DeepMigration: Flow Migration for NFV with Graph-based Deep Reinforcement Learning

Penghao Sun[1], Julong Lan[1], Zehua Guo[2], Di Zhang[3], Xianfu Chen[4], Yuxiang Hu[1], Zhi Liu[5]
[1]National Digital Switching System Engineering & Technological R&D Center [2]Beijing Institute of Technology
[3]Zhengzhou University [4]VTT Technical Research Centre of Finland [5]Shizuoka University

***Abstract*—Network Function Virtualization (NFV) enables flexible deployment of network services as applications. Network operators expect to use a limited number of Network Function (NF) instances to handle the fluctuating traffic load and provide network services. However, it is a big challenge to guarantee the Quality of Service (QoS) under the unpredictable network traffic while minimizing the processing resources. One typical solution is to realize NF scale-out, scale-in and load balancing by elastically migrating the related traffic flows with Software-Defined Networking (SDN). However, it is difficult to optimally migrate flows since many real-time statuses of NF instances should be considered to make accurate decisions. In this paper, we propose *DeepMigration* to solve the problem by efficiently and dynamically migrating traffic flows among different NF instances. *DeepMigration* is a *Deep Reinforcement Learning (DRL)*-based solution coupled with *Graph Neural Network (GNN)*. By taking advantages of the graph-based relationship deduction ability from our customized GNN and the self-evolution ability from the experience training of DRL, DeepMigration can accurately model the cost (e.g., migration latency) and the benefit (e.g., reducing the number of NF instances) of flow migration among different NF instances and generate dynamic and effective flow migration policies to improve the QoS. Experiment results show that DeepMigration requires less migration cost and saves up to 71.6% of the computation time than existing solutions.**

***Index Terms*—Network Function Virtualization, Flow Migration, Deep Reinforcement Learning, Graph Neural Network**

I. Introduction

Network Function Virtualization (NFV) has been proposed to virtualize Network Functions (NFs) from dedicated hardware middleboxes (e.g., intrusion detection system, firewall, and load balancer) to general-purpose hardware (e.g., x86 servers). Thus, network operators can save their Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) [1], [2] and improve their service quality by efficiently deploying new functions and providing flexible, agile services [3]–[5]. One critical benefit of NFV is elastic control, which flexibly deploys NF instances to accommodate the varying traffic [6], [7]. NFV elastic control relies on Software-Defined Networking (SDN) [8] to dynamically steer the network traffic. In this case, the network can accommodate the varying network traffic with NF scaling and load balancing [9], [10]. During the traffic steering process, the processing flows should be migrated among different NF instances. Existing works [9], [11]–[13] consider the scalability and efficiency of the data plane for flow migration. However, during the flow migration, these works may degrade the Quality of Service (QoS) of migrated flows. For example, the buffer of the destination NF instance may be overflowed. OFM [10] takes into account the impact of migration on the QoS and formulates the migration problem as an Integer Linear Programming (ILP) problem. However, its proposed solution is heuristic-based and cannot realize a good performance concerning the migration cost under different migration scenarios.

To ensure both the efficiency of the migration policy and the QoS of the migrated flows, in this paper, we propose DeepMigration to address the flow migration problem with machine learning. To better process the graph-structured workload in the network topology, DeepMigration models the flow migration problem as a graph input for the Graph Neural Network (GNN). Then, DeepMigration employs the self-evolution ability of Deep Reinforcement Learning (DRL) to train the GNN. In this way, we can combine the advantage of the graph-based relationship deduction ability of GNN and the self-evolution ability of DRL to dynamically generate near-optimal flow migration policies. To the best of our knowledge, our work first proposes to address the flow migration problem for elastic control in NFV by combining GNN and DRL. The contributions of this paper are summarized as follows:

- We design a scalable GNN-based neural network structure that directly takes the topological information of the network as the input to flexibly handle the flow distribution.
- We propose a DRL-based training framework to automatically train the GNN and improve the flow migration policy.
- We implement DeepMigration on a POX controller and conduct experiments to validate the effectiveness of DeepMigration on a hardware environment. The experiment results show that DeepMigration requires less migration cost and saves up to 71.6% of the computation time compared to existing solutions.

