Delay-Aware VNF Scheduling: A Reinforcement Learning Approach With Variable Action Set

Junling Li, *Graduate Student Member, IEEE*, Weisen Shi, *Graduate Student Member, IEEE*, Ning Zhang, *Senior Member, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—Software defined networking (SDN) and network function virtualization (NFV) are the key enabling technologies for service customization in next generation networks to support various applications. In such a circumstance, virtual network function (VNF) scheduling plays an essential role in enhancing resource utilization and achieving better quality-of-service (QoS). In this paper, the VNF scheduling problem is investigated to minimize the makespan (i.e., overall completion time) of all services, while satisfying their different end-to-end (E2E) delay requirements. The problem is formulated as a mixed integer linear program (MILP) which is NP-hard with exponentially increasing computational complexity as the network size expands. To solve the MILP with high efficiency and accuracy, the original problem is reformulated as a Markov decision process (MDP) problem with variable action set. Then, a reinforcement learning (RL) algorithm is developed to learn the best scheduling policy by continuously interacting with the network environment. The proposed learning algorithm determines the variable action set at each decision-making state and captures different execution time of the actions. The reward function in the proposed algorithm is carefully designed to realize delay-aware VNF scheduling. Simulation results are presented to demonstrate the convergence and high accuracy of the proposed approach against other benchmark algorithms.**

***Index Terms*—Delay-aware VNF scheduling, SDN, NFV, resource allocation, reinforcement learning.**

I. Introduction

With the fast evolvement of communication technologies, the 5G wireless networks are anticipated to accommodate a massive number of Internet-of-Things (IoT) devices with highly diversified quality-of-service (QoS) requirements [2], [3]. Conventional network functions are placed at function-specific network servers or middleboxes, which are cost-ineffective for differentiated service customization since a large number of servers need to be augmented to support different types of services. In addition, the traffic routing paths among network elements are distributedly calculated with considerable overhead, which is not efficient in achieving high network performance with load balancing [4], [5]. In the 5G era, new requirements arise for the evolving network paradigm, including 1) cost-effective network deployment for seamless device access, 2) enhanced resource utilization to accommodate high traffic volume, and 3) flexible function placement for service customization [6], [7]. Software-defined networking (SDN) and network function virtualization (NFV) are two promising and complementary technologies to reduce service function provisioning cost and improve heterogeneous resource utilization for customized end-to-end (E2E) service delivery, and thus have attracted significant attention from both industry and academia [8]–[10]. SDN decouples the network control from forwarding functions, which facilitates agile network management and enables network programmability. NFV abstracts network functions, e.g., firewall and deep packet inspection (DPI), from dedicated devices, and allows network functions to be virtualized and placed on high capacity commodity servers in the network. In this way, network services can be provided by orchestrating virtual network functions (VNFs) at different network servers to form VNF chains, which are embedded onto a physical substrate network. Integrating SDN and NFV for future networking is expected to feature centralized network management, virtualized service function chaining, reduced costs, and enhanced service quality [11].

One of the fundamental research issues in NFV is how to efficiently and fairly allocate physical resources in the substrate network to support the embedding of multiple VNF chains, referred to as the NFV resource allocation (NFV-RA) problem [12]. The NFV-RA problem typically consists of VNF composition, VNF chain embedding, and VNF scheduling on embedded NFV nodes. During the embedding process, multiple VNFs can be placed onto a common NFV-enabled network commodity server (i.e., NFV node) to reduce function provisioning cost and improve physical resource utilization. In VNF scheduling, the execution timings of embedded VNFs on NFV nodes are scheduled to minimize the makespan (i.e., the time period from the execution of the first VNF to the completion of last VNF among all the scheduled VNFs for all the services). Existing studies have shown that the classical VNF scheduling problem can be formulated as a job-shop problem (JSP) [13], [14], which is an NP-hard combinatorial optimization problem [15]. To obtain its near-optimal

Manuscript received August 3, 2019; revised November 17, 2019 and March 5, 2020; accepted April 8, 2020. Date of publication April 21, 2020; date of current version March 8, 2021. This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada. This paper is presented in part at IEEE/CIC ICCC 2019 [1]. The associate editor coordinating the review of this article and approving it for publication was G. Ding. *(Corresponding author: Ning Zhang.)*

Junling Li, Weisen Shi, and Xuemin Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: j742li@uwaterloo.ca; w46shi@uwaterloo.ca; sshen@uwaterloo.ca).

Ning Zhang is with the Department of Computing Science, Texas A&M University–Corpus Christi, Corpus Christi, TX 78412 USA (e-mail: ning.zhang@tamucc.edu).

Digital Object Identifier 10.1109/TCCN.2020.2988908

solutions, a number of heuristic/metaheuristic algorithms have been developed [16]–[20]. These heuristic algorithms (e.g., greedy algorithms) are in general fast and easy to implement, but their performance highly depends on the characteristics of the problem and may deteriorate as the network size expands. On the other hand, the metaheuristic algorithms such as particle swarm optimization (PSO) and genetic algorithm (GA) may suffer a low convergence rate in the iterative process, leading to increased computational cost and operational time. Their performance also relies on the initialization of parameters and may converge prematurely, falling into a local optimum especially for complex problems. Moreover, strict E2E delay requirements are important to 5G service provisioning. The incorporation of E2E delay constraints for different services in the standard JSP problem poses additional challenge to conventional VNF scheduling. With different E2E delay requirements, the existing heuristic algorithms for standard JSP need to be carefully adjusted to achieve delay-constrained VNF scheduling. This requires manual designs for appropriate rules of reducing the search space for the heuristic algorithms.

