi-layer VNE problem can be decomposed into multiple single-layer VNE problems, the existing algorithms cannot be employed to solve our problem directly due to the following reasons.

(1) Exact algorithms need to formulate mathematical programming models, so the complexity of multi-layer VNE problem will make it very time-consuming.

(2) Heuristic and meta-heuristics algorithms aim to obtain approximate optimal solutions with low execution time. However, with the arrival and departure of VNRs, the resource status of the substrate network may change dynamically over time, while those algorithms always adopt a fixed mapping policy, which cannot be dynamically adjusted. For multi-layer VNE problem, it is more unreasonable to perform fixed mapping policy all the time.

(3) The multi-layer VNE problem includes the mapping between upper VNRs and under VNRs in addition to the mapping between VNRs and the physical network. Moreover, the topology and total resources of the physical network are static, but the accepted substrate VNRs can dynamically adjust their current mapping results to accept more upper VNRs. Therefore, the mapping policy should be more flexible.

B. Network Model

We model the physical network as a weighted undirected graph and denote it by $G^S = (N^S, L^S, A_n^S, A_l^S)$, where N^S and L^S denote the set of physical nodes and links respectively, A_n^S and A_l^S denote their resource attributes respectively. In SDN, switches forward data according to the flow rules generated by controller, so the flow entry space in each switch is an important resource, which should be carefully considered during the node mapping stage. Besides, the more granularities the virtualized resources are, the better the isolation between different VNRs is. Therefore, in this paper, A_n^S consists of CPU power $c(n^S)$, flow entry space $f(n^S)$, and queue storage $q(n^S)$, while A_l^S consists of bandwidth $b(l^S)$.

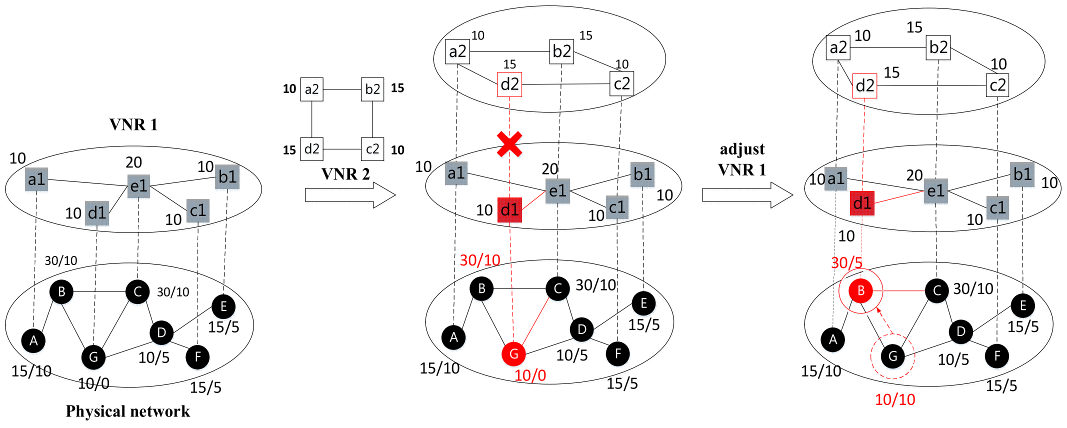


Fig. 1. An instance of Multi-layer VNE.

Similar to the physical network, we also model virtual networks as weighted undirected graphs and denote each of them by $G^V = (N^V, L^V, C_n^V, C_l^V)$, where N^V and L^V denote the set of virtual nodes and links respectively, C_n^V and C_l^V denote the resource constraints of them respectively. C_n^V consists of CPU power $c(n^V)$, flow entry space $f(n^V)$, and queue storage $q(n^V)$, while C_l^V consists of bandwidth $b(l^V)$. Then, a VNR can be denoted as a triple $V(G^V, t_a, t_d)$, where t_a denotes its arriving time and t_d denotes its duration time.

C. Multi-Layer VNE Model

The mathematical formulation of single-layer VNE is to bulid a mapping $M : G^V \to G_{sub}^S(N_{sub}^S, P_{sub}^S)$ between G^V and G^S, where G_{sub}^S is a subgraph of G^S, denoting the topology of G^V on G^S, N_{sub}^S is the subset of N^S, P_{sub}^S denotes the subset of P^S which is the set of loopless paths in the substrate network.

In multi-layer VNE problem, both the physical network and an accepted under VNR can be regarded as the substrate networks. Therefore, the mathematical formulation of multi-layer VNE includes the mappings of $M : G^V \to G_{sub}^S(N_{sub}^S, P_{sub}^S)$ and $M' : G^{V'} \to G_{sub}^V(N_{sub}^V, P_{sub}^V)$, where M denotes the mapping between an under VNR G^V and the physical network G^S, M' denotes the mapping between an upper VNR $G^{V'}$ and an under VNR G^V. In order to reduce the complexity, we adopt the idea of two-step mapping. Therefore, M can be splitted into two subproblems: node mapping $M_N : N^V \to N_{sub}^S$ and link mapping $M_L : L^V \to P_{sub}^S$. Similarly, M' can also be splitted into $M'_N : N^{V'} \to N_{sub}^V$ and $M'_L : L^{V'} \to P_{sub}^V$.

Considering that the resources of the physical network and the demands of VNRs are limited, some restrictions need to be satisfied when solving the mapping of nodes and links. In addition, for the sake of unified description, we do not distinguish between the physical network and an accepted under VNR, giving them a unified name, substrate networks.

(1) Any two different virtual nodes cannot be mapped to the same substrate node.

$$\begin{cases} M_N(n^V) = M_N(m^V), & if\ n^V = m^V \\ M_N(n^V) \neq M_N(m^V), & if\ n^V \neq m^V \end{cases} \tag{1}$$

(2) The resource demands of any virtual node should not exceed the residual availbable resources of the selected substrate node that host the virtual node.

$$\begin{cases} c(n^V) \le c_a(M_N(n^V)) \\ f(n^V) \le f_a(M_N(n^V)) \\ q(n^V) \le q_a(M_N(n^V)) \end{cases} \tag{2}$$

(3) The bandwidth demand of any virtual link should not exceed the residual availbable bandwidth of the selected substrate link that host the virtual link.

$$b\left(l^V\right) \le \min_{l^S \in M_L(l^V)} b_a\left(l^S\right) \tag{3}$$

Fig. 1 shows an instance of multi-layer VNE. The physical network is composed of seven black nodes. When VNR1 with five nodes arrives, a feasible mapping solution is:

The node mapping:

$$\{a_1 \to A, b_1 \to E, c_1 \to F, d_1 \to G, e_1 \to C\}$$

The link mapping :

$$\{(a_1, e_1) \to (A, B, C), (d_1, e_1) \to (G, C),$$
$$(c_1, e_1) \to \quad (F, D, C), (b_1, e_1) \to (E, D, C)\}$$

Then, VNR2 arrives, whose substrate network is VNR1. From Fig. 1, we can see that the topologies of the physical network, VNR1 and VNR2 are all different from each other, which means the mapping policy of the physcial network may not apply to VNR1. Therefore, the VNE algorithm should adjust its mapping policy according to different VNRs and substrate networks with up-to-date resource status.

A possible mapping solution for VNR2 is as follows.