II. Background and Related Works

A. Flow Migration in NFV

NFV aims to replace dedicated and specialized middleboxes with virtualized NFs. Under the fluctuating network traffic, the workload distribution among a group of NF instances could change, and the flows are typically migrated among different NF instances under the following three scenarios:

1) Scale-out: When the workload of some NF instances

This paper is supported by the National Key Research and Development Plan under Grant Number 2017YFB0803204, the National Natural Science Fund of China under Grant Numbers 61521003 and 61872382, and the Beijing Institute of Technology Research Fund Program for Young Scholars. The corresponding author is Zehua Guo.

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

exceeds a certain threshold, such instances cannot efficiently handle their received traffic flows. In this case, new NF instances can be created, and some flows need to be migrated to the newly created instances.

2) Scale-in: When one or more NF instances are underloaded, some NF instances can be shut down for better OPEX and the flows on these instances should be migrated to the active instances.

3) Load balancing: When there are hot spots and cold spots among current NF instances, some flows need to be migrated from hot spots to cold spots for better QoS.

Flow migration can bring benefits such as an improvement of the QoS and a reduction of the OPEX, but it also comes with some cost. Assume migrating a flow from a source instance nf_s to a destination instance nf_d. For nf_s, a flow migration can reduce the queueing delay in this instance. For nf_d, a migrated flow needs to be temporarily stored in the buffer of the instance before processing, which increases the queue of the buffer of nf_d. In addition, the migration operation itself also incurs a latency (migration cost). The optimal flow migration policy, on the other hand, needs to be designed to maximize the overall benefit while reducing the overall migration cost.

B. Related works

Existing studies in [9]–[11], [14] have shown the importance of state migration in the elastic control of NFV. OpenNF [9] concentrates on the safety and efficiency of flow migration and E2 [15] proposes a strategy named migration avoidance to realize elastic control. These two works omit the migration cost and cannot guarantee the QoS of existing flows. Lin et al. [16] propose to transfer all the states in the NF instances for flow migration efficiency and scalability. However, it does not consider the fine-grained migration cost. Sun et al. [10] model the flow migration problem as an ILP problem and propose a heuristic method to solve this problem with reduced migration cost and low computation complexity. However, the heuristic-based method cannot ensure near-optimal migration costs under different migration scenarios.

III. Flow Migration Problem Formulation

In this section, we formulate the flow migration problem.

Suppose that we have N instances of the same type and the workload on instance j ($j \in [1, N]$) is l_j. We use Th^{high}, Th^{low}, and Th^{var} to denote the threshold of peak load, bottom load, and the variance of traffic load, respectively. Then, the three scenarios scale-out, scale-in and load balancing can be formulated as $\sum_{j=1}^{N} l_j \geq Th^{high}$, $\sum_{j=1}^{N} l_j \leq Th^{low}$, and $\mathrm{var}(l_1, l_2, ..., l_N) \geq Th^{\mathrm{var}}$, where $\mathrm{var}(\cdot)$ denotes the variance.

We use the Service Level Agreement (SLA) to evaluate the performance of the flow migration policy [17], [18]. Specifically, we use latency as the main SLA metric. Considering that there are m flows, the processing latency of flow f in NF instance i is $la\_p_i^f$, the queueing delay in NF instance i is $la\_q_i^f$, the migration latency from NF instance i to instance j is la_{ij}^f, and the SLA latency constraint for this flow is LA^f. In order to ensure the SLA, the migration operation of f should satisfy:

$$la_{ij}^f \leq LA^f - la\_p_j^f - la\_q_j^f, \forall f \in [1, m] \tag{1}$$

where $la\_p_j^f$ is a constant and $la\_q_i^f$ is influenced by the queue length in the buffer.

In this case, we can formulate the migration cost of f as follows:

$$C_{ij}^f = \begin{cases} \beta(la_{ij}^f + la\_p_j^f + la\_q_j^f - la\_q_i^f), \text{ (1) holds} \\ \delta + \beta(la_{ij}^f + la\_p_j^f + la\_q_j^f - LA^f), \text{ else} \end{cases} \tag{2}$$

where δ is a basic penalty for violating of SLA and β is a penalty weight for the exceeded value of the SLA.