Recently, reinforcement learning (RL) has emerged as a promising approach to solve combinatorial optimization problems with reduced complexity and high accuracy [21]–[27]. In RL approaches, a learning agent learns the best policy iteratively by directly interacting with the environment, and multiple objectives can be supported simultaneously by designing an appropriate reward function for the learning system. In this way, the useful information for solving the problem at hand can be automatically extracted. The delay-aware VNF scheduling problem has a long-term objective which is to minimize the overall makespan. Also, the task of scheduling the VNFs in the system can be considered as a sequential decision process where the future status of the system depends on the current status and decision. Therefore, the VNF scheduling problem can be naturally modeled as a Markov decision process (MDP). This motivates us to employ RL to obtain near-optimal/optimal solutions to delay-aware VNF scheduling with reduced computational complexity.

In this paper, the delay-guaranteed VNF scheduling problem is first formulated as a mixed-integer linear program (MILP) which is NP-hard. To solve the MILP with high efficiency and accuracy, the original problem is reformulated as a Markov decision process problem. Then, we develop an RL framework to address the VNF scheduling problem with E2E delay guarantee. With the SDN/NFV-enabled network architecture, the logically centralized VNF scheduler has global network information and makes scheduling decisions. Based on this, we propose a centralized learning algorithm to solve the VNF scheduling problem. The action of the agent (i.e., the VNF scheduler) is improved through continuously interacting with the network environment. Different from conventional decentralized approaches for standard job-shop problems [23]–[25], the centralized RL improves the solution accuracy of the VNF scheduling, thanks to the global view of the whole system. Also, without the coordination message overhead between multiple agents, the solution complexity is reduced [26]. In the proposed RL approach, the action set for each state is finite, state-dependent, and fixed for a given state. At each decision-making state, a feasible/admissible action set is updated with the consideration of VNF dependency. Each action can take varying amount of time to complete, i.e., the length of the time period between two decision-making states is not fixed, depending on both the current state and the action taken. The effectiveness of our proposed scheduling algorithm is evaluated under different network scales and is compared with other heuristic algorithms. The main contributions of this work are four-fold:

1) We formulate the VNF scheduling problem as an MILP, considering the E2E delay requirements of differentiated services. To solve it, we transform the formulation into a discrete-time problem, where the VNF scheduling decisions are made at the beginning of each time slot;
2) A novel Q-learning based VNF scheduling algorithm is developed, in which system states, actions, variable action set, and reward function are designed for the learning algorithm. In conventional RL approaches, the feasible action set for the agent is state-independent and the execution time for all the actions are identical. However, in VNF scheduling, the action set for the agent are state-dependent since a VNF can be traversed only after its previous VNF has been passed through by the packet batch, and each action can take a varying amount of time to complete. Thus, our proposed Q-learning algorithm determines the variable action set at each decision-making state and accommodates diverse execution time of the actions;
3) The RL approach efficiently guarantees the QoS requirements for different services via a customized reward function that is composed of two parts. The first part reflects the makespan, where a short makespan leads to a large reward; The second part reflects whether the E2E delay requirements of the services are satisfied. If the delay requirement is satisfied for a given service, a positive reward is fed back to the agent. By iteratively accumulating the reward, the agent learns the optimal scheduling policy that results in the minimal makespan with E2E delay guarantee.
4) Extensive simulations are conducted to demonstrate the convergence of the proposed RL algorithm and to compare the performance of the proposed approach with that of several heuristic/metaheuristic algorithms and the decentralized approach. Simulation results show that our proposed approach can address the delay-aware VNF scheduling problem effectively with reduced complexity and high accuracy, and that the RL approach outperforms the benchmark algorithms in comparison.

The remainder of this paper is organized as follows. In Section II, we provide an overview of the related works. In Section III, the system model under consideration is presented in detail. In Section IV, we formulate the continuous-time VNF scheduling problem as an MILP which is then transformed to a discrete-time formulation. In Section V, a single-agent RL approach is proposed to solve the delay-aware VNF scheduling problem. In Section VI, simulation results are presented to demonstrate the convergence and accuracy of the proposed approach. Section V concludes this paper.

II. Related Works

VNF chain embedding problems are studied extensively to either maximize the number of accommodated services or minimize the average long-run embedding cost under the assumption that each NFV node can support multiple network functions [12], [28]–[31]. When multiple VNFs are embedded onto an NFV node, a new and challenging research issue is how to properly schedule the embedded VNFs to minimize the overall completion time of packets processing for all service requests. Compared with the research on VNF chain embedding, the investigation of VNF scheduling is still in its infancy. As the first study on VNF scheduling, Riera *et al.* formulate a joint VNF assignment and scheduling problem as a job-shop problem (JSP) which can be solved in two separate stages [13]. In [17], Mijumbi *et al.* present three greedy algorithms and a tabu search meta-heuristic algorithm to solve a joint VNF chain embedding and VNF scheduling problem. Wang *et al.* present a one-hop greedy scheduling algorithm, which is part of the NFV-RA optimization framework [31]. In [18], Qu *et al.* formulate a joint VNF scheduling and traffic routing problem as an MILP, considering the link transmission delays for VNF scheduling. A genetic algorithm based heuristic algorithm is presented to obtain local optimal solutions, without QoS consideration. Although the existing heuristic algorithms can help to find local optimal solutions, the optimality gaps of the algorithms are difficult to validate, and the delay requirements are yet to be taken into account. In [19], Alameddine *et al.* study the deadline-aware VNF scheduling problem and present an MILP formulation for the joint problem of VNF chain embedding, traffic routing, and VNF scheduling based on a discrete-time model. To address the MILP, a tabu search-based heuristic algorithm is provided to obtain the near-optimal solutions. In [20], Pham *et al.* present a matching game based approach to address deadline-aware VNF scheduling.

