Fair Virtual Network Function Scheduling with Deep Reinforcement Learning

Zhenran Kuai and Shaowei Wang
School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China
Email: DZ1923021@smail.nju.edu.cn, wangsw@nju.edu.cn

GLOBECOM 2021 - 2021 IEEE Global Communications Conference | 978-1-7281-8104-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/GLOBECOM46510.2021.9686006

***Abstract*—Network function virtualization aims at deploying network functions on general-purpose hardwares, referred to as virtual network functions (VNFs), rather than the specialized devices. A network service can be implemented by scheduling different VNFs. In this paper, we study the VNF scheduling problem with the objective of satisfying the diversified end-to-end delay requirements while maintaining the fairness among different network services. We formulate the problem as a mixed integer nonlinear program and propose a VNF scheduling algorithm based on deep reinforcement learning, in which proximal policy optimization is adopted to optimize the policy network. Numerical results show that the proposed scheduling algorithm outperforms other canonical ones and the designed policy network can scale to fit different problem sizes.**

***Index Terms*—Deep reinforcement learning, network function virtualization, virtual network function scheduling.**

I. Introduction

Network function virtualization (NFV) is transforming the way that network operators deploy network services and execute network functions [1]. Traditionally, due to the specific requirements of different services, certain types of network functions are implemented on dedicated hardwares (e.g., firewall, deep package inspection, load balancers, etc.). However, emerging new services demand a large number of dedicated devices to support diverse network functions, which incurs high capital expense and operational expenditure. For another, service innovation is accelerating, but dedicated hardwares cannot be updated simply. Newly deployed devices have only a short life cycle. In the NFV framework, network functions can be implemented by virtual network functions (VNFs), which are realized by softwares on virtual machines or containers. A network service can be represented by a service function chain (SFC) composed of the required VNFs. The data flow is traversed through the SFC sequentially to provide the service.

When enabling network services with NFV technology, several research problems need to be addressed. For each required VNF type, we need to determine how many VNF instances to implement and where to deploy in a physical network. Moreover, these instances should be assigned and chained together to form complete SFCs, which is also referred to as VNF mapping. If one VNF instance is shared across SFCs, the execution sequence will be scheduled to achieve a global optimization objective. This challenge is known as VNF scheduling, where two main constraints should be satisfied: the precedence constraint of VNFs and the computing resource constraint due to the limited number of VNF instances.

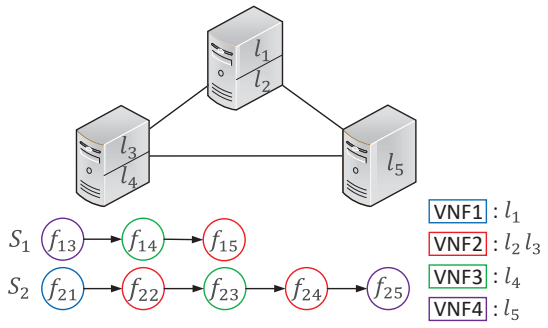
In [2], the authors formulate the joint VNF mapping and scheduling as a flexible job shop scheduling problem and provide a two-stage idea for future solutions. In [3], greedy strategy and tabu are proposed to resolve the online VNF mapping and scheduling problem. In [4], the authors propose a reinforcement learning method to learn the scheduling policy, in which the reward function is designed to satisfy the different delay requirements of all services. In [5], a cost efficient heuristic algorithm is proposed to address the VNF placement and scheduling in public cloud networks, in which the impacts of VNF threading attributes and deployment time are considered. In [6], resource sharing and preemption are allowed between VNF instances, and the authors develop a genetic algorithm to evaluate the effectiveness of the proposed sharing and preemption model.

In future communication networks, diversified network services should be provided by network operators and different types of traffic will be multiplexed [7], which implies flexible scheduling scheme is necessary to fulfill various quality of service (QoS) requirements while maintaining fairness. Managing the network resources has become a major challenge for service customization [8]. In the aforementioned researches, little attention is paid to satisfy service specific QoS requirements, especially the fairness between services and the alternative of VNF mapping.