The node mapping: $\{a_2 \to a_1, b_2 \to e_1, c_2 \to c_1, d_2 \to d_1\}$

The link mapping:

$$\{(a_2, b_2) \to (a_1, e_1), (b_2, c_2) \to (e_1, c_1),$$
$$(c_2, d_2) \to (c_1, e_1, d_1), (a_2, d_2) \to (a_1, e_1, d_1)\}$$

Obviously, VNR2 will be rejected because the node d_1 in VNR1 cannot satisfy the resource requirement of the node d_2 in VNR2. However, VNR2 can be mapped successfully if we adjust the mapping result of VNR1. For example, we can migrate d_1 from physical node G whose resource is insufficient

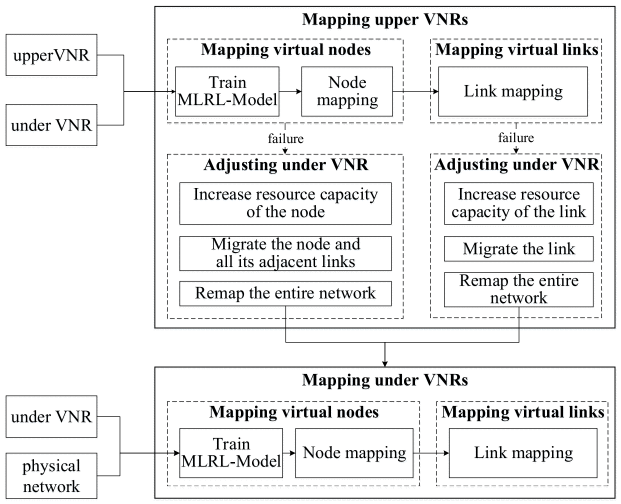


Fig. 2. The Algorithm Framework of ML-VNE.

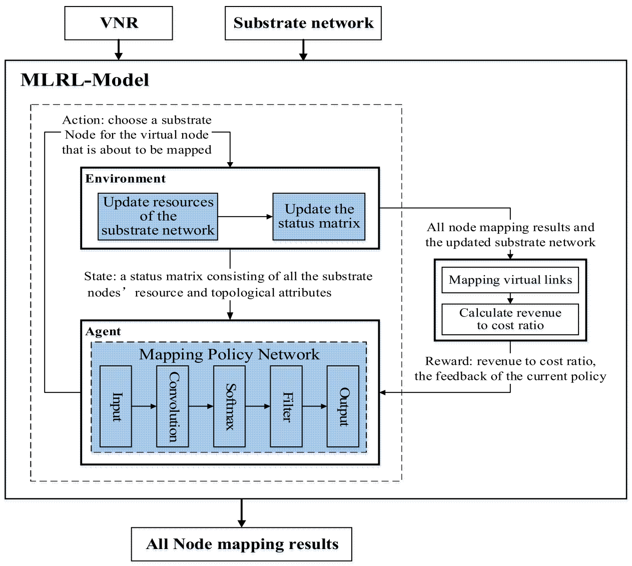

VNR
Substrate network
MLRL-Model
Action: choose a substrate Node for the virtual node that is about to be mapped
Environment
Update resources of the substrate network
Update the status matrix
All node mapping results and the updated substrate network
Mapping virtual links
Calculate revenue to cost ratio
State: a status matrix consisting of all the substrate nodes' resource and topological attributes
Reward: revenue to cost ratio, the feedback of the current policy
Agent
Mapping Policy Network
Input
Convolution
Softmax
Filter
Output
All Node mapping results

Fig. 3. MLRL-Model for Mapping Nodes.

to physical node *B* whose resource is sufficient, and remap its adjacent link (d_1, e_1). After this adjustment, d_1 can satisfy the resource requirement of d_2 by leasing more resource from *B*.

D. Objectives

In single-layer VNE problem, two objectives are mainly considered: reducing mapping costs and increasing revenues. For the multi-layer VNE problem in this paper, we also consider these two objectives of different layers.

1) Reduce Mapping Costs: Mapping costs are measured by the network resources occupied by VNRs. However, each kind of network resource has a different unit of measurement. In order to treat the contribution of different resources fairly, we firstly normalize them before calculating.

For $\forall n^V \in N^V$,

$$\begin{cases} \bar{c}\left(n^V\right) = \frac{c\left(n^V\right)}{\sum_{n^V \in N^V} c\left(n^V\right)} \\ \bar{f}\left(n^V\right) = \frac{f\left(n^V\right)}{\sum_{n^V \in N^V} f\left(n^V\right)} \\ \bar{q}\left(n^V\right) = \frac{q\left(n^V\right)}{\sum_{n^V \in N^V} q\left(n^V\right)} \end{cases} \quad (4)$$

where $c(n^V)$, $f(n^V)$, and $q(n^V)$ are respectively the requirements of the CPU, flow entry space and queue storage of n^V; $\sum_{n^V \in N^V} c(n^V)$, $\sum_{n^V \in N^V} f(n^V)$, and $\sum_{n^V \in N^V} q(n^V)$ are the sum of all nodes respectively.

For $\forall l^V \in L^V$,

$$\bar{b}\left(l^V\right) = \frac{b\left(l^V\right)}{\sum_{l^V \in L^V} b\left(l^V\right)} \quad (5)$$

where $b(l^V)$ is the bandwidth requirement of l^V, $\sum_{l^V \in L^V} b(l^V)$ is the sum of all virtual links.

Therefore, we formulate the revenue achieved by mapping a VNR as the sum of VNR resource requirements as follows.

$$R = \sum_{n^V \in N^V} \left(\bar{c}\left(n^V\right) + \bar{f}\left(n^V\right) + \bar{q}\left(n^V\right)\right) + \sum_{l^V \in L^V} \bar{b}\left(l^V\right) \quad (6)$$

The cost incurs by mapping a VNR is formulated as the sum of the resources that the substrate network actually allocates to it as follows.

$$C = \sum_{n^V \in N^V} \left(\bar{c}\left(n^V\right) + \bar{f}\left(n^V\right) + \bar{q}\left(n^V\right)\right) + \sum_{l^V \in L^V} \sum_{l^S \in L^S} f\left(l^V, l^S\right) \bar{b}\left(l^V\right) \quad (7)$$

where $f(l^V, l^S)$ is a binary value, denoting whether the virtual link l^V is mapped to the substrate link l^S.

2) Increase Mapping Revenues: Higher mapping revenues means more VNRs are mapped successfully. The ratio of the number of accepted VNRs to the number of arrived VNRs defines the acceptance ratio.

$$p_a = \lim_{T \to \infty} \frac{\sum_{t=0}^{T} V_a}{\sum_{t=0}^{T} V} \quad (8)$$

where, $\sum_{t=0}^{T} V_a$ denotes the number of the successfully mapped VNRs during $t = 0$ to *T*, $\sum_{t=0}^{T} V$ denotes the total number of the arrived VNRs during the same time interval.

Besides, the long-term revenue to cost ratio can more reasonably reflect the comprehensive performance of a VNE algorithm. It is defined as the ratio of the sum of revenues to the sum of costs during a period of time.

$$R/C = \lim_{T \to \infty} \frac{\sum_{t=0}^{T} R\left(G^V\right)}{\sum_{t=0}^{T} C\left(G^V\right)} \quad (9)$$

From (9), we can see that a higher long-term revenue to cost ratio means the VNE algorithm can achieve more profit while cost less network resources.

IV. The Proposed Multi-Layer VNE Algorithm Based on Reinforcement Learning

As mentioned in Section III, multi-layer VNE algorithm should be flexible enough to dynamically adjust its mapping policy over time, so we introduce reinforcement learning (RL) into multi-layer VNE problem and propose a RL model-based algorithm framework ML-VNE to provide self-renewal ability. Considering that network nodes have multiple attributes and it is much feasible to build RL model based on them, our algorithm includes a two-stage mapping process in which virtual link mappings are done only after the preceding virtual node mappings have concluded successfully. This applies to all the VNRs whether they are upper or under VNRs.