Recently, the RL approach has been applied in the deployment of radio access networks (RANs) to improve energy efficiency [32] and optimize resource management [33]. Also, some research works exist in the literature that apply the RL approach to solving combinatorial optimization problems [21], [22], including standard JSP [23]–[25]. Specifically, in [23]–[25], various multi-agent RL algorithms have been presented to solve the JSP problem in different scenarios, where a centralized control of network states is hard to be instantiated. The job scheduling problem has been formulated as a decentralized MDP with changing action sets, and then been solved in a distributed manner with inter-agent coordinations. However, the optimal scheduling results may not be achieved with only local network information, as the restrictions imposed on the learning agents degrade the accuracy of the solutions [26]. Also, our objective in this work is to minimize the overall makespan while satisfying the delay constraints. Both the makespan evaluation and the delay constraints verification require the status information of the VNFs in the whole system. For decentralized learning, each agent only has a local view of the system and thus may not optimize the makespan with satisfied delay constraints well. Introducing inter-agent coordination to exchange necessary information among all the agents can introduce some overheads into the learning process. To address this issue, in the following, we leverage the SDN/NFV-enabled network architecture, where global network state information is collected to enhance the performance of VNF scheduling. By introducing a single scheduling agent placed in the SDN control module, the agent is able to make network-wide scheduling decisions and learn the optimal VNF scheduling policy. Moreover, a specific reward function is designed to incorporate the E2E delay requirements of diverse services into the VNF scheduling process.

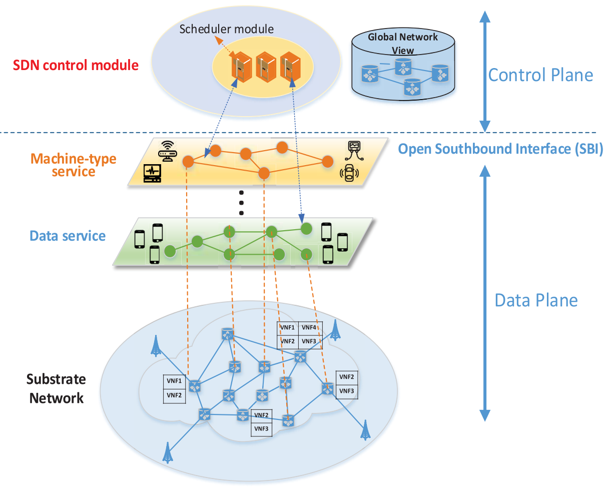


Fig. 1. An SDN/NFV-enabled network architecture.

III. System Model

Consider an SDN/NFV-enabled network architecture as shown in Fig. 1, where the control plane is decoupled from the data plane and migrated to a centralized SDN control module. The SDN controller is logically centralized and can be physically deployed in a decentralized way, in which case some information change between different local SDN controllers is required. The control module orchestrates the placement of VNFs from different VNF chains and configures traffic routing between consecutive VNFs for load balancing in the data plane. Through open southbound Interface (SBI) between the control module and the substrate network, the state information from the underlying substrate network can be collected. Multiple types of services, e.g., machine-type services with stringent E2E delay requirements and data services with non-stringent delay requirements, are considered and supported by different VNF chains. These services manifest different levels of delay requirements [6]. In the SDN/NFV-enabled network architecture, service customization and isolation should be achieved by embedding different VNF chains onto the same physical substrate network. VNF scheduling is periodically performed by the VNF scheduler (as a sub-module integrated inside the SDN control module) as service requests arrive. All the arrived service requests are

TABLE I
SUMMARY OF IMPORTANT NOTATIONS AND DECISION VARIABLES

Notation	Description
$\mathcal{R}$	set of network services
S_i	ith network service
D_i	processing time deadline of S_i for one packet batch
$\mathcal{F}$	set of VNFs in all the services
$\mathcal{F}_i$	set of VNFs in S_i
f_{ij}	jth VNF in S_i
$\mathcal{N}$	set of NFV nodes
n_k	kth NFV node
$\mathcal{I}_{ij}$	set of NFV nodes that can support VNF f_{ij}
ρ_{ijk}	packet batch processing time at $f_{ij} \in \mathcal{F}_i$ on node n_k
y_{ijk}	binary constant indicating if f_{ij} is assigned to node n_k
T	total number of time slots
τ	duration time of a time slot
$\mathcal{M}$	overall makespan
M	a big positive number
Decision variable	Description
t^s_{ij}	time when the packet batch of S_i starts processing at f_{ij}
x^t_{ik}	binary variable indicating whether or not S_i starts being supported on n_k at the tth time slot
$z^k_{ij,pq}$	binary variable indicating whether f_{ij} starts execution before f_{pq}, if f_{ij} and f_{pq} are both embedded onto n_k

scheduled simultaneously at the beginning of every scheduling period. Important parameters and variables of the system model are listed in Table I.