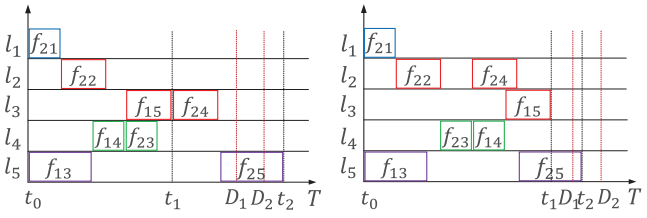
In this paper, we investigate the VNF scheduling problem from the viewpoint of specific end-to-end delay requirements while maintaining fairness. VNF mapping is also incorporated in our scheme, since efficient utilization of computing resources can greatly affect the performance of each service. We formulate the VNF scheduling as a mixed integer linear program problem. Then we propose a VNF scheduling algorithm based on deep reinforcement learning (DRL), where the decisions of VNF mapping are incorporated in the scheduling action space. The policy network takes the features of each action as input, thus the learned policy can be applied to different problem sizes. The reward function can decompose the ultimate performance into reward components in each step, which alleviates the impacts of reward sparsity. Numerical results show that the priority of scheduling each candidate VNF is well learned with the designed policy network.

This work was supported in part by the National Natural Science Foundation of China under Grants 61931023 and U1936202.

978-1-7281-8104-2/21/$31.00 ©2021 IEEE



(a) A substrate network and SFCs.



(b) Two scheduling results.

Fig. 1. A VNF scheduling example.

The rest of the paper is organized as follows: In Section II, we give our system model and formulate the VNF scheduling problem as a mixed integer nonlinear program. In Section III, the proposed VNF scheduling algorithm is presented in detail. In Section IV, numerical results and discussions are presented. In Section V, we conclude our work.

II. System Model and Problem Formulation

A. System Model

Consider an NFV system consisting of a substrate network and multiple network services. The substrate network is represented by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of NFV nodes (e.g., host servers, switches) in the network and $\mathcal{E}$ represents the set of physical links. Each NFV node can host multiple VNF instances of different types. Suppose L VNF instances have been placed on the NFV nodes, then the set of VNF instances is denoted by $\mathcal{L} = \{l_1, ..., l_L\}$.

A network service can be represented by an SFC, which is composed of an ordered sequence of VNFs. VNF scheduling is performed in each scheduling period. We define a set of m processing SFCs as $\mathcal{P} = \{S_1, ..., S_m\}$, where all VNFs have been mapped and scheduled in the last scheduling period and data flows have traversed through partial VNFs. The set of n SFCs $\mathcal{Q} = \{S_{m+1}, ..., S_{m+n}\}$ arriving in the current scheduling period are waiting to be scheduled, which are named as pending SFCs. The set of VNFs in S_i is denoted by $\mathcal{F}_i$ and the j-th VNF in S_i is denoted by f_{ij}. Let $\mathcal{F}$ represent the set of all VNFs. Denote D_i as the end-to-end delay requirement of S_i. Let $\mathcal{L}_{ij} \subseteq \mathcal{L}$ represent the set of VNF instances which are able to process f_{ij}.

Given the VNF placement in the substrate network, VNF scheduling is performed periodically to orchestrate the VNFs in processing and pending SFCs. A simple example is shown in Fig. 1. Consider one processing SFC S_1 and one pending SFC S_2. The VNFs in S_1 have been mapped to specific VNF instances in the previous scheduling period. Due to the computational cost of VNF remapping, the mapping results of the processing SFC are reserved and the execution orders can be rescheduled. As for the pending SFC S_2, VNFs should first be mapped to the instances. Then in the scheduling process, the VNFs should not start before the completion time of the predecessors in the same SFC. When multiple VNFs are mapped to the same VNF instance, only one VNF can be processed at a time.

In this example, the completion time of S_1 is t_1. We define the earliness of S_1 as $D_1 - t_1$, which is the time advance of the service completion over the end-to-end delay requirement. As shown in the first scheduling result in Fig. 1(b), S_2 does not meet the delay requirement while S_1 has a sufficient earliness. However, both delay requirements can be satisfied in the second scheduling result. In addition, if the earliness of S_2 is not sufficient, the available time for scheduling will decrease as the scheduling period progresses. S_2 has no spare time to help the following pending services meet the delay requirements. Therefore, maintaining fairness is also meaningful for the following scheduling processes.