Specifically, we design a MLRL-Model and train it before mapping the nodes of each VNR, making the mapping policy can be updated in real time. Moreover, we propose a dynamic and collaborative mapping mechanism for upper VNRs in ML-VNE. When failing to satisfy the resource requirements of an upper VNR, the corresponding under VNR will adjust itself by following the principle that giving priority to increasing the resource capacity of particular under virtual nodes/links or migrating them to other physical nodes/links with remapping the entire network as the complement operation. Fig. 2 shows the framework of ML-VNE.

Next we will introduce our MLRL-Model, and then elaborate the ML-VNE algorithm base on RL.

A. MLRL-Model

In RL model, agent outputs a series of actions in a changing environment, which can be used to solve the sequential decision-making problems. Therefore, we model the virtual node mapping stage as a Markov decision process and introduce a RL model to lean an optimal node mapping policy for each VNR based on current substrate network with up-to-date residual resources.

RL is an interactive learning process that obtains feedbacks from the environment through the interaction between the agent and the environment. Its noteworthy feature is trial-and-error and delayed rewards. The agent gives an action by observing the status of the environment, and the environment returns a reward to the agent after performing the action. In many tasks, the reward is delayed, which means it is not only related to current action, but also may be related to subsequent actions. In this paper, we customize the basic reinforcement learning model and design our MLRL-Model for embedding as Fig. 3 shows. It is worth noting that in MLRL-Model, the substrate network is the physical network when handling an under VNR while it turns to be an under VNR when handling an upper VNR.

1) Environment State: With the arrival and departure of VNRs, the residual resources of the substrate network will change over time. Furthermore, network nodes usually have more topological attributes. Based on these considerations, we firstly build the node vectors for the substrate network with their resource and topological attributes and then combine all these node vectors as a matrix to denote the environment state in MLRL-model.

For $\forall n_i^S \in N^S$, the node vector contains four kinds of resource attributes and three kinds of topological attributes.

① Residual CPU capacity $C_i = c(n_i^S)$

② Residual flow entry space $F_i = f(n_i^S)$

③ Residual queue storage $Q_i = q(n_i^S)$

④ Sum of residual bandwidth of all adjacent links

$$L_i = \sum_{n_j^S \in adj(n_i^S)} b\left(l_{ij}^S\right)$$

⑤ Degree Centrality $D_i = \frac{Degree(n_i^S)}{m-1}$

$Degree(n_i^S)$ denotes the number of adjacent links of node n_i^S and m is the number of substrate nodes. A node is more reachable to other nodes if it has more adjacent links.

⑥ Closeness centrality $J_i = \frac{m-1}{\sum_{j=1}^{m} d_{ij}}$

d_{ij} denotes the hops of a shortest path whose endpoints are n_i^S and n_j^S. This attribute reflects the closeness of a node to other nodes.

⑦ Betweenness centrality $B_i = \sum_{j \neq i \neq k} \frac{g_{jk}(i)}{g_{jk}}$

$g_{jk}(i)$ denotes the number of shortest paths from n_j^S to n_k^S passing through n_i^S, g_{jk} denotes the number of all shortest paths from n_j^S to n_k^S. This attribute reflects that how many times a node acts as an intermediary in the shortest path of any other two different nodes. The node will get a higher betweeness centrality if it acts more times as the intermediary.

Obviously, D_i, J_i, and B_i range from 0 to 1 while C_i, F_i, Q_i, and L_i are not in the same order of magnitude. Therefore, in order to treat the contribution of different features fairly, we first normalize the four resource attributes as follows:

$$\begin{cases} \bar{C}_i = \frac{C_i}{\sum_{n_j^S \in N^S} C_j} \\ \bar{F}_i = \frac{F_i}{\sum_{n_j^S \in N^S} F_j} \\ \bar{Q}_i = \frac{Q_i}{\sum_{n_j^S \in N^S} Q_j} \\ \bar{L}_i = \frac{L_i}{\sum_{n_j^S \in N^S} L_j} \end{cases} \tag{10}$$

Then, each substrate node n_i^S can be denoted as a 7-dimensional vector.

$$\mathbf{Vec}_i = \left(\bar{C}_i, \bar{F}_i, \bar{Q}_i, \bar{L}_i, D_i, J_i, B_i\right) \tag{11}$$

We further concatenate all node vectors into a status matrix.

$$M = (\mathbf{Vec}_1, \mathbf{Vec}_2, \mathbf{Vec}_3, \ldots, \mathbf{Vec}_m)^T \tag{12}$$

This matrix serves as an input to the learning agent and is updated along with the changing substrate network.

2) Mapping Policy Network: Policy-Based and Value-Based methods are the two most important control methods in RL. Policy-Based methods directly predict the action that should be taken in a certain environment state, while Value-Based methods predict all the expected values (Q value) of candidate actions and select the action with the highest Q value. Value-Based methods are suitable for learning tasks whose environment only accepts a few discrete actions, while Policy-Based methods are more general and suitable for learning tasks whose environment accepts actions with a large quantity or continuous values. Deep neural network not only has a strong ability of nonlinear approximation, but also provides an end-to-end learning method, which can directly get the results of classification or regression from raw data. Therefore, with the breakthrough of deep learning in recent years, RL has also rapidly developed. After combining deep learning, the Policy-Based method is developed as policy network while the Value-Based method is developed as value network.

In multi-layer VNE problem, the action in RL corresponds to selecting a suitable substrate node for each virtual node. Besides, the action space is very large when the substrate network has many nodes. Therefore, the Policy-Based method is more appropriate. The policy network is essentially a neural network. It takes an environment state as input and outputs the probability distribution of all candidate actions through forward propagation. We build a simple policy network with basic elements of a neural network as shown in Fig. 3. It contains an input layer, a convolutional layer, a softmax layer, a filter layer and an output layer.

a) Input layer: The role of the input layer is to read the latest state of the environment, namely the status matrix calculated by (12). After a virtual node is mapped, it will pre-occupy the residual resource of a substrate node. In order to ensure that the adjacent links of this virtual node can be successfully mapped, it will also pre-occupy the bandwidth resources of the adjacent links of the certain substrate node. Therefore, *M* is re-calculated as a new input data before mapping a new virtual node.

b) Convolutional layer: This layer performs a convolution operation on the input to produce a vector representing the appropriate degree of each substrate nodes for certain virtual node. For substrate node n_i^S, the output is computed as:

$$y_i = Re\mathrm{LU}(w \cdot \mathbf{Vec}_i + b) = \begin{cases} w \cdot \mathbf{Vec}_i + b, & \text{if } w \cdot \mathbf{Vec}_i + b \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (13)$$

where *w* and *b* are the convolution kernel weight vector and the bias respectively.

c) Softmax layer: The output of the convolution layer is transmitted to this layer to produce a probability for each substrate node, which indicates the likelihood of yielding a better result if mapping a virtual node to it. For substrate node n_i^S, the probability is:

$$p_{i_{SL}} = \frac{e^{y_i}}{\sum_i e^{y_i}} \quad (14)$$

d) Filter layer: This layer is responsible for selecting the candidate substrate nodes. Firstly, the nodes that could not meet the resource requirements of a certain virtual node should be removed from the candidate set. Secondly, because multiple virtual nodes in a VNR cannot be mapped onto the same substrate node, the substrate nodes that have already host other virtual nodes of the same VNR should also be removed from the candidate set. Therefore, the probability of mapping a virtual node to the substrate node n_i^S is converted by the filter layer as:

$$p_i = \begin{cases} 0, & n_i^S \in N^S_{chosen} \bigcup (N^S - N^S_{candidate}) \\ p_{i_{SL}}, & \text{otherwise} \end{cases} \quad (15)$$

where N^S_{chosen} is the set of substrate nodes that have host virtual nodes for a VNR, $N^S_{candidate}$ is the set of substrate nodes that meet the resource requirements of current virtual node. Therefore, if a substrate node n_i^S has hosted other virtual node or cannot satisfy the resource requirements of current virtual node, its corresponding probability will be set to zero.

e) Output layer: This layer outputs a probability distribution as follows:

$$\mathbf{P} = (p_1, p_2, p_3, \ldots, p_m) \quad (16)$$

Each item in (16) corresponds to the selective probability of a substrate node. If mapping a virtual node to it can yield a better result, the probability can achieve a higher value.