Substrate network – The substrate network under consideration is represented by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, where $\mathcal{V}$ denotes the set of physical nodes in the network and $\mathcal{L}$ represents the set of physical links connecting any pair of nodes. Define $\mathcal{N} (\subseteq \mathcal{V})$ as the set of NFV nodes, and NFV node $n_k \in \mathcal{N}$ ($k \in \mathbb{N}$, where $\mathbb{N} = \{1, 2, \dots, |\mathcal{N}|^1\}$) is capable of hosting multiple VNFs of different types. Denote the CPU processing capacity of n_k for a VNF f as $c_k(f)$.

VNF chain – A network service is composed of an ordered sequence of VNFs, referred to as a VNF chain. Suppose that there are $|\mathcal{R}|$ services, denoted by $S_1, S_2, \dots, S_{|\mathcal{R}|}$, and define $\mathbb{R} = \{1, 2, \dots, |\mathcal{R}|\}$. Denote the set of VNFs in S_i by $\mathcal{F}_i$. Let f_{ij} represent the jth VNF in $\mathcal{F}_i$, where $j \in \mathbb{F}_i$ and $\mathbb{F}_i = \{1, 2, \dots, |\mathcal{F}_i|\}$. Let $\mathcal{I}_{ij}$ be the set of NFV nodes that can support VNF f_{ij}.

Traffic model – Each NFV node is assumed to have a number of processing queues, where each packet from a specific service is buffered for processing at certain VNF embedded onto the NFV node. Packets of service S_i arrive at one of the processing queues at the first NFV node of an embedded VNF chain as a Poisson process with arrival rate λ_i (packet/s). When multiple VNFs are embedded at a common NFV node, it is required to determine the scheduling sequence of VNFs for packet processing since each VNF is scheduled for packet

[1] $|\cdot|$ represents the cardinality of a finite set.

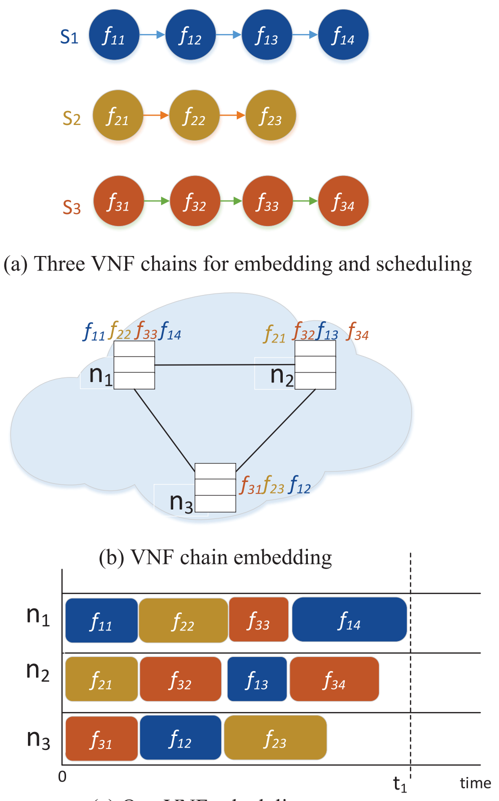

S1
f_{11}
f_{12}
f_{13}
f_{14}
S2
f_{21}
f_{22}
f_{23}
S3
f_{31}
f_{32}
f_{33}
f_{34}
(a) Three VNF chains for embedding and scheduling
f_{11} f_{22} f_{33} f_{14}
f_{21} f_{32} f_{13} f_{34}
n1
n2
n3
f_{31} f_{23} f_{12}
(b) VNF chain embedding
n1
n2
n3
0
t1
time
(c) One VNF scheduling pattern

Fig. 2. A simple example to illustrate the VNF scheduling process.

processing one at a time.[2] To reduce the switching overhead for VNF scheduling, we assume that a VNF is scheduled for packet processing only if its associated buffer occupancy is above a threshold B. The number of packets to be processed for one-time VNF scheduling is B, which is also called one *packet batch*. Thus, the processing time of one packet batch at f_{ij} on node n_k ($n_k \in \mathcal{I}_{ij}$) for service S_i is given by

$$\rho_{ijk} = B/c_k(f_{ij}). \tag{1}$$

VNF scheduling – VNF scheduling is performed cyclically. We define the *makespan* of one VNF scheduling cycle as the time duration that all embedded VNFs are scheduled once for processing a single packet batch. For simplicity, we assume that the processing queue occupancy of a subsequent VNF on an NFV node is above the threshold B once the previous VNF scheduling completes.[3] For the first scheduling cycle, we assume that VNFs at different NFV nodes start to be scheduled at the same time instant (i.e., the time instant 0 in Fig. 2 (c)). The scheduling starting and completion time for a packet batch of S_i traversing f_{ij} in a scheduling cycle are represented by t^s_{ij} and t^c_{ij}, respectively. Between t^c_{ij} and $t^s_{i,j+1}$, the packet batch waiting time can exist due to packet queueing on the NFV node where $f_{i,j+1}$ is embedded. The packet batch of S_i has to

[2] In a real system, the computing resources on each NFV node are indivisible. Each NFV node can only support at most one VNF at a time and all the computing resources are allocated to the VNF for packet batch processing [12], [18], [20], [34].