B. Problem Formulation

For each service, the goal is to satisfy its end-to-end delay requirement, thus the VNFs along the SFC S_i should be scheduled to maximize its earliness. However, fairness cannot be guaranteed when the competing QoS requirements are fulfilled. Therefore, the minimum earliness over all SFCs should be maximized to prevent one SFC from having significant superiority over the others, which is expressed by

$$\min_i \mathcal{W}_i = \min_{i,j} \{D_i - (t^s_{ij} + \sum_{l_k \in \mathcal{L}_{ij}} y_{ijk}\rho_{ijk})\}, \tag{1}$$

where t^s_{ij} represents the start time of f_{ij} and ρ_{ijk} represents its duration on the VNF instance $l_k \in \mathcal{L}_{ij}$. y_{ijk} is a binary variable indicating if the VNF instance $l_k \in \mathcal{L}_{ij}$ is assigned to process f_{ij}.

We need to make decisions on the start time of VNFs and the mapping of pending SFCs. The feasibility of a schedule should be guaranteed by satisfying the following constraints. First, the processing of a VNF starts after the completion of its predecessor in an SFC. This precedence constraint is expressed by

$$t^s_{i,j+1} - t^s_{ij} \geq \sum_{l_k \in \mathcal{L}_{ij}} y_{ijk}\rho_{ijk}, \quad \forall S_i \in \mathcal{P} \cup \mathcal{Q}, \forall f_{ij} \in \mathcal{F}_i \setminus \{f_{i|\mathcal{F}_i|}\}. \tag{2}$$

Meanwhile, one VNF has to be processed after the completion of the other when two VNFs are mapped to the same instance. Let $\mathbb{I}(\cdot)$ represent an indicator function which equals to 1 if

an event holds or 0 otherwise, the constraints for all f_{ij} and f_{pq} in different SFCs are given by

$$\begin{aligned}&\mathbb{I}(t^s_{ij} > t^s_{pq})(t^s_{ij} - t^s_{pq}) \geq \\ &\qquad \mathbb{I}(t^s_{ij} > t^s_{pq}) y_{ijk} y_{pqk} \rho_{pqk}, \forall l_k \in \mathcal{L}_{ij} \cap \mathcal{L}_{pq}, \\ &\mathbb{I}(t^s_{pq} > t^s_{ij})(t^s_{pq} - t^s_{ij}) \geq \\ &\qquad \mathbb{I}(t^s_{pq} > t^s_{ij}) y_{ijk} y_{pqk} \rho_{ijk}, \forall l_k \in \mathcal{L}_{ij} \cap \mathcal{L}_{pq}.\end{aligned} \tag{3}$$

For processing SFCs, the VNF mapping results are reserved. For pending SFCs, one VNF can only be mapped to one VNF instance. Thus we have

$$y_{ijk} = \mathbb{I}(l_k = l^p_{ij}), \forall S_i \in \mathcal{P}, \forall f_{ij} \in \mathcal{F}_i, \forall l_k \in \mathcal{L}_{ij}, \tag{4}$$

$$\sum_{l_k \in \mathcal{L}_{ij}} y_{ijk} = 1, \forall S_i \in \mathcal{Q}, \forall f_{ij} \in \mathcal{F}_i, \tag{5}$$

where l^p_{ij} is the instance to which f_{ij} in $S_i \in \mathcal{P}$ is mapped. Therefore, the problem can be formulated as a mixed integer nonlinear program (MINLP):

$$\begin{aligned}&\max_{y_{ijk}, t^s_{ij}} : \min_{i,j}\{D_i - (t^s_{ij} + \sum_{l_k \in \mathcal{L}_{ij}} y_{ijk}\rho_{ijk})\}, \\ &s.t. : (2) - (5), \\ &\qquad t^s_{ij} \geq 0, \forall S_i \in \mathcal{P} \cup \mathcal{Q}, \forall f_{ij} \in \mathcal{F}_i, \\ &\qquad y_{ijk} \in \{0,1\}, \forall S_i \in \mathcal{P} \cup \mathcal{Q}, \forall f_{ij} \in \mathcal{F}_i, \forall l_k \in \mathcal{L}_{ij}.\end{aligned} \tag{6}$$

This problem is a standard flexible job shop scheduling problem, which is NP-hard [9]. To efficiently generate a feasible schedule, priority dispatching rule (PDR) is widely used in real-world scheduling systems. It makes scheduling decisions according to some specific rules. For example, we can select the SFC based on the longest remaining processing time and schedule the first VNF waiting to be processed. Then the VNF instance to process the VNF can be selected based on the shortest processing time. These steps are repeated until a complete schedule is obtained.