3) Reward Function: The agent in MLRL-Model needs to learn a mapping policy $\pi_\theta(s, a)$ by trying many times. According to this policy, the agent can choose a suitable substrate node for a virtual node. Therefore, the performance of the mapping policy depends on the long-term cumulative rewards obtained after all virtual nodes have been mapped. However, although all virtual nodes are mapped successfully, it does not mean that all virtual links can also be mapped successfully. Therefore, it makes no sense to calculate the reward every time a virtual node mapping action is performed. Based on the Monte Carlo reinforcement learning method, we firstly obtain a mapping solution of all virtual nodes through the mapping policy network, then performs virtual links mapping, and finally calculates a reward based on the virtual node and link mapping solutions as the long-term cumulative reward.

As mentioned in Section III, reducing mapping costs and increasing mapping revenues are two main goals. A good mapping policy should achieve better results on them, so we define the following reward function.

$$Reward = \begin{cases} \frac{R}{C} & \text{if VNR is mapped successfully} \\ -\infty & \text{otherwise} \end{cases} \quad (17)$$

If the mapping solution based on a mapping policy that can generate more revenue and pay less cost, this mapping policy is better. Of course, if no mapping solution can be achieved base on a certain policy, the reward function set as an infinitely small value to avoid the agent selecting this policy next time.

4) Model Training: Through the above definitions, it can be found that the policy network is the core of MLRL-Model. Since the policy network is essentially a neural network model, in order to achieve better outputs, we train it using a powerful learning algorithm. The Back propagation (BP) algorithm is one of the most outstanding representative and successful neural network algorithms. BP algorithm evolves into Policy Gradient in reinforcement learning. During the training process, the policy network may output good actions and get the high expected values, or output bad actions and get the low expected values. Therefore, by learning from these samples, the policy network will increase the probability of selecting a good action and reduce the probability of selecting a bad action.

Algorithm 1 shows the pseudo of training MLRL-Model. The input data is a VNR and the corresponding substrate network. We can see that by training MLRL-Model, we will get multiple node mapping samples and each sample is achieved from the policy network that is trained with the last sample.

Algorithm 1 Training MLRL-Model

```
Input: $G^S$, substrate network
       $G^V$, VNR
       numSample, sample times
       $\alpha$, learning rate
Output: $\theta = (\omega, b)$, the parameters of the policy network
1: initiailize $\theta$ in the policy network where $\omega$ follows a normal distribution and $b = 0$
2: for $i = 1, 2, \ldots, numSample$ do
3:   for $t = 1, 2, 3, \ldots, T$ do
4:     get the $t-$th virtual node $n_t^V$ in $G^V$
5:     for each $n^S \in G^S$ do
6:       build node vector of $n^S$ according to (11)
7:     end for
8:     construct the status matrix M of $G^S$
9:   P = policyNetwork.getOutput(M)//Get the probability
     distribution from policy nework
10:  host = choose(P) //Choose a node as host for $n_t^V \in G^V$
11:  end for
12:  map virtual links via Algorithm3
13:  calculate Reward according to (17)
14:  for $t = 1, 2, \ldots, T$ do
15:    $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot Reward$ // Updating
       policy parameters using gradients, $\nabla_\theta$ means computing
       gradients
16:  end for
17: end for
18: return $\theta$
```

Algorithm 2 Mapping Virtual Nodes Using MLRL-Model

```
Input: $G^S$: substrate network
       $G^V$: VNR
Output: NM // the node mapping results of $G^V$
1: training MLRL-Model via Algorithm 1
2: $NM \leftarrow \emptyset$
3: for each $n^V \in G^V$
4:   for each $n^S \in G^S$
5:     update the node vector of $n^s$
6:   end for
7:   construct the status matrix M of $G^S$
8:   P = policyNetwork.getOutput (M)
9:     map $n^V$ to the substrate node $n^S$ with maxProb P
   //choose the node with the highest probability as the host
10:      $NM \leftarrow NM + \{n^V \rightarrow n^S\}$
11: end for
12: return NM
```

B. ML-VNE Algorithm

1) VI-VNE: The Sub-Algorithm for Mapping Under VNRs: When mapping an under VNR, MLRL-Model is trained based on the current resource status of the physical network and the VNR to get the optimal node mapping policy. Then, this policy is used to map all virtual nodes of this VNR in the node mapping stage.

Algorithm 2 shows the details of the process above.

After finishing the node mapping stage, the K shortest paths (KSP) algorithm is employed to map virtual links to the shortest physical paths that have enough bandwidth resources between fixed nodes. In this paper, we set K to 1. Algorithm 3 describes the implementation details.

Therefore, VI-VNE for mapping under VNRs is composed of Algorithm 2 and Algorithm 3.

Algorithm 3 Mapping Virtual Links Using KSP

```
Input: $G^S$, substrate network
       $G^V$, VNR
       NM, the node mapping results of $G^V$
Output: LM, the link mapping results of $G^V$
1: $LM \leftarrow \emptyset$
2: for each virtual link $l^V$ in $G^V$ do
3:   $G^S_{tmp} \leftarrow G^S$
4:   for each substrate link $l^S$ in $G^S$ do
5:     if $b(l^S) < b(l^V)$ then
6:       cut $l^S$ in $G^S_{tmp}$
7:     end if
8:   end for
9:   get two endpoints of $l^V$ : $n_1^V$ and $n_2^V$
10:   get the host $n_1^S$ of $n_1^V$ and the host $n_2^S$ of $n_2^S$ from NM
11:   get K shortest paths as U between $n_1^S$ and $n_2^S$ in $G^S_{tmp}$
12:   for each path $u \in U$ do
13:     if each link on path u satisfies $l^V$ then
14:       $LM \leftarrow LM + \{l^V \rightarrow u\}$
15:       break
16:     end if
17:   end for
18: end for
19: return LM
```

2) VV-VNE: The Sub-Algorithm for Mapping Upper VNRs: Different from VI-VNE, mapping upper VNRs is more flexible. Because the substrate network of upper VNR is logical, it can be expanded or re-constructed by submitting its new resource applications. Therefore, when an accepted VNR fails to map its upper VNRs, it can adjust its mapping results to accept them. In order to make under VNRs accept upper VNRs as much as possible and minimize the instability of the network virtualization environment, we propose a dynamic mapping mechanism that can coordinate the resources of physical network and under VNRs. It gives priority to increasing the resource capacity of particular under virtual nodes/links or migrating them to other physical nodes/links with remapping the entire network as the complement operation. As shown in Fig. 2, this mechanism will be triggered whether failure occurs during the node or link mapping stage.

The node mapping stage of VV-VNE is similar to VI-VNE, but MLRL-Model is trained with the data on the under VNR, not the physical network any more. If all the virtual nodes are mapped successfully, the algorithm will try to map virtual links. Otherwise, the under VNR will be adjusted as following situations.