[3] Dynamic VNF scheduling by considering the buffer occupancy status will be studied in our future work.

be processed at a chain of VNFs in a predefined order, with the duration of $(t^s_{i|\mathcal{F}_i|} + \rho_{i|\mathcal{F}_i|k})$ for traversing all services' VNFs in a scheduling cycle. The scheduling results include two parts: 1) For service S_i, we determine the time when a packet batch starts being processed at f_{ij}; 2) For NFV node n_k, we determine the scheduling sequence of all embedded VNFs. Let D_i denote the maximum acceptable processing time for a packet batch of S_i passing through the VNF chain in one scheduling cycle. The VNF scheduling process is to jointly determine the scheduling sequence and the starting time for packet processing for the VNFs of all services to minimize the makespan. By conforming to an optimal scheduling sequence, each cycle of VNF scheduling proceeds repeatedly to reduce the overall processing delay of packets traversing each VNF chain.

A simple illustration of the VNF scheduling process is shown in Fig. 2. Suppose that there are three VNF chains of network services S_1, S_2, S_3 (labeled by different colours) to be scheduled at each scheduling cycle. On each NFV node, the VNF scheduling sequence has to be determined to minimize the makespan. The VNF chain embedding results are shown in Fig. 2(b). In Fig. 2(c), packet batches start traversing the VNFs of S_1, S_2, and S_3 at nodes n_1, n_2, and n_3, respectively, from time instant 0. Packet batch processing time at each scheduled VNF is also displayed by different colours. Time instant t_1 indicates the makespan of one VNF scheduling cycle.

IV. Problem Formulation

In this section, the delay-aware VNF scheduling problem is presented by a continuous-time formulation, in which the timings of the scheduling process are clearly described. For the continuous-time representation, additional binary variables are required to indicate the processing sequence of the scheduled batches, which complicates the mathematical formulation and the way of solving the problem. Hence, we transform the continuous-time formulation into a discrete-time interpretation in a time-slotted system, where the VNF scheduling decisions are made at the beginning of each time slot [35]. For both formulations, we incorporate delay constraints to achieve E2E delay guaranteed service provisioning.

A. Continuous-Time Formulation

Objective – The objective of the VNF scheduling problem (P1) is to minimize the makespan $\mathcal{M}$, which is expressed as

$$\min_{t^s_{ij}, z^k_{ij,pq}, z^k_{pq,ij}} \mathcal{M} \tag{2}$$

where the makespan $\mathcal{M}$ is the time duration of one VNF scheduling cycle for all services, given by

$$\mathcal{M} = \max_{i\in\mathbb{R}, j\in\mathbb{F}_i} \left\{ t^s_{ij} + \sum_{n_k\in\mathcal{I}_{ij}} y_{ijk}\rho_{ijk} \right\}. \tag{3}$$

In (3), binary parameter y_{ijk} is introduced, where $y_{ijk} = 1$ indicates f_{ij} is embedded at server n_k; otherwise, $y_{ijk} = 0$.

Constraints – The following constraints are imposed to guarantee the feasibility of the VNF scheduling:

1) If any two VNFs (e.g., f_{ij} and f_{pq}) are embedded onto the same NFV node, it is required that the packet batch processing at one of the two VNFs cannot start before the processing at the other one finishes [13]. To impose these constraints, we define an auxiliary binary variable $z^k_{ij,pq}$ as

$$z^k_{ij,pq} = \begin{cases} 1 & \text{if } f_{ij} \text{ starts packet batch processing before} \\ & f_{pq} \text{ on NFV node } n_k, \\ 0 & \text{otherwise.} \end{cases}$$

Then, the constraints ensuring that preemption is not allowed at any time on any NFV node are expressed as

$$t^s_{ij} + z^k_{ij,pq} \sum_{n_k\in\mathcal{N}} y_{ijk}\rho_{ijk} \le t^s_{pq} + z^k_{pq,ij} M, \tag{4}$$

$$t^s_{pq} + z^k_{pq,ij} \sum_{n_k\in\mathcal{N}} y_{pqk}\rho_{pqk} \le t^s_{ij} + z^k_{ij,pq} M, \tag{5}$$

$$z^k_{ij,pq} + z^k_{pq,ij} = 1 \tag{6}$$

where $p \in \mathbb{R}$, $q \in \mathbb{F}_p$, $p \neq i$ or $q \neq j$ and M is a big positive number [29]. Note that since M is large, constraint (4) is non-restrictive when $z^k_{ij,pq} = 0$ and $z^k_{pq,ij} = 1$ (i.e., when f_{pq} starts packet batch processing before f_{ij}), and constraint (5) is non-restrictive when $z^k_{ij,pq} = 1$ and $z^k_{pq,ij} = 0$ (i.e., when f_{pq} starts packet batch processing after f_{ij}).