Compared with optimization methods and metaheuristics, PDR is computationally fast and easy to implement while the others are not applicable when problems scale up. However, the following factors restrict its design for our problem: (1) A simple PDR cannot meet the complicated requirements; (2) Designing an effective PDR requires substantial expertise; (3) There exists an interplay between VNF mapping and scheduling. Therefore, we adopt DRL to address these issues in this work.

III. Deep Reinforcement Learning For VNF Scheduling

We first describe the VNF scheduling under a Markov decision process (MDP) framework, then present the DRL-based algorithm in detail.

A. Markov Decision Process Formulation

We use the designed PDR to generate a schedule with $|\mathcal{F}|$ consecutive decisions, each of which is involved with a VNF in SFCs. In particular, one VNF will be selected from the available VNF set for mapping and scheduling in each step.

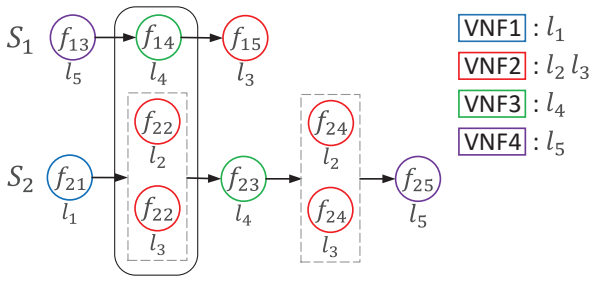


Fig. 2. An example of available actions.

The start time of the selected VNF is arranged on the settled instance as early as possible to generate a tight schedule [10]. As given in (6), the whole process is completed when the start time and mapping decisions of all VNFs are confirmed. The main components in this MDP are illustrated as follows.

1) Action: An action is to make decisions on mapping and scheduling. Decomposing the decision into a map-then-schedule style or vice versa ignores the existing interplay between them [11]. Therefore, we consider the mapping and scheduling decisions as actions in one action space. The example shown in Fig. 2 is used for illustration. Suppose f_{13} and f_{21} have been scheduled, f_{14} and f_{22} are available. The selectivity of the VNF mapping can be incorporated into the scheduling of f_{22}. Then we can choose from three candidate actions: f_{14} on l_4, f_{22} on l_2 and f_{22} on l_3. Let $\mathcal{A}_t = \{a^1_t, ..., a^{|\mathcal{A}_t|}_t\}$ represent the set of candidate actions in step t.

2) State: A state $s_t \in \mathcal{S}$ representing current progress of scheduling contains the raw features of all candidate actions. The state can be expressed as $s_t = [h^1_t, ..., h^{|\mathcal{A}_t|}_t]$, where h^k_t refers to the raw features of the k-th action in $\mathcal{A}_t$. All the features are listed in Table I. Note that the first three features are derived from the SFC where the action is located and the following four features are involved with VNF instances. Specifically, C^{LB}_{ijk} and C_{ijk} are determined for scheduled VNFs, and they can be rewritten as C_{ij}. For a candidate action, it is calculated as $C^{LB}_{ijk} = C_{i,j-1} + \rho_{ijk}$. The slack time T^s_i is calculated as $T^s_i = D_i - C^{LB}_{ijk}$. When a VNF f_{ij} of one processing SFC mapped on l_k is scheduled, T^p_k is updated as $T^p_k = T^p_k - \rho_{ijk}$.

3) State Transition: Once an action happens, the mapping decision is determined and the actions relating to the same VNF are excluded from $\mathcal{A}_t$. The actions relating to the successor VNF in the same SFC are included. Features of the new available action set $\mathcal{A}_{t+1}$ represent the state of the next step.