In NFV elastic control, the number of NF instances may change due to the scale-out and scale-in operation. We thus use N_e to denote the number of NF instances before the migration, and N_n the number of NF instances after the migration. In addition, we assume that adding/deleting an NF instance brings a constant resource cost/benefit, which is denoted as R. We use x_{ij}^f as a binary variable to indicate whether flow f is migrated from instance i to instance j, $size_f$ as the size of flow f, $\Delta N = N_n - N_e$, $C_R = R \times \Delta N$ and Buf_j as the buffer size of instance j.

For the scale-out/scale-in scenario, as the number of NF instances changes, the resources used on the hardware also change. This change of the resource cost can be evaluated with $C_R = R \times \Delta N$. When N_n equals N_e, C_R is a constant. Then, the optimization problem of flow migration in any of the three scenarios can be formulated as:

$$\min_x \sum_{i=1}^{N_e} \sum_{j=1}^{N_n} \sum_{f=1}^{m} x_{ij}^f C_{ij}^f \tag{3}$$

$$\text{s.t.} \sum_{i=1}^{N_e} \sum_{j=1}^{N_n} x_{ij}^f = 1, \ \forall f \in [1, m] \tag{4}$$

$$\sum_{i=1}^{N_e} \sum_{f=1}^{m} (x_{ij}^f * size_f) \leq Buf_j, \ \forall j \in [1, N_n] \tag{5}$$

$$x_{ij}^f \in \{0, 1\}, \forall i \in [1, N_e], \forall j \in [1, N_n], \forall f \in [1, m] \tag{6}$$

Constraint (4) ensures that each flow is assigned to only one NF instance, constraint (5) ensures that the flow size on each NF instance should not exceed the buffer size and constraint (6) sets the range of value of each variable. The objective function of flow migration defined above is an Integer Problem (IP) and piecewise due to C_{ij}^f. Existing solutions cannot solve this problem within an acceptable time [10]. In this paper, we propose an online algorithm based on DRL and GNN to generate flow migration policies without human experience.

IV. Overview of DeepMigration

In our scheme, DeepMigration is established based on a controller of SDN, as shown in Fig. 1. DeepMigration mainly includes three functional modules: State Collection module, DRL Framework module, and Policy Translator module.

In DeepMigration, the network status and flow migration performance metric are collected periodically by the State

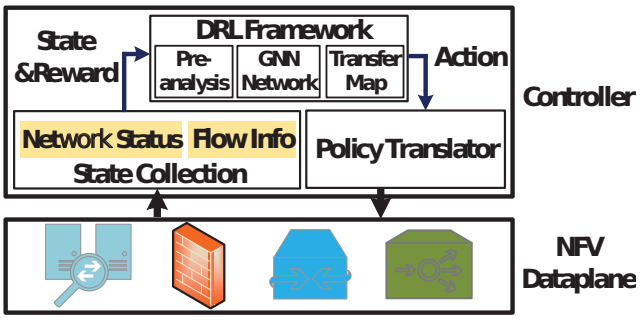

Fig. 1: Architecture of DeepMigration

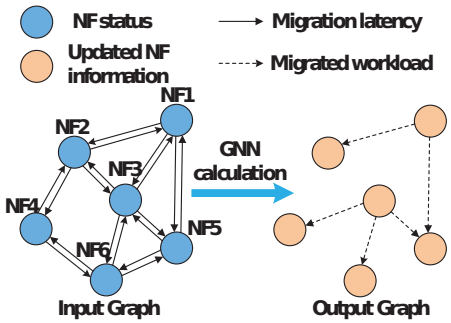

Fig. 2: A calculation example of GNN

Collection module and taken as the input to a DRL module, which uses GNN to perform the induction of the relationship between NFs and the distribution of the traffic flows. After a certain calculation process, the GNN produces a graph with different properties (Section V.B), where the node values denote the traffic load distribution on NF instances and the values of directed edges between two nodes denote the migrated traffic load between two NF instances. The traffic migration information is sent to the Policy Translator module, where the traffic load migration information is translated to concrete flow migration operation. When the flows are migrated, the controller collects the performance metric of the migration action as a policy reward to update the parameters of the GNN for neural network evolution (Section V.A). An example of the policy generation process is shown in Fig. 2. The GNN calculates the original workload distribution in the input graph and generates an output graph. In the output graph, the edge values denote the migration workload.