2) The specified processing sequence of the VNFs in each service, $f_{i1} \to f_{i2}, \ldots, \to f_{i|\mathcal{F}_i|}$, $\forall i \in \mathbb{R}$, is enforced by [13]

$$t^s_{i,j+1} - t^s_{ij} \ge \sum_{n_k\in\mathcal{I}_{ij}} y_{ijk}\rho_{ijk}, \quad \forall i \in \mathbb{R}, j \in \mathbb{F}_i. \tag{7}$$

Constraint (7) guarantees that the processing of a packet batch at a subsequent VNF cannot start until the processing at its previous VNF is completed.

3) The duration time for a packet batch passing through the VNF chain of S_i in one scheduling cycle should satisfy

$$t^s_{i|\mathcal{F}_i|} + \sum_{n_k\in\mathcal{N}} y_{i|\mathcal{F}_i|k}\rho_{i|\mathcal{F}_i|k} \le D_i, \quad \forall i \in \mathbb{R}. \tag{8}$$

Therefore, the continuous time formulation for the delay-guaranteed VNF scheduling problem (P1) is presented as an MILP program, given by

$$\begin{aligned} \min_{t^s_{ij}, z^k_{ij,pq}, z^k_{pq,ij}} \quad & \mathcal{M} \\ \text{s.t.} \quad & (4)\text{–}(8), \\ & t^s_{ij} \ge 0, \\ & z^k_{ij,pq} \in \{0,1\}, \\ & z^k_{pq,ij} \in \{0,1\}. \end{aligned}$$

B. Discrete-Time Transformation

The continuous-time formulation presented above requires additional binary variables (i.e., $z^k_{ij,pq}$ and $z^k_{pq,ij}$) to indicate the packet batch processing sequence for scheduling VNFs. This makes both the mathematical formulation and the way of solving the problem more complicated. In addition, the difficulty of the algorithm design increases when continuous variables are involved. To overcome this problem, a discrete-time transformation of the continuous-time formulation is presented in this section.

Time is divided into T time slots of equal and fixed duration time τ [19]. It is assumed that a packet batch can only start processing at one VNF at the beginning of a certain time slot, and the processing time is an integer multiple of one time slot. We define binary variable x_{ik}^t to indicate the packet batch processing state of service i on NFV node n_k, where $x_{ik}^t = 1$ indicates the packet batch of service S_i starts being processed at the VNF embedded on NFV node n_k at (the beginning of) time slot t, and $x_{ik}^t = 0$ otherwise. The discrete time formulation for the VNF scheduling problem (P2) is presented as follows:

Objective – The objective is to minimize the makespan of packet batch processing for all services, given by

$$\min_{x_{ik}^t} \mathcal{M} \tag{9}$$

where

$$\mathcal{M} = \max_{i\in\mathbb{R}} \left\{ \sum_{k=1}^{|\mathcal{N}|} \sum_{t=1}^{T} x_{ik}^t y_{i|\mathcal{F}_i|k} \Big((t-1)\tau + \rho_{i|\mathcal{F}_i|k} \Big) \right\} \tag{10}$$

Constraints:

$$x_{ik}^t y_{ijk} + \sum_{i'\in\mathbb{R}, i'\neq i} x_{i'k}^{t'} \leq 1, \quad \forall i \in \mathbb{R},\ \forall j \in \mathbb{F}_i,\ \forall k \in \mathbb{N},$$
$$\forall t, t' \in [1, T], t \leq t' < t + \sum_{n_k\in\mathcal{I}_{ij}} y_{ijk}\rho_{ijk} \tag{11}$$

$$\sum_{k=1}^{|\mathcal{N}|} \sum_{t=1}^{T} x_{ik}^t \Big(y_{i(j+1)k} - y_{ijk} \Big)(t-1)\tau \geq \sum_{k=1}^{|\mathcal{N}|} \sum_{t=1}^{T} x_{ik}^t y_{ijk}\rho_{ijk},$$
$$\forall i \in \mathbb{R}, \forall j \in \mathbb{F}_i \setminus \{|\mathcal{F}_i|\} \tag{12}$$

$$\sum_{i=1}^{|\mathcal{R}|} x_{ik}^t \leq 1, \quad \forall t \in [1, T], \forall k \in \mathbb{N} \tag{13}$$

$$\sum_{k=1}^{|\mathcal{N}|} \sum_{t=1}^{T} x_{ik}^t y_{ijk} = 1, \quad \forall i \in \mathbb{R}, \forall j \in \mathbb{F}_i \tag{14}$$

$$\sum_{k=1}^{|\mathcal{N}|} \sum_{t=1}^{T} x_{ik}^t y_{i|\mathcal{F}_i|k} \Big((t-1)\tau + \rho_{i|\mathcal{F}_i|k} \Big) \leq D_i, \quad \forall i \in \mathbb{R} \tag{15}$$

where T is the total number of time slots, and is a large number to ensure the completion of one packet batch processing for all services. $[1, T]$ represents the set of integers between 1 and T. Constraint (11) indicates that packet batch processing is conducted for only one VNF at a time on an NFV node [19]; Constraint (12) indicates that a packet batch processing at a VNF cannot start until its previous VNF completes the processing; Constraint (13) guarantees that at any time slot t, the packet batch of at most one service is processed by a NFV node; Constraint (14) ensures that the packet batch processing at a VNF will not be repetitively conducted; Constraint (15) is the processing time deadline constraint of one packet batch for each service.