4) Reward: The minimum earliness $\min_i \mathcal{W}_i$ can only be observed after a complete schedule. To mitigate the impact of reward sparsity, we decompose the ultimate performance into reward components in each step:

$$r(s_t, a_t) = \min_{i,j}\{D_i - C_{ij}(t+1)\} - \min_{i,j}\{D_i - C_{ij}(t)\}, \tag{7}$$

TABLE I
THE STATE REPRESENTATIONS OF AN SFC

Feature	Description
I_i^c	Indicate whether the SFC is complete.
D_i	The due time of the SFC.
T_i^s	The slack time before the due time.
C_{ijk}^{LB}	The lower bound of the completion time of f_{ij} on l_k.
ρ_{ijk}	The processing time of f_{ij} on l_k.
C_{ijk}	The completion time of f_{ij} on l_k.
T_k^p	The total time required for the VNF instance l_k to process the VNFs in the processing SFCs.

where $C_{ij}(t)$ represents the completion time of completed VNF f_{ij} before t. Thus Eq. (7) estimates the decrement of minimum earliness caused by a_t. The cumulative discounted reward is $\sum_{t=1}^{|\mathcal{F}|}\gamma^{t-1}r(s_t,a_t)$. When the discount factor $\gamma=1$, the cumulative reward is $\min_{i,j}\{D_i-C_{ij}\}-\min_i\{D_i\}$, which conforms to the minimum earliness.

5) Policy: In step t, the policy function $\pi_\theta:\mathcal{S}\rightarrow P(\mathcal{A}_t)$ takes the state s_t as input and outputs a distribution over all actions in $\mathcal{A}_t$.

B. Proximal Policy Optimization for VNF Scheduling

In reinforcement learning, the goal of an agent is to generate a high-quality policy in the process of interacting with an environment. To this end, the agent needs to go through different states and actions to evaluate and optimize the policy iteratively. In this paper, proximal policy optimization (PPO) [12] is adopted to optimize the policy network.

1) Policy Network: In the VNF scheduling problem, the policy π_θ is optimized to generate an accurate distribution over $\mathcal{A}_t$ with s_t. Since the state space is too large to explore, neural network is used as a policy function approximator that takes the raw features as input. Specifically, we adopt a multi-layer perceptron to obtain a scalar value $v(a_t^k)=MLP_\theta(h_t^k)$ for each action a_t^k in $\mathcal{A}_t$. A softmax operation is performed over the scalar values to output a distribution $P(\mathcal{A}_t)$, from which the action a_t is sampled. Here, the parameters are shared across the candidate actions, thus the policy network can deal with the situation where the size of $\mathcal{A}_t$ is different in each step and is potentially size-agnostic.

2) Critic Network: After the sampled action a_t is executed, r_t and s_{t+1} will be observed. Repeating this process will generate a sampled trajectory τ of length $|\mathcal{F}|$: $\{(s_1,a_1,r_1),...,(s_{|\mathcal{F}|},a_{|\mathcal{F}|},r_{|\mathcal{F}|})\}$, which can be utilized to evaluate and optimize π_θ. Although the policy can be simply evaluated with the discounted cumulative reward $\sum_t\gamma^{t-1}r_t$, the variance of the sampled trajectory is too high. Thus the policy is evaluated with a critic network $V_\phi:\mathcal{S}\rightarrow\mathbb{R}$, which estimates the expected cumulative reward in state s_t. It has the same structure as the policy network, except that the last operation is a summation. The critic network is updated to minimize the mean square error between $V_\phi(s_t)$ and the target cumulative rewards:

$$L^{VF}(\phi)=\sum_{t=1}^{|\mathcal{F}|}\left[\sum_{i=t}^{|\mathcal{F}|}\gamma^{i-1}r_i-V_\phi(s_t)\right]^2. \tag{8}$$