DeepMigration gets trained offline. After training, the DRL agent of DeepMigration will learn enough knowledge and then can be formally deployed in the network for flow migration.

V. Design Details of DeepMigration

In this section, we introduce the detailed design and working mechanism of DeepMigration.

A. Training Framework

DeepMigration uses a DRL framework to generate dynamic flow migration policies. A DRL framework utilizes neural networks as the function approximators of Reinforcement Learning (RL), of which the training process in the network environment is modeled as a Markov Decision Process (MDP) [19] [20] [21]. In the MDP, state s is the network status which is the input of the neural network; a is the action, which corresponds to the output of the neural network and is used for the generation of flow migration policy; r is the reward of a flow migration operation, which is used to evaluate the performance of the DRL policy and adjust the neural network parameters. An action a is chosen under state s according to a policy $\mu(s)$ (i.e., the function of the neural networks), so $a = \mu(s)$ as illustrated in [22] and the policy can be denoted with a probability function $\mu_\theta(s, a)$, where θ denotes the parameters in the neural networks.

The training process of DeepMigration takes place in episodes, and each episode consists of multiple action steps. The overall aim of the agent is to maximize the cumulated reward $R = \sum_{t=0}^{T} r(t)/T$ in each episode, where T denotes the total number of steps in an episode. After each episode, the DRL agent evaluates the performance of the policy with the cumulated reward R. Then, in DeepMigration, we use policy gradient methods to polish the neural networks in the DRL agent. In policy gradient methods, the gradient descent is used for the evolution of the neural network parameters θ following $\theta' = \theta + \alpha \sum_{t=1}^{T} \nabla_\theta \log \mu_\theta(s_t, a_t) Q_t$ [23], where α denotes a learning rate that controls the evolution speed of the θ during each episode, and Q_t denotes the quality evaluation of the policy in the current episode. In DeepMigration, we use $Q_t = \sum_{t'=t}^{T} r_{t'} - b_t$ to calculate Q_t, where t denotes the time and b_t is used as a baseline value to limit the variance of the policy gradient [24]. In this way, the neural networks are corrected towards the gradient direction based on the intuition that the action $a = \mu(s)$ with a larger average reward should get a better chance to be selected under state s. At the early stage of the training, since the performance of the policy is not good, the DRL agent should do more exploration than exploitation. Therefore, we use short episode lengths for the initial episodes, and gradually increase the episode length [25].

B. GNN-based Function Approximator

As mentioned in the aforementioned section, DRL uses a set of neural networks as the function approximator. The role of the function approximator in DRL is to analyze the characteristics of the input data, and calculate an abstract information as the output action. However, the resource and traffic load distribution are graph-like structured data. Information processing of the graph-like structured data is hard for a series of prevalent neural networks such as Convolutional Neural Network (CNN) [26] since these neural networks can only operate on regular Euclidean data (e.g., 2D image data and 1D text sequence) [27]. Therefore, we use GNN in DeepMigration to process the graph-like structured network status data.

Unlike other types of neural networks, GNN directly operates on graph data. There are various versions of models in the GNN family, and in DeepMigration we use one of the most popular version Graph Network (GN) [28]. The GN framework uses a directed graph named GN block for graph message processing. A GN block G is defined with three attributes $G = (V, E, u)$, where V denotes the node attributes, E denotes the edge attributes and u denotes a global attribute that describes an overall feature of the graph. Each of the

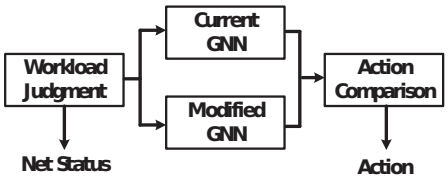

Fig. 3: Double GNN for NF variation in the DRL module

attributes can be expressed with an attribute vector that defines a series of elements considered in that attribute. GN uses three update functions denoted with ϕ (i.e., ϕ^e, ϕ^v and ϕ^u for edge, node and global attributes, respectively) and three aggregation functions denoted with ρ (i.e., $\rho^{e\to v}$,$\rho^{e\to u}$ and $\rho^{v\to u}$) to carry out the information computation for the edge, node and global attributes.