From (2)-(8), it is observed that Problem (P1) involves continuous variables t_{ij}^s as well as binary variables $z_{ij,pq}^k$ and $z_{pq,ij}^k$. Therefore, (P1) is an MILP. The objective is to minimize the overall makespan while respecting the precedence relations between VNFs and satisfying the delay requirements of the services. In fact, the formulation of (P1) falls into the category of job-shop scheduling problem with deadlines (JSSD).

Remark 1: JSSD is NP-hard.

Proof: See the Appendix for the proof. ■

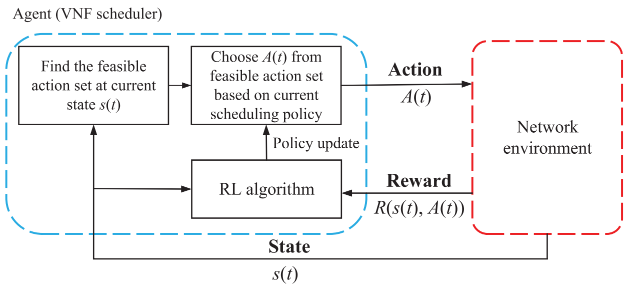

Agent (VNF scheduler)
Find the feasible action set at current state $s(t)$
Choose $A(t)$ from feasible action set based on current scheduling policy
Policy update
RL algorithm
Action
$A(t)$
Reward
$R(s(t), A(t))$
Network environment
State
$s(t)$

Fig. 3. Reinforcement learning framework for VNF scheduling.

V. A Single Agent Q-Learning Algorithm

In this section, we reformulate the VNF scheduling problem (P2) in Section IV as an MDP problem with variable action set. An MDP is typically composed of five parts, denoted by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$, where $\mathcal{S}$ represents a set of system states, $\mathcal{A}$ a set of actions, $\mathcal{P}$ the state transition probabilities, R a reward function, and γ the discount factor. In what follows, we provide their specific representations in the context of VNF scheduling, based on which a single-agent RL approach is proposed to allow the agent to learn the optimal decision policy for the MDP. Fig. 3 shows the overall reinforcement learning framework for VNF scheduling. At each time slot t, the agent (i.e., VNF scheduler) first finds the feasible action set based on the current system state $s(t)$, then chooses an action $A(t)$ from the feasible action set according to the current scheduling policy. The reward obtained from the network environment, $R(s(t), A(t))$, is fed back to the agent for updating the scheduling policy through the RL algorithm.

A. System State

The system state is captured at the beginning of each time slot $t\ (\in [1, T])$, and is defined as $s(t) = [\boldsymbol{M}(t), \boldsymbol{F}(t)]$, where $\boldsymbol{M}(t) = [m_1(t), m_2(t), \ldots, m_{|\mathcal{N}|}(t)]$ denotes the states of NFV nodes and $\boldsymbol{F}(t) = [\xi_1(t), \xi_2(t), \ldots, \xi_{|\mathcal{F}|}(t)]$ represents VNF states. $\mathcal{F}$ is the set of VNFs to be scheduled for all network services. For notation convenience, we sort all VNFs in $\mathcal{F}$ according to the their processing times to have a one-to-one mapping between f_l and f_{ij}, where f_l represents the lth VNF in the sorted VNFs. At time slot t, the definitions of $m_k(t)$ and $\xi_l(t)$ with $k \in \mathbb{N}$ and $l \in \mathbb{F}$ are given by

$$m_k(t) = \begin{cases} 0, & \text{if } n_k \text{ is not processing packets,} \\ 1, & \text{if } n_k \text{ is processing packets,} \end{cases} \tag{16}$$

$$\xi_l(t) = \begin{cases} 0, & \text{if } f_l \text{ is waiting to be traversed,} \\ 1, & \text{if } f_l \text{ is being traversed,} \\ 2, & \text{if } f_l \text{ has been traversed.} \end{cases} \tag{17}$$

Note that a one-to-one mapping also exists between $\xi_l(t)$ and $\xi_{i,j}(t)$. At the initial time slot, all the NFV nodes are idle and the VNFs in all the services are waiting to be traversed. Let the initial and completion states of one VNF scheduling process be s_{ini} and s_{ter}, respectively. According to (16) and (17), we have $s_{ini} = (0, \ldots, 0, 0, \ldots, 0)$ and $s_{ter} = (0, \ldots, 0, 2, \ldots, 2)$.

B. Action and Variable Action Set

In our problem, due to the dependency of VNFs, not all the actions are feasible for all the states. Therefore, it is useful to introduce an additional mapping which assigns the set of feasible/admissible actions to each state [36]. An action of the VNF scheduler (i.e., the agent in RL) at state *s*(*t*), *A*(*t*), indicates the VNFs chosen to be traversed at state *s*(*t*) on all the NFV nodes, and is represented by $A(t) = [a_1(t), a_2(t), \ldots, a_{|\mathcal{N}|}(t)]$, where $a_k(s_t)$ is the action to take at s_t on NFV node n_k. For example, *A*(*t*) = (1, 5, 2, 7) indicates that at state *s*(*t*), the VNFs chosen to be traversed on n_1, n_2, n_3, and n_4 are f_1, f_5, f_2, and f_7, respectively. The feasible action set of the agent at state *s*(*t*) is denoted by $\mathcal{A}(s_t) = \mathcal{A}_1(s_t) \otimes \mathcal{A}_2(s_t) \otimes \cdots \otimes \mathcal{A}_{|\mathcal{N}|}(s_t)$, where $\mathcal{A}_k(s_t)$ is the feasible action set of NFV node n_k at state *s*(*t*), and $\otimes$ denotes the Cartesian product. $\mathcal{A}_k(s_t)$ contains the indices of all the VNFs that can be traversed on NFV node n_k at s_t. If n_k does not have any VNF waiting to be traversed at state s_t, we set $\mathcal{A}_k(s_t) = \{0\}$ and $a_k(s_t) = 0$. We refer to them as a null feasible action set and a null action, respectively. A system state with a non-null feasible action set is called a *decision-making state*. Note that since the packet processing time at different VNFs can be different, the length of the time period between two decision-making states is not fixed. For notation convenience, let $\mathcal{A}_{null}(s_t) = \{(0, 0, \ldots, 0)\}$.