Algorithm 1 PPO-Based VNF Scheduling

```
Initialize the policy network π_θ(a|s) with θ.
Initialize the critic network V_φ(s) with φ.
for iteration t = 1, 2, ··· , N_t do
    for environment e = 1, 2, ··· , N_e do
        Take actions according to the policy π_θ to generate
        a trajectory τ_e in the e-th environment;
    end for
    θ_old ← θ;
    for epoch p = 1, 2, ··· , N_p do
        Compute r_t(θ) and Â_t with τ_e for each environment;
        Compute L_e^F(θ, φ) for each environment;
        Update θ and φ by a gradient method w.r.t
        (Σ_e L_e^F(θ, φ))/N_e;
    end for
end for
```

3) Policy Optimization: Traditionally, a gradient ascent algorithm is applied on the samples of the performance objective $J(\theta)$ to optimize the policy [13]. The performance objective $J(\theta)$ and its gradient are written by

$$J(\theta)=\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{|\mathcal{F}|}\gamma^{t-1}r_t\right], \tag{9}$$

$$\nabla J(\theta)\approx\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{|\mathcal{F}|}\nabla_\theta\log\pi_\theta(a_t|s_t)\hat{A}_t\right], \tag{10}$$

where $\hat{A}_t=r_t+\gamma V_\phi(s_{t+1})-V_\phi(s_t)$ measures the advantage of a_t over the other actions in s_t. As given in Eq. (9), the objective is based on the trajectory sampled from the policy being optimized. After the parameters of the policy are updated with a single gradient ascent, whether the performance of the new policy π_θ is improved is unpredictable. PPO applies importance sampling to the objective, which is expressed by

$$L^{CLIP}(\theta)=\sum_{t=1}^{|\mathcal{F}|}\Big[\min(r_t(\theta)\hat{A}_t,\mathrm{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\hat{A}_t)\Big], \tag{11}$$

where $r_t(\theta)=\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio. With importance sampling, $r_t(\theta)\hat{A}_t$ exploits the trajectory of an old policy $\pi_{\theta_{old}}$ to obtain the expected performance objective of the new policy π_θ. Optimizing the surrogate objective will improve π_θ. Each trajectory can be exploited for several gradient ascent steps to guarantee the improvement. However, if the policy update is excessively large, the divergence between these two policies could not be corrected by the probability ratio of the samples. To restrict the divergence, $r_t(\theta)$ is clipped to the

TABLE II
SIMULATION PARAMETERS

Parameter	Value
γ	1
ϵ	0.1
Coefficient of L^{CLIP}	2
Coefficient of L^{VF}	0
Learning rate	0.0002
Optimizer	Adam
Number of iterations	$N_t = 10000$
Number of parallel environments	$N_e = 100$
Number of epochs	$N_p = 3$
Activation function	ReLU

interval $[1-\epsilon, 1+\epsilon]$. Then the aforementioned objectives can be combined as

$$L^F(\theta, \phi) = c_1 L^{CLIP}(\theta) - c_2 L^{VF}(\phi), \tag{12}$$

where c_1 and c_2 are coefficients. The parameters of the policy and critic network are updated by a gradient ascent step to optimize the surrogate objective $L^F(\theta, \phi)$. Multiple trajectories can be sampled from parallel environments to reduce the variance of training experiences. The procedure of the proposed PPO-based VNF scheduling is summarized in **Algorithm 1**.

IV. NUMERICAL RESULTS

We compare the PPO-based algorithm with the other three algorithms: random forest (RF) [14], greedy best availability (GBA) [3] and random scheduling (RS) algorithms. In the RF-based algorithm, RF is used to extract PDRs of mapping and scheduling from the solutions generated by Google OR-Tools [15]. The GBA scheduling algorithm is based on the VNF instances whose current VNF queue has the earliest completion time. The RS algorithm selects a random action from the available action set in each step.

A. Settings

Consider an NFV system with five NFV nodes. Since we focus on the performance of scheduling, the substrate network is fully connected and the transmission delay is not considered. 10 types of VNF instances are deployed on the network. For each type, the number of the available instances is set to 2 (i.e., $|\mathcal{L}_{ij}| = 2, \forall f_{ij}$), which is referred to as the flexibility of VNF instances. The process rate of each VNF type is randomly chosen from $\{1, 2, 3\}$ Mbps with equal probability.