The computation contains six consecutive operations and we use an apostrophe ($'$) to denote the value of next time step:

1) $e'_k = \phi^e(e_k, v_{r_k}, v_{s_k}, u)$ is calculated per edge, which reflects a type of edge information, where r_k/s_k denote the receiving/sending node of edge k, respectively.
2) $\overline{e'_i} = \rho^{e\to v}(E'_i)$ aggregates the edge information and binds such information with the node information, where $E'_i = \{(e'_k, r_k, s_k)\}_{r_k=i,k=1:|E|}$.
3) $v'_i = \phi^v(\overline{e'_i}, v_i, u)$ is carried out per node, which calculates a type of node-based information.
4) $\overline{e}' = \rho^{e\to u}(E')$ aggregates all the edge information, where $E' = \cup_i E'_i$.
5) $\overline{v}' = \rho^{v\to u}(V')$ aggregates all the node information, where $V' = \{v'_i\}_{i=1:|V|}$.
6) $u' = \phi^u(\overline{e'}, \overline{v}', u)$ updates the global attribute.

The intersected computation of edge and node related information in the aforementioned six operations is a reflection of the induction process of the GNN. The edge attributes denote the migration latency and influence the traffic load redistribution among NF instances. The node attributes denote the traffic load distribution and NF status, which in return influence the amount of traffic load to be migrated through edges. Besides, the global attributes also reflect an overall traffic load level on the NF instances and can be used to decide whether a scale-in or scale-out operation should be conducted. In DeepMigration, the six functions are implemented with six different neural networks, among which we use one simple Recurrent Neural Network (RNN) for each of ϕ (i.e., ϕ^e, ϕ^v and ϕ^u), and use one two-layer Feedforward Neural Network (FNN) for each of ρ (i.e., $\rho^{e\to v}$,$\rho^{e\to u}$ and $\rho^{v\to u}$). All of the neural network parameters can be denoted as θ as defined in Section V.A and updated by the training process of DRL.

For the scenario of scale-in and scale-out that introduces a variation of the NF instance, the DRL module employs a double GNN structure, as shown in Fig. 3. The traffic load Judgment submodule compares the overall traffic load with a threshold to decide the number of active NF instances. Then, the network status is sent to two GNNs: one with the current graph topology and the other one with a modified graph topology. The modified GNN modifies the structure of the current GNN by adding a vertex connected to the currently highest loaded NF instance for the scale-out scenario and deleting a vertex of the least loaded NF instance for the scale-in scenario. Then, the action generated by the two GNNs will be sent to an Action Comparison submodule, which compares the cost of two migration actions and chooses the one with lower cost as the final action of the DRL module.

Algorithm 1 Training process of the DRL agent

Input: network status and flow information;
Output: traffic load migration;
1: **for** iteration = 1 to MAX **do**
2: Episode length l=l_{init};
3: Run episode series i=1,...,N in the simulator and get: $(s_1^i, a_1^i, r_1^i, ..., s_l^i, a_l^i, r_l^i) \sim \mu_\theta$;
4: Compute the total reward $R_t^i = \sum_{t'=t}^{l} r_{t'}^i$;
5: $\Delta\theta = 0$;
6: **for** t = 1 to l **do**
7: Compute the baseline value $b_t = \frac{1}{N}\sum_{i=1}^{N} R_t^i$;
8: **for** i = 1 to N **do**
9: $\Delta\theta = \Delta\theta + \nabla_\theta \log \mu_\theta(s_t^i, a_t^i)(R_t^i - b_t)$;
10: **end for**
11: **end for**
12: l = l + ϵ, $\theta = \theta + \alpha\Delta\theta$;
13: **end for**

C. Training Process

In DeepMigration, we define the state, action and reward of the DRL framework as follows:

State: The state information is represented with the network status. The network status includes NF processing capability, current traffic load on NF instances, migration latency between NF instances and queueing length on NF instances, all of which mapped into a graph representation.