(1) For the under virtual node n_S^V that is chosen by the upper virtual node n^V, update its requirements according to (18).

$$\begin{cases} c_{new}(n_S^V) = c_{old}(n_S^V) + c(n^V) - c_a(n_S^V), \\ \quad \text{if } c_{old}(n^V) > c_a(n_S^V) \\ f_{new}(n_S^V) = f_{old}(n_S^V) + f(n^V) - f_a(n_S^V), \\ \quad \text{if } f_{old}(n^V) > f_a(n_S^V) \\ q_{new}(n_S^V) = q_{old}(n_S^V) + q(n^V) - q_a(n_S^V), \\ \quad \text{if} q_{old}(n^V) > q_a(n_S^V) \end{cases} \tag{18}$$

where $c(n^V)$ is the CPU requirement of n^V; $c_{old}(n_S^V)$ and $c_{new}(n_S^V)$ are respectively the original and updated CPU

Algorithm 4 Virtual Node Mapping Stage of VV-VNE

```
Input: $G_S^V$, under VNR
       $G^V$, upper VNR
Output: $NM_{vv}$, the node mapping results of $G^V$
1: $NM_{vv} \leftarrow \emptyset$
2: for each node $n^V$ in $G^V$ do
3:   map virtual nodes via Algorithm 2
4:   if fail to map $n^V$ onto the $n_S^V$ of $G_S^V$ then
5:     update the requirements of $n_S^V$ according to (18)
6:     get $n^S$, the host physical node of $n_S^V$
7:     if the available resource of $n^S$is larger than $n_S^V$then
8:       allocate more physical resource for $n_S^V$ to satisfy its new
         resource requirement
9:     else
10:      remap $n_S^V$ onto another physical node which is
         within 5 hops from $n^S$ and remap all its adjacent links
11:      if success then
12:        map $n^V$ to the newly mapped $n_S^V$
13:      else:
14:        remap $G_S^V$ via Algorithm 2 & 3
15:        if success then
16:          Back to Step2
17:        else:
18:          return failure
19:        end if
20:      end if
21:    end if
22:  end if
23:  $NM_{vv} \leftarrow NM_{vv} + \{(n^V \rightarrow n_S^V)\}$
24: end for
25: return $NM_{vv}$
```

requirement of n_S^V; $c_a(n_S^V)$ is the residual CPU resource before updating its requirement; symbol f represents flow entry space and q represents queue storage.

(2) If the host physical node of n_S^V can meet its new resource requirements, it will allocate corresponding resources to n_S^V and then n^V can be accepted by n_S^V.

(3) Otherwise, following steps will be taken: Migrate n_S^V to another physical node that has sufficient resources and is within 5 hops from its original host physical node, and then remap all its adjacent links. If fails to find a replaceable physical node for n_S^V, the entire under VNR that n_S^V belongs to will be remapped using Algorithm 2 and Algorithm 3.

Algorithm 4 shows the implementation details.

In the link mapping stage, VV-VNE will also map virtual links successively using KSP algorithm, where K = 1. If all the links are successfully mapped, the algorithm will try to map another upper VNR. Otherwise, the dynamic adjustment mechanism of the corresponding under VNR is triggered. The detailed adjustment process is as follows.

(1) For an upper virtual link l^V to be mapped, select an under virtual path randomly and update the bandwidth requirement of each link in this under virtual path according to (19).

$$b_{new}\left(l_S^V\right) = b_{old}\left(l_S^V\right) + b\left(l^V\right) - b_a\left(l_S^V\right) \tag{19}$$

where $b_{old}(l_S^V)$ and $b_{new}(l_S^V)$ respectively denotes original and updated bandwidth resource requirements of l_S^V; $b(l^V)$ is the bandwidth resource requirement of l^V; $b_a(l_S^V)$ is the residual bandwidth resource before updating its requirement.

Algorithm 5 Virtual Link Mapping of VV-VNE

```
Input: $G_S^V$, under VNR
       $G^V$, upper VNR
       $NM_{vv}$, the node mapping results of $G^V$
Output: $LM_{vv}$, the link mapping result of $G^V$
1: for each link $l^V$ in $G^V$ do
2:   map virtual links via Algorithm 3
3:   if fail to map $l^V$ onto $G_S^V$ then
4:     randomly choose a failed path $p_S^V$ in $G_S^V$
5:     for each link $l_S^V$ in $p_S^V$ do
6:       update the requirement of $l_s^V$ according to (19)
7:       get $l^S$, the host physical path of $l_S^V$
8:       if $b_a(l^S) \geq b(l_S^V)$ then
9:         allocate physical link resource for $l_S^V$ to satisfy its new
           resource requirement
10:      else
11:        remap $l_S^V$ using KSP (K = 5) algorithm
12:        if success then
13:          map $l^V$ onto the newly mapped $l_S^V$
14:        else
15:          remap $G_S^V$ via Algorithm 2 & 3
16:          if success then
17:            map $G^V$ onto $G_S^V$ via Algorithm 4 & 5
18:          else
19:            return failure
20:          end if
21:        end if
22:      end if
23:    end for
23:  end if
24:  $LM_{vv} \leftarrow LM_{vv} + \{l^V \rightarrow l_S^V\}$
25: end for
26: return $LM_{vv}$
```

(2) If the host physical path can meet the new resource requirements of this under virtual path, it will allocate corresponding bandwidth resources to the under virtual path.

(3) Otherwise, following steps will be taken: Migrate each link in this under path to another physical path that has sufficient bandwidth resources using KSP algorithm. In order to reduce the resource occupation and minimize the impact on original topology, K is set to 5 here. If all under virtual links are migrated successfully, the adjustment is over. Otherwise, the under VNR that these virtual links belong to will be remapped using Algorithm 2 and Algorithm 3.

Algorithm 5 shows the implementation details.

V. Evaluation

In order to evaluate the performance of our proposed ML-VNE algorithm, we conduct a number of simulation experiments in both the single-layer mapping scenario and the multi-layer mapping scenario.

A. Simulation Environment

We simulate the algorithms on two kinds of physical networks respectively with general topologies and hierarchical topologies. In the general networks, all nodes are randomly

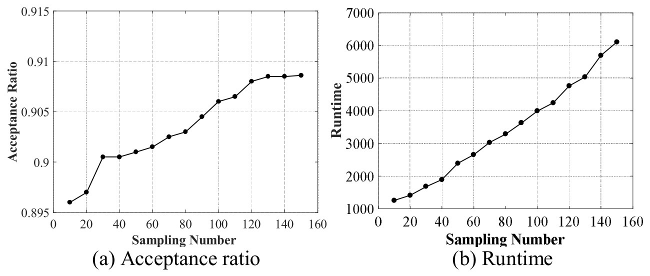


Fig. 4. Performance of ML-VNE adopting different sampling numbers.

TABLE I
PARAMETERS OF THE GENERAL PHYSICAL NETWORK

Parameter Names	Values
Node number	100
Link number	570
Initial CPU, flow entry space, and queue storage of nodes	50-100
Initial bandwidth resources of links	50-100

placed in a plane and connected with a certain probability. In the hierarchical networks, stub domains and transit domains are defined. A transit domain is composed of backbone nodes and the multiple stub domains are connect to them. We employ GT-ITM tool to generate the physical networks, and their detailed parameter settings are respectively summarized in Table I and Table II.

Similarly, we use GT-ITM tool to generate multiple VNRs. Table III summarizes the parameters of VNRs in the single-layer mapping scenario, while Table IV and Table V summarize the parameters of under VNRs and upper VNRs in the multi-layer mapping scenario.

B. Comparison Algorithms and Evaluation Metrics

The performance of ML-VNE is evaluated by comparing it with GRC-VNE [8], MCTS-VNE [9] and RL-VNE [10] in single-layer mapping scenario firstly and then in multi-layer scenario.

(1) *GRC-VNE:* This algorithm optimizes the virtual node mapping by considering resource and global topology attributes of nodes, but only uses Markov random walk algorithm to evaluate the fitness of mapping a virtual node to a certain physical node. The comparison between ML-VNE and GRC-VNE can reflect the effect of reinforcement learning on VNE.