For network services, the number of processing SFCs is $m = 5$ and the number of pending SFCs is $n = 5$. The length of pending SFCs is $|\mathcal{F}_i| = 10$. The length of processing SFCs is half that of the pending SFCs and each VNF in SFCs is randomly chosen from these 10 types with equal probability. The mapping result of each VNF in the processing SFCs is also randomly chosen from the available instances with equal probability. The deadline of each SFC is uniformly distributed between the total durations of VNFs and twice that, and the traffic size is set to an integer that is uniformly distributed over $[5, 10]$ Mbits. These parameters are generated independently in each parallel environment. In our settings, the policy and critic networks share the preceding layers except for the last layer to accelerate learning. The parameters of the algorithm are summarized in Table II.

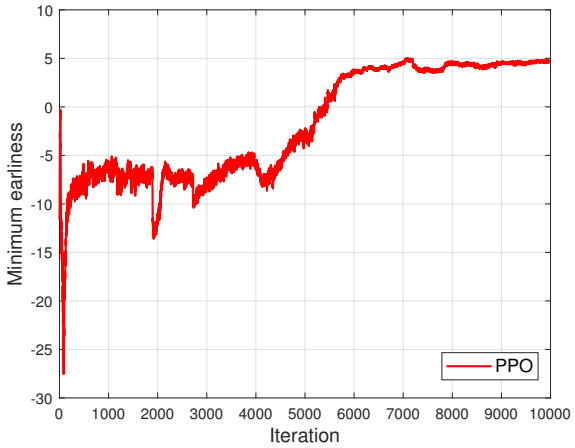


Fig. 3. Average minimum earliness as a function of the iteration process with 100 parallel environments.

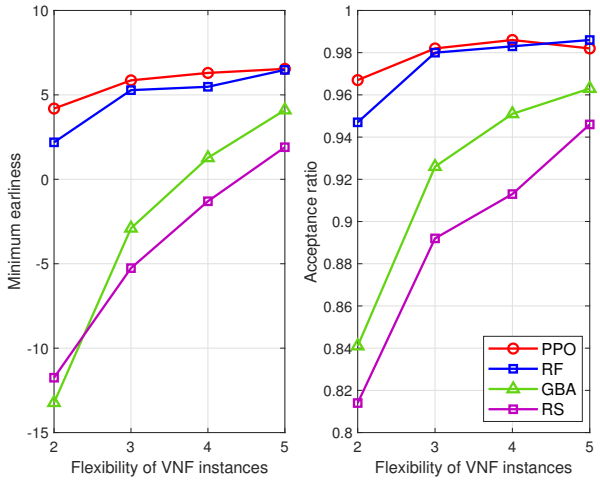


Fig. 4. Performance of the scheduling algorithms with the increasing flexibility of VNF instances.

B. VNF Scheduling Performance

The policy network is trained offline. Fig. 3 shows the average minimum earliness as a function of the iteration process, in which the number of parallel environments is $N_e = 100$. As we can see, the average minimum earliness increases as the learning process goes on. This implies the reward sparsity is resolved by the designed reward decomposition and the priorities of scheduling over the candidate actions are well learned with the policy network. Eventually the minimum earliness converges to a value greater than zero, which means

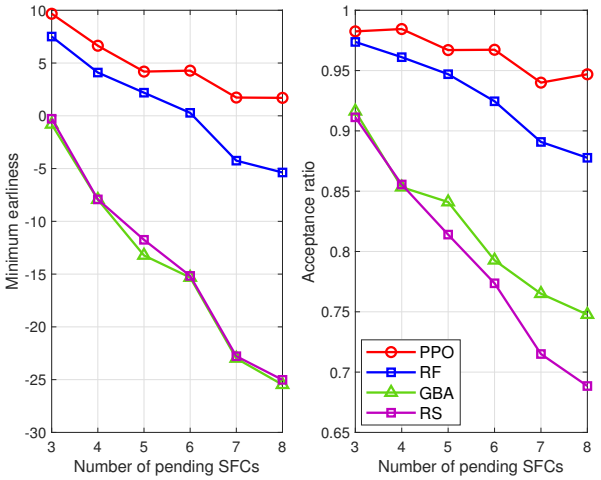


Fig. 5. Performance of the scheduling algorithms with the increasing pending SFCs.

most of the end-to-end delay requirements are satisfied and the fairness across SFCs is well maintained.