Action: The action in DeepMigration is denoted with a list of migrated traffic load among the NF instances. Based on the action, the policy translator converts the traffic load value to exact flows on each NF instance as the flow migration policy.

Reward: We use the overall evaluation of the migration quality as the reward function for the DRL agent, which is the sum of the cost defined in equation (2) and C_R. Also, we set a penalty coefficient ω and penalty calculating equations for the violation of the threshold Th^{high} and Th^{low} with $\omega \times (L - Th^{high})$ and $\omega \times (Th^{low} - L)$, respectively. The reward value is calculated after each action is executed.

The training process is shown in Algorithm 1. In line 1, we set the total number of iterations of the training process. Lines 2-4 run a series of experiment episode for the DRL agent and obtain the reward data. Lines 5-11 calculate the update value for the parameters of the neural networks based on the obtained reward data. Line 12 regulates the episode length and updates the neural networks of the DRL agent.

VI. Evaluation

A. Experiment Setup

In our experiment, the functional modules of DeepMigration are implemented as modules on a POX controller with Ten-

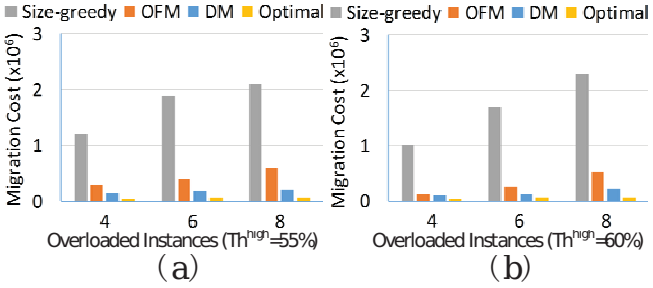

Fig. 4: Performance in the scale-out scenario

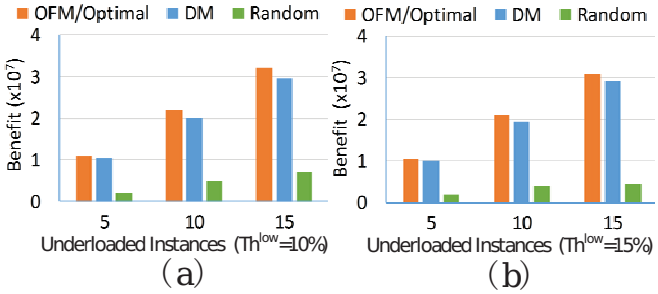

Fig. 5: Performance in the scale-in scenario

sorFlow. The network testbed is built with eight servers and one work station. The eight servers run NF instances, and the work station runs the POX controller. Each server is equipped with two Intel Xeon E5-2600 CPUs, 128G RAM, two 10G NICs, and Ubuntu 16.04. The work station has one Intel Xeon W2145 CPU, 256G RAM, one TITAN V GPU, and one 10G NIC. All the servers are connected to a PICA8 switch, and the work station is connected to the PICA8 switch through the controller port. We use a real-world traffic pattern LBNL/ICSI trace [29] to evaluate the performance of DeepMigration. In our experiment, we train the model with 40000 iterations. The learning rate α is set with an initial value of 5×10^{-3} and declines to a final value of 1×10^{-3} as the number of iteration increases.

B. Comparison Algorithms

We compare the performance among the following schemes:

1) **OFM** [10] uses one three-step heuristic algorithm for the scale-out scenario, one three-step heuristic algorithm for the load balancing scenario, and solves the optimization problem to get an optimal solution for the scale-in scenario.
2) **Size-greedy** picks the largest flow from the hot spot to migration flows to other NF instances. It maximizes the size of flows placed on current existing instances and minimizes the required number of new NF instances in the scale-out scenario.
3) **Pairwise** iteratively picks an overloaded and an underloaded NF instance as a pair and then migrates the flows between each pair of instances for load balancing.
4) **Random** picks an underloaded NF instance to another instance randomly in the scale-in scenario.
5) **DeepMigration (DM)** In DeepMigration, use a set of GNNs to calculate the flow migration policy. The GNNs are trained through a DRL framework.