(2) *MCTS-VNE:* Similar to ML-VNE, MCTS-VNE also need to try many mapping polices and then choose the optimal one to map a VNR. However, it adopts the method called Monte Carlo Tree Search instead of reinforcement learning.

(3) *RL-VNE:* This algorithm employs a policy network based on reinforcement learning to make node mapping decisions, but the optimization is achieved by training the policy network with the data of historical VNRs. However, the mapping policy in RL-VNE will never change after training, which is the essential difference from our algorithm.

Considering that the objectives of most VNE algorithms are to maximize the revenue and minimize the cost for accepting VNRs, we use the following five metrics to measure

TABLE II
PARAMETERS OF THE HIERARCHICAL PHYSICAL NETWORK

Parameter Names	Values
Node number	100
Link number	250
Initial CPU, flow entry space, and queue storage of transit nodes	150-200
Initial CPU, flow entry space, and queue storage of stub nodes	50-100
Initial bandwidth resources of transit-transit links	150-200
Initial bandwidth resources of transit-stub links	100-150
Initial bandwidth resources of stub-stub links	50-100

TABLE III
PARAMETERS OF VNRS IN THE SINGE-LAYER MAPPING SCENARIO

Parameter Names	Values
Number of VNRs	2000
Arrival rate	Follow a Poisson distribution with an average of 4 VNRs over 100 time units
Arrival time	0 to 50000 time units
Lifetime of each VNR	Follow an exponential distribution with an average of 1000 time units
Node number of each VNR	2-10
Link number of each VNR	$n(n-1)/4$, n is node number
CPU, flow space entry and queue storage demand of a virtual node	0-50
Bandwidth demand of a virtual link	0-50

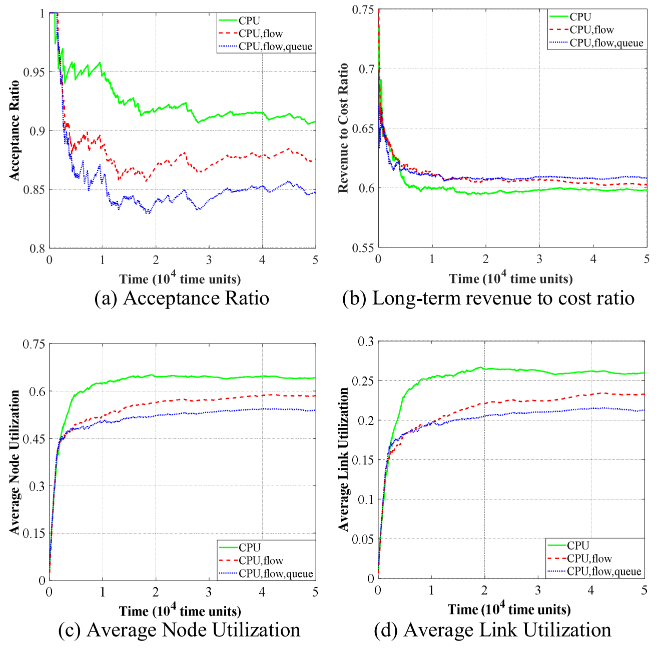
TABLE IV
PARAMETERS OF UNDER VNRS IN THE MULTI-LAYER MAPPING SCENARIO

Parameter Names	Values
Number of VNRs	400
Arrival rate	Follow a Poisson distribution with an average of 8 VNRs over 1000 time units
Arrival time	0 to 50000 time units
Lifetime of each VNR	Follow an exponential distribution with an average of 1000 time units
Node number of each VNR	10-20
Link number of each VNR	$n(n-1)/4$, n is node number
CPU, flow space entry and queue storage demand of a virtual node	25-50
Bandwidth demand of a virtual link	25-50
Number of upper requests for each under VNR	4

the performance of ML-VNE and the comparison algorithms: 1) VNR acceptance ratio; 2) Long-term revenue to cost ratio; 3) Average node utilization; 4) Average link utilization.

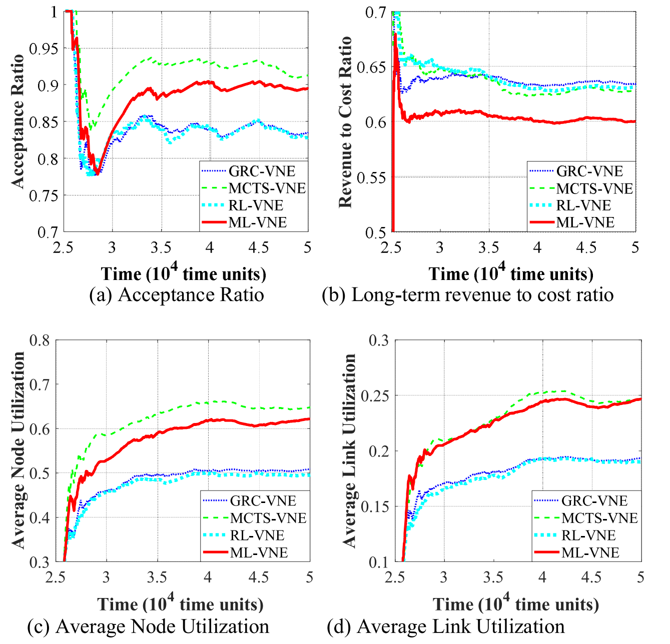
C. Simulations and Results Analysis

1) Determine Optimal Sampling Number in ML-VNE: As our algorithm is based on the Monte Carlo reinforcement learning method, a virtual network will try to be mapped for many times to get enough samples for the policy training. Therefore, before comparing with other algorithms, we firstly determine the optimal sampling number when training the MLRL-Model in ML-VNE. Fig. 4 shows the acceptance ratio and runtime of ML-VNE with different sampling numbers on a general physical network.



(a) Acceptance Ratio (b) Long-term revenue to cost ratio (c) Average Node Utilization (d) Average Link Utilization

Fig. 5. Performance of ML-VNE considering different resource granularities.



(a) Acceptance Ratio (b) Long-term revenue to cost ratio (c) Average Node Utilization (d) Average Link Utilization

Fig. 6. Performance comparison between ML-VNE and contrast algorithms on the general physical network.

It can be seen in Fig. 4 that, with the increase of sampling number, the acceptance ratio grows rapidly at the beginning and starts to slow down when the sampling number is 120. However, the runtime presents a linear growth trend all the time. This means when we further increase the sampling number, ML-VNE will achieve a limited performance enhancement but cost much more runtime. Therefore, we set the sampling number in MLRL-Model to 120.

In addition, we can see the main range of the acceptance ratio is about 0.9 to 0.91, which means ML-VNE is fairly stable and performs well under different learning circumstances.

TABLE V
PARAMETERS OF UPPER VNRS IN THE MULTI-LAYER MAPPING SCENARIO

Parameter Names	Values
Number of VNRs	1600
Arrival time	Subject to the lifetime of its under VNR
Lifetime of each VNR	Follow an exponential distribution with an average of 2000 time units
Node number of each VNR	2-9
Link number of each VNR	$n(n-1)/4$, n is node number
CPU, flow space entry and queue storage demand of a virtual node	0-25
Bandwidth demand of a virtual link	0-25

We are pleased to find out that the acceptance ratio converged nicely while the runtime grows linearly, which means the learning model can adapt itself to mapping samples and there's no need to set a bigger sampling number.

2) The Influence of Different Virtualized Resource Granularity on the Performance of ML-VNE: The resource granularity means how many kinds of network resources to be considered in VNE. In order to evaluate the impact of virtualized resource granularity on the overall performance of ML-VNE, we perform another set of simulation experiments. We expand ML-VNE by considering following different node resource granularities: 1) only CPU; 2) CPU and flow entry space; 3) CPU, flow entry space and queue storage.