In the test stage, since the designed policy network is potentially size-agnostic, we directly apply the learned policy to perform scheduling in different settings to demonstrate its superiority and scalability. The policy is validated on 100 random environments in each setting.

Fig. 4 shows the average minimum earliness and acceptance ratio with the increasing flexibility of VNF instances. The acceptance ratio is defined as the proportion of the total SFCs that are completed before delay requirements. It can be seen that the proposed PPO-based algorithm outperforms other methods when the flexibility of VNF instances is equal to 2. This indicates that the policy learned by PPO can generate a more effective schedule on SFCs when the VNF instance resources are insufficient. However, with the increasing flexibility of VNF instances, the PPO-based algorithm shows a decreasing advantage over the others, and is even inferior to the RF-based algorithm in acceptance ratio when the flexibility is equal to 5. The reason is that there are plenty of VNF instances, so the same type of VNFs can choose other idle instances without competing for the same instance. The advantage of scheduling spare VNF instance resources diminishes.

Fig. 5 shows the performance with the increasing pending SFCs, in which the flexibility of VNF instances is equal to 2. As the number of pending SFCs increases, both the minimum earliness and acceptance ratio decrease because more VNFs have to wait for the completion of other VNFs when competing for the same VNF instance. The advantages of the PPO-based algorithm over the other algorithms also expand with the increasing pending SFCs, which implies the the proposed algorithm can maintain a much better performance than the others even if the NFV system is highly loaded. Meanwhile, the advantage obtained by using the same policy network shows that it has a good scalability for different problem sizes.

V. Conclusion

In this paper, we have investigated the VNF scheduling problem with the objective of maximizing the minimum earliness to satisfy the diversified end-to-end delay requirements while maintaining the fairness across SFCs. We formulate the problem as a MINLP and proposed a PPO-based algorithm to solve it. The decisions of VNF mapping are incorporated in the action space of scheduling to resolve the interplay between mapping and scheduling, and the ultimate performance was decomposed into reward components to mitigate the impact of reward sparsity in the scheduling process. Numerical results demonstrate that the proposed algorithm outperforms other methods. In addition, the designed policy network can scale up to fit different problem sizes.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, 2015.

[2] J. F. Riera, E. Escalona, J. Batall¨ı, E. Grasa, and J. A. Garc¨a-Esp¨an, "Virtual network function scheduling: Concept and challenges," in *Proc. IEEE SaCoNeT'14*, 2014.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proc. IEEE NetSoft'15*, 2015.

[4] J. Li, W. Shi, N. Zhang, and X. Shen, "Delay-aware vnf scheduling: A reinforcement learning approach with variable action set," *IEEE Trans. Cogn. Commun. and Netw.*, vol. 7, no. 1, pp. 304–318, 2021.

[5] T. Gao, X. Li, Y. Wu, W. Zou, S. Huang, M. Tornatore, and B. Mukherjee, "Cost-efficient vnf placement and scheduling in public cloud networks," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4946–4959, 2020.

[6] Y. Zhang, F. He, T. Sato, and E. Oki, "Network service scheduling with resource sharing and preemption," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 764–778, 2020.

[7] T. Wang, W. Yu, and S. Wang, "Inter-slice radio resource management via online convex optimization," in *Proc. IEEE ICC'21*, 2021.

[8] T. Wang, S. Wang, and Z.-H. Zhou, "Machine learning for 5g and beyond: From model-based to data-driven mobile wireless networks," *China Commun.*, vol. 16, no. 1, pp. 165–175, 2019.

[9] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, 1976.

[10] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *Proc. NeurIPS'20*, 2020.

[11] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, 2008.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] X. Sheng and S. Wang, "Sensing-transmission tradeoff for multimedia transmission in cognitive radio networks," in *Proc. IEEE GLOBECOM'20*, 2020.

[14] S. Jun, S. Lee, and H. Chun, "Learning dispatching rules using random forest in flexible job shop scheduling problems," *Int. J. Prod. Res.*, vol. 57, no. 10, pp. 3290–3310, 2019.

[15] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/