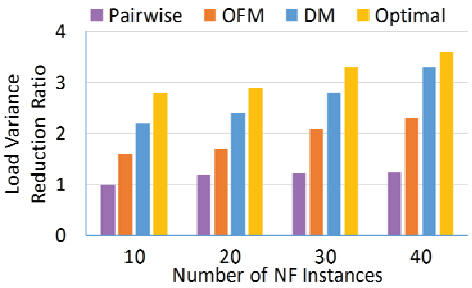

Fig. 6: Performance in the load balancing scenario

6) **Optimal** solves the optimization problem (3)-(6) and gets an optimal solution for each scenario.

C. Performance

In this section, we compare the performance of DeepMigrationwith other schemes in the following aspects: migration performance in scale in/scale out/load balancing scenarios, and the migration policy calculation time.

Scale-out. In this scenario, we implement ten NF instances of the same resource allocation with the type of Prads [10], and set the number of overloaded instances as 4,6,8. The traffic pattern follows the LBNL/ICSI trace. When the number of overloaded instances increases, the addition of NF instances is more likely to be required. The threshold Th_{high} is also set to 55% and 60% respectively to test the performance under different trigger requirements. We compare the performance of DeepMigration with OFM, the optimal solution and a size-greedy algorithm, and the result is shown in Fig. 4. As the figure shows, compared to size-greedy and OFM, DeepMigration induces less migration cost and the performance is closer to the optimal solution. When the Th^{high} is set to 60%, DeepMigration has 63.3% less migration cost than OFM (2.2×10^5 compared to 6.0×10^5). This is mainly due to the advantage of GNN in the analysis of graph-structured data.

Scale-in. When there are underloaded NF instances, we can migrate the flows of such instances to others and reduce the overall number of instances, which brings a scale-in benefit. To test the performance under this scenario, we implement totally 50 instances, and set the number of underloaded instances to 5, 10, and 15, respectively. The LBNL/ICSI trace is scattered to generate a low traffic load on NF instances. We also set Th_{low} to 10% and 15% to test the performance under different trigger conditions. The result is shown in Fig. 5. As shown in the figure, DeepMigration shows similar performance with the optimal solution. Both DeepMigration and OFM perform far better than the random algorithm.

Load Balancing. In the load balancing scenario, the migration policy aims to reduce the variance of the traffic load among all NF instances. In our experiment, we vary the number of NF instances from 10 to 40 to test the performance under different network scale. We use var_{ori} to denote the original variance of traffic load and var_{lb} to denote the traffic load variance after a load balancing operation, then we can evaluate the load balancing performance with $ratio = \mathrm{var}_{ori}/\mathrm{var}_{lb}$. As shown in Fig. 6, DeepMigration achieves an obvious higher variance reduction ratio (2.23, 2.42, 2.87 and 3.36) than OFM

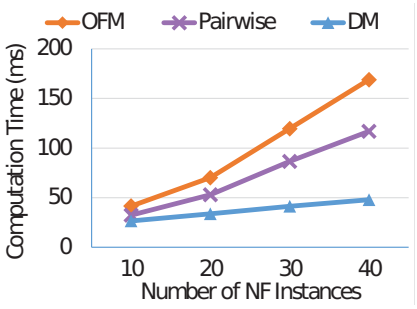

Fig. 7: Computation time

(1.63, 1.76, 2.15, and 2.32) and Pairwise (1.02, 1.21, 1.22, and 1.23), and the ratio is approximate to the optimal solution (2.75, 2.82, 3.21, and 3.57) when the number of NF instances is 40.

Computation Time. The computation time is used to reflect the scheme's scalability. We pick flows from LBNL/ICSL trace to generate similar traffic patterns for different numbers of NF instances. Moreover, we use the load balancing scenario to test the computation time of DeepMigration. The average number of flows on each instance is set to 40, and the number of instances varies from 10 to 40. As shown in Fig. 7, DeepMigration can significantly reduce the computation time compared to Pairwise and OFM. Furthermore, as the scale of the network expands from 10 to 40, the computation time of DeepMigration increases by 104% (from 24 ms to 49 ms), which is much less than the Pairwise (277%) and OFM (312%). Specifically, when the number of NF instances is 40, the computation time of DeepMigration is only 28.4% of OFM and 41% of Pairwise. Therefore, DeepMigration has a better scalability performance than Pairwise and OFM.