As shown in Fig. 5, the acceptance ratio, average node utilization and average link utilization of ML-VNE will decrease when more node resource granularities are considered while the revenue to cost ratio maintains a steady level. The reason is that, as the resource granularity of virtualization increases, VNRs will subject to more resource constrains, so there will be fewer feasible mapping solutions, which leads to worse performance of ML-VNE. This shows that considering fine-grained virtualized network resources can provide better isolation between different VNRs, but it would make the solving process of VNE more difficult. Therefore, there should be a trade-off between ensuring the performance of VNE algorithm and the granularity of virtualized node resources.

3) Performance Comparisons Between Different Algorithms: Since GRC-VNE, MCTS-VNE and RL-VNE only consider CPU resource when solving VNE problem, in order to fairly compare their performance with that of ML-VNE, we conduct an experiment for the case that ML-VNE also only considers CPU resource. In addition, RL-VNE needs to divide the VNRs equally into two sets, the first half is used for training algorithm model, and the latter half is used for real mapping. In order to be consistent with RL-VNE, in this experiment, we also divide all VNRs into two parts, the former part is only used for the training process of RL-VNE, and the latter is used by all algorithms to evaluate their performance. We perform this experiment on the physical network with general topology and the physical network with hierarchical topology respectively.

a) Performance on general physical network: Fig. 6 shows the performance comparison between ML-VNE and the contrast algorithms on the general physical network.

The acceptance ratio metric measures the number of VNRs that could be completely embedded by the VNE algorithm. It can be seen from Fig. 6(a) that, ML-VNE achieves a high acceptance ratio only second to MCTS-VNE. This is because before truly mapping a VNR, both of ML-VNE and MCTS-VNE will try many times iteratively in order to acquire an optimal mapping policy. That is, they adjust their mapping policies all the time to make the physical network accept VNRs as much as possible. RL-VNE also obtains a higher acceptance ratio because it learns an optimal mapping policy from a set of VNRs in advance. However, RL-VNE cannot adjust its mapping policy after model learning, so its acceptance ratio is lower than ML-VNE and MCTS-VNE. The mapping policy of GRC-VNE is more rigid than RL-VNE because it relies on artificial rules to make mapping decisions, which means the parameters in GRC-VNE are always fixed and cannot be dynamically optimized, leading to a lower acceptance ratio.

Depending on the random topologies of VNRs, it is meaningless to compare different algorithms based on costs or revenues separately. Therefore, long term revenue to cost ratio is adopted to balance them. The higher the ratio, the less resources were needed to embed VNRs, which means the algorithm is more efficient. It can be seen from Fig. 6(b) that the ratios of all algorithms are close to each other. Because RL-VNE and GRC-VNE have a relatively lower acceptance ratio, costs and revenues generated are consequently less while leading to a relatively high ratio. As for ML-VNE, it will map some virtual links onto longer physical paths in order to increase the acceptance ratio, resulting higher costs. Consequently, the ratio of ML-VNE will not be much higher than other algorithms.

Average node and link utilization measure the resource utilization of the physical network. As can be seen from Fig. 6(c) and Fig. 6(d), ML-VNE and MCTS-VNE achieve much higher average node utilization and average link utilization than GRC-VNE and RL-VNE due to their high acceptance ratio. Combining Fig. 6(b) with Fig. 6(d), we can see that the average link resource utilization of ML-VNE is very close to that of MCTS-VNE, which means that ML-VNE allocate much more link resources in the link mapping phase. Therefore, ML-VNE has a lower long-term revenue to cost ratio than MCTS-VNE.

b) Performance on hierarchical physical network: Fig. 7(a) shows that the acceptance ratio of each algorithm has a similar change tendency compared with that on the general physical network in Fig. 6(a), but performance degradation occurs. This is because the differences between nodes is larger and the quantity of links is small in hierarchical physical network, making it harder to get feasible mapping solutions. However, ML-VNE and MCTS-VNE still achieve better performance than RL-VNE and GRC-VNE due to their characteristic of adjusting mapping policies in real time as shown in Fig. 7(a), (c) and (d). As can be seen from Fig. 7(b), ML-VNE achieves a much higher revenue to cost ratio than MCTS-VNE because the topological attributes of nodes are more obvious in the hierarchical network, ML-VNE can better learn about the relationships between these attributes.

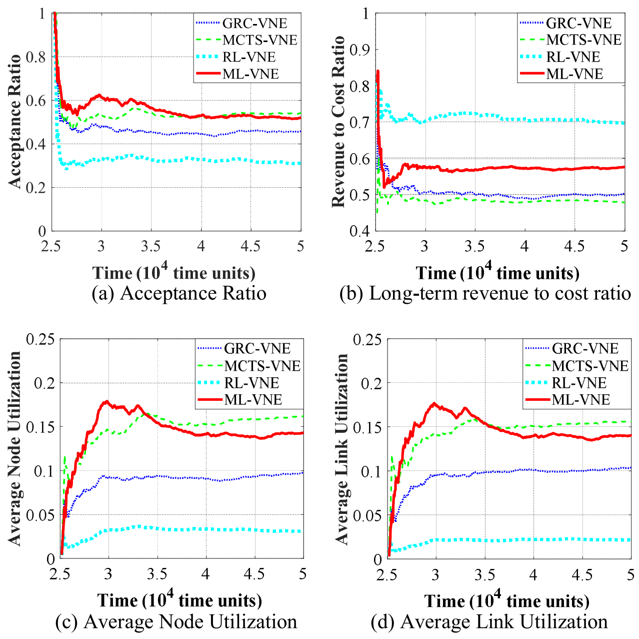


Fig. 7. Performance comparison between ML-VNE and contrast algorithms on the hierarchical physical network.

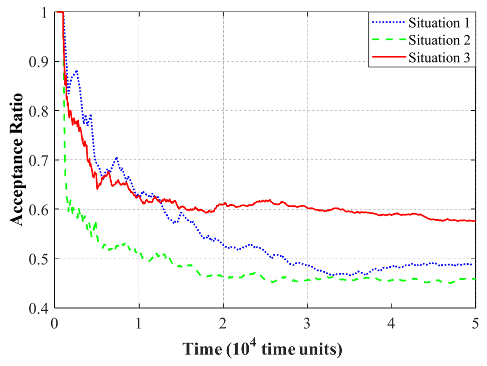


Fig. 8. Acceptance ratio of ML-VNE in different situations in multi-layer mapping scenario.

4) Performance of ML-VNE in Multi-Layer Mapping Scenario: Although a multi-layer VNE algorithm was proposed in [7], it is not suitable for a comparison scheme in multi-layer mapping scenario for two reasons: firstly, it supposes virtual nodes are pre-fixed and cannot be mapped arbitrarily; secondly, it always maps upper VNRs successfully since under VNRs are completely constructed based on them. Therefore, we conduct following experiments in three different situations to evaluate the performance of ML-VNE:

(1) *Situation 1:* only map under VNRs.

(2) *Situation 2:* map under and upper VNRs without dynamic adjustment mechanism.

(3) *Situation 3:* map under and upper VNRs with dynamic adjustment mechanism.

Fig. 8 shows the acceptance ratio of ML-VNE in different situations in multi-layer mapping scenario. Compared with the first situation, the acceptance ratio in the second situation is significantly lower. It is because more VNRs in the second situation need to be mapped, leading more upper VNRs may

TABLE VI
AVERAGE RUNNING TIME FOR MAPPING A VNR

Algorithm Names	Values (unit: seconds)
ML-VNE	0.833
GRC-VNE	0.438
MCTS-VNE	0.595
RL-VNE	0.642

be rejected. However, as the dynamic adjustment mechanism is enabled in the third situation, the upper VNRs that fail to be mapped can be accepted by adjusting the mapping results of corresponding under VNRs. Therefore, the total acceptance ratio of ML-VNE increases a lot in the third situation.

5) Time Cost: All of our experiments are running on the same computer with 3.70 GHz Intel Core i5-9600 CPU and 2.00 GB RAM. In order to evaluate the time complexity, the average running time for mapping a VNR using different algorithms is regarded as an appropriate indicator. We calculate the average running time in the first scenario of the third experiment above to reduce repetitive experiments and list them as follows.

In Table IV, ML-VNE cost more time than GRC-VNE and RL-VNE, while less than MCTS-VNE. As we mentioned above, GRC-VNE and RL-VNE only need to calculate once using the prepared mapping policy before mapping an arrived VNR without any trials. That explains why ML-VNE and MCTS-VNE cost more time. Moreover, because ML-VNE is based on the Monte Carlo reinforcement learning method, which take advantage both of Monte Carlo method and reinforcement learning method, makes the optimization of mapping policies faster than MCTS-VNE.

To sum up, ML-VNE not only achieves a relatively good performance in single-layer mapping scenario, but also ensures its performance in multi-layer mapping scenario. Therefore, MLRL-Model and the dynamic mapping mechanism both play an effective role in our proposed ML-VNE algorithm.

VI. CONCLUSION

ML-VNE, a dynamic and collaborative multi-layer VNE algorithm, is proposed in this paper for solving the mapping between multi-layer VNRs that arrive dynamically and the physical network. It not only realizes the real-time update of node mapping policy through MLRL-model, but also improves the acceptance ratio of upper VNRs by coordinating the resources of the physical network and under VNRs.

Several possible research directions can be pursued from our current work. Firstly, some excellent deep learning models can be introduced to enhance the MLRL-Model. Secondly, a virtual link is only allowed to be mapped onto a single substrate path in this paper, so path splitting can be considered to improve the success ratio of mapping virtual links.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] T. Anderson, L. Peterson, S. Shenker, and J. Turner, “Overcoming the Internet impasse through virtualization,” *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[3] N. Feamster, L. Gao, and J. Rexford, “How to lease the Internet in your spare time,” *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 61–64, Jan. 2007.

[4] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, “Virtual network embedding: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.

[5] J. Zhang *et al.*, “Dynamic virtual network embedding over multilayer optical networks,” *IEEE/OSA J. Opt. Commun. Netw.*, vol. 7, no. 9, pp. 918–927, Sep. 2015.

[6] S. R. Chowdhury *et al.*, “Multi-layer virtual network embedding,” *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 3, pp. 1132–1145, Sep. 2018.

[7] H. T. Nguyen *et al.*, “A generalized resource allocation framework in support of multi-layer virtual network embedding based on SDN,” *Comput. Netw.*, vol. 92, no. 2, pp. 251–269, Dec. 2015.

[8] L. Gong, Y. Wen, Z. Zhu, and T. Lee, “Toward profit-seeking virtual network embedding algorithm via global resource capacity,” in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Toronto, ON, Canada, 2014, pp. 1–9.

[9] S. Haeri and L. Trajkoviæ, “Virtual network embedding via Monte–Carlo tree search,” *IEEE Trans. Cybern.*, vol. 48, no. 2, pp. 510–521, Feb. 2018.

[10] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, “A novel reinforcement learning algorithm for virtual network embedding,” *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.

[11] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha, “Optimal virtual network embedding: Node-link formulation,” *IEEE Trans. Netw. Service Manag.*, vol. 10, no. 4, pp. 356–368, Dec. 2013.

[12] M. Chowdhury, M. R. Rahman, and R. Boutaba, “ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping,” *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[13] M. Yu, Y. Yi, J. Rexford, and M. Chiang, “Rethinking virtual network embedding: Substrate support for path splitting and migration,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Apr. 2008.

[14] X. Cheng *et al.*, “Virtual network embedding through topology-aware node ranking,” *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 38–47, Apr. 2011.

[15] M. Feng, J. Liao, J. Wang, S. Qing, and Q. Qi, “Topology-aware virtual network embedding based on multiple characteristics,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, 2014, pp. 2956–2962.

[16] Z. Zhang, X. Cheng, S. Su, Y. Wang, K. Shuang, and Y. Luo, “A unified enhanced particle swarm optimization-based virtual network embedding algorithm,” *Int. J. Commun. Syst.*, vol. 26, no. 8, pp. 1054–1073, Aug. 2013.

[17] K. Guo, Y. Wang, X. Qiu, W. Li, and A. Xiao, “Particle swarm optimization based multi-domain virtual network embedding,” in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, Ottawa, ON, Canada, 2015, pp. 798–801.

[18] C. Wang, Y. Su, L. Zhou, S. Peng, Y. Yuan, and H. Huang, “A virtual network embedding algorithm based on hybrid particle swarm optimization,” in *Proc. Int. Conf. Smart Comput. Commun. (SmartCom)*, Shenzhen, China, 2016, pp. 568–576.

[19] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, “VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kyoto, Japan, 2011, pp. 1–6.

[20] X. Guan, X. Wan, B. Choi, and S. Song, “Ant colony optimization based energy efficient virtual network embedding,” in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Niagara Falls, ON, Canada, 2015, pp. 273–278.

[21] J. Wang, W. Chen, H. Cong, Z. Zhan, and J. Zhang, “An ant colony system based virtual network embedding algorithm,” in *Proc. IEEE Int. Conf. Syst. Man Cybern. (SMC)*, Banff, AB, Canada, 2017, pp. 1805–1810.

[22] X. Liu, Z. Zhang, X. Li, and S. Su, “Optimal virtual network embedding based on artificial bee colony,” *EURASIP J. Wireless Commun. Netw.*, vol. 2016, no. 1, pp. 273–281, Aug. 2016.

[23] Z. Qiang, W. Qiang, F. Sheng, and L. Wu, “Heuristic survivable virtual network embedding based on node migration and link remapping,” in *Proc. IEEE 7th Joint Int. Inf. Technol. Artif. Intell. Conf. (ITAIC)*, Chongqing, China, 2014, pp. 181–185.

[24] M. Demirci and M. Ammar, “Design and analysis of techniques for mapping virtual networks to software-defined network substrates,” *Comput. Commun.*, vol. 45, no. 1, pp. 1–10, Jun. 2014.

[25] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. D. Turck, and S. Latré, "Dynamic resource management in SDN-based virtualized networks," in *Proc. 10th Int. Conf. Netw. Service Manag. (CNSM) Workshop*, Rio de Janeiro, Brazil, 2014, pp. 412–417.

Meilian Lu (Member, IEEE) received the Ph.D. degree in communication and information system from the Beijing University of Posts and Telecommunications, China, in 2012, where she is currently a Professor working with the State Key Laboratory of Networking and Switching Technology. Her main research interests include machine learning, social network analysis, and future network architecture.

Yun Gu received the M.S. degree in computer science and technology from the Beijing University of Posts and Telecommunications, China, in 2019. His main research interests are network virtualization and machine learning.

Dongliang Xie (Member, IEEE) received the Ph.D. degree from the Beijing Institute of Technology, China, in 2002. He is a Full Professor and the Director of Broadband Network Research Center, State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China, where he was a Postdoctoral Research Fellow and a Visiting Professor with the State University of New York, Stony Brook. He has been conducting research work in the design of network architectures, protocols and algorithms. His research interests have expanded from wireless sensor network to the future information centric network, mobile social network, and mobile cloud computing. He was the poster Co-Chair of ACM/IEEE IWQoS in 2014 and local arrangement Chair of ACM/IEEE IWQoS in 2016. He has been serving technical program committee of conferences in networking and computing areas, including IEEE Infocom 2015 and 2016, and IEEE GlobeCom 2016.