VII. Conclusion

In this paper, we propose DeepMigration to efficiently solve the flow migration problem and improve the QoS in NFV elastic control. DeepMigration combines GNN and DRL to generate dynamic flow migration policies based on the varying network traffic. DeepMigration is deployed online and adjusts the flow migration policy based on continuously monitoring the network status. The results shows that DeepMigration can reduce the flow migration cost compared to existing schemes. The advantage of DeepMigration validated in this paper proves the potential of GNN and DRL in solving the control problems in networks.

References

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

[2] R. Mijumbi, J. Serrat, and et al., "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[3] B. Yi, X. Wang, and et al., "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[4] F. Z. Yousaf, M. Bredel, and et al., "NFV and SDN-Key technology enablers for 5G networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.

[5] Z. Xu, F. Liu, and et al., "Demystifying the energy efficiency of network function virtualization," in *IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, 2016, pp. 1–10.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[7] A. Roy, H. Zeng, and et al., "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[8] N. McKeown, T. Anderson, and et al., "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[9] A. Gember-Jacobson, R. Viswanathan, and et al., "OpenNF: Enabling innovation in network function control," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 163–174.

[10] C. Sun, J. Bi, and et al., "Enabling NFV elasticity control with optimized flow migration," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2288–2303, 2018.

[11] Y. Wang, G. Xie, and et al., "Transparent flow migration for NFV," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 2016, pp. 1–10.

[12] B. Kothandaraman, M. Du, and P. Sköldström, "Centrally controlled distributed VNF state management," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. ACM, 2015, pp. 37–42.

[13] S. Woo, J. Sherry, and et al., "Elastic scaling of stateful network functions," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 299–312.

[14] A. Gember-Jacobson and A. Akella, "Improving the safety, scalability, and efficiency of network function state transfers," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. ACM, 2015, pp. 43–48.

[15] S. Palkar, C. Lan, and et al., "E2: a framework for NFV applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015, pp. 121–136.

[16] Y. Lin, U. C. Kozat, and et al., "Pausing and resuming network flows using programmable buffers," in *Proceedings of the Symposium on SDN Research*. ACM, 2018, p. 7.

[17] M. Alhamad, T. Dillon, and E. Chang, "Conceptual SLA framework for cloud computing," in *4th IEEE International Conference on Digital Ecosystems and Technologies*. IEEE, 2010, pp. 606–610.

[18] L. Wu, S. K. Garg, and R. Buyya, "SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments," in *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2011, pp. 195–204.

[19] A. Sadeghi, G. Wang, and G. B. Giannakis, "Deep reinforcement learning for adaptive caching in hierarchical content delivery networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1–10, Dec. 2019.

[20] G. Wang, B. Li, and G. B. Giannakis, "A multistep lyapunov approach for finite-time analysis of biased stochastic approximation," *arXiv:1909.04299*, 2019.

[21] X. Chen, C. Wu, T. Chen, H. Zhang, Z. Liu, Y. Zhang, and M. Bennis, "Age of information-aware radio resource management in vehicular networks: A proactive deep reinforcement learning perspective," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2020.

[22] D. Silver, G. Lever, and et al., "Deterministic policy gradient algorithms," 2014.

[23] R. S. Sutton, D. A. McAllester, and et al., "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[24] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *Journal of Machine Learning Research*, vol. 5, no. Nov, pp. 1471–1530, 2004.

[25] H. Mao, M. Schwarzkopf, and et al., "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 2019, pp. 270–288.

[26] H. Hu, Z. Liu, and J. An, "Mining mobile intelligence for wireless systems: A deep neural network approach," *IEEE Computational Intelligence Magazine*, vol. 15, no. 1, pp. 24–31, 2020.

[27] J. Zhou, G. Cui, and et al., "Graph neural networks: A review of methods and applications," *arXiv:1812.08434*, 2018.

[28] P. W. Battaglia, J. B. Hamrick, and et al., "Relational inductive biases, deep learning, and graph networks," *arXiv:1806.01261*, 2018.

[29] "LBNL/ICSI enterprise tracing project," http://www.icir.org/enterprise-tracing, accessed October 4, 2019.