The feasible action set $\mathcal{A}(s_t)$ depends on each state, and can be obtained without observing the environment. Therefore, in this paper, the feasible action set is finite, state-dependent, and fixed for a given state. This state-dependent feasible action set imposes additional challenge for designing the learning algorithm. Since the feasible actions are continually changing as the state changes, a specific algorithm has to be devised to find all the feasible actions for the VNF scheduler at a given system state. Then, the VNF scheduler can choose an action from the feasible action set before taking an action. Note that the feasible action set remains constant for each state, so it only needs to be found once before learning. The details for finding the feasible action set for the agent at state s_t is shown in Algorithm 1. At the beginning of the algorithm, all the idle servers are found and added to the set $\mathcal{N}_{idle}(s_t)$. For busy servers, let their feasible action set contain a null action only (Line 1 - Line 5). Then, the algorithm finds all the VNFs currently waiting for being traversed on each idle server (Line 7) and adds feasible actions (i.e., VNF indices) to the action set $\mathcal{A}_k(s_t)$ with the consideration of VNF dependency (Line 8 - Line 14). Finally, the feasible action set for the VNF scheduler is determined by the Cartesian product of the feasible action set of all the servers (Line 15).

Algorithm 1: Algorithm for Finding the Feasible/Admissible Action Set at s_t

```
Input: Current system state $s_t$
Output: Feasible action set $\mathcal{A}(s_t)$
1  for all $n_k \in \mathcal{N}$ do
2      if $n_k$ is idle at time slot t then
3          Add $n_k$ to $\mathcal{N}_{idle}(s_t)$;
4      else
5          Set $\mathcal{A}_k(s_t) \leftarrow \{0\}$;
6  for all $n_k \in \mathcal{N}_{idle}(s_t)$ do
7      $\mathcal{F}_k(s_t) \leftarrow findWaitingFunctions(n_k, s_t)$;
8      if $\mathcal{F}_k(s_t)$ is empty then
9          Set $\mathcal{A}_k(s_t) \leftarrow \{0\}$;
10     else
11         Add $\{0\}$ to $\mathcal{A}_k(s_t)$;
12         for all $f_{ij} \in \mathcal{F}_k(s_t)$ do
13             if $j = 1$ or $\xi_{i,j-1}(t) = 2$ then
14                 Add $f_{ij}$ to $\mathcal{A}_k(s_t)$;
15 Set $\mathcal{A}(s_t) \leftarrow \mathcal{A}_1(s_t) \otimes \mathcal{A}_2(s_t) \otimes \cdots \otimes \mathcal{A}_{|\mathcal{N}|}(s_t)$.
```

Complexity Analysis for Algorithm 1: For a given system state s_t, the task of finding its feasible action set contains two steps. The first step is to find all the idle NFV nodes and add them into $\mathcal{N}_{idle}(s_t)$, which has a computational complexity of $O(|\mathcal{N}|)$. Then, for each idle NFV node $n_k \in \mathcal{N}_{idle}(s_t)$, we find all the waiting functions on it (denoted by $\mathcal{F}_k(s_t)$) and add all the feasible ones into $\mathcal{A}_k(s_t)$, whose complexity is $O(|\mathcal{N}_{idle}(s_t)| \cdot |\mathcal{F}_k(s_t)|)$. Therefore, the total computational complexity of finding feasible action set is $O(\max(|\mathcal{N}|, |\mathcal{N}_{idle}(s_t)| \cdot |\mathcal{F}_k(t)|))$, upper-bounded by $O(|\mathcal{N}| \cdot |\mathcal{F}|)$. Note that Algorithm 1 is performed only once for a given state during the whole learning process. Once the admissible/feasible action set is found for a given state, the mapping between feasible action set and state does not need to be found again for that state.

C. System State Transition Rules

In this section, we describe the system state transition rules between two consecutive time slots. At time slot *t* = 1, the system is in its initial state, i.e., $s(1) = s_{ini}$. Then, the system state is updated at the beginning of each time slot until it reaches the completion state s_{ter}. The state transitions include the transitions of server state ***M***(*t*) and function state ***F***(*t*). The state transitions for ***M***(*t*) are given by

$$m_k(t+1) = \begin{cases} 0, & \text{if } a_k(t) = 0, m_k(t) = 1, \theta_k(t) = 1 \\ 1, & \text{if } a_k(t) \neq 0, m_k(t) = 0, \theta_k(t) > 1 \\ m_k(t), & \text{otherwise} \end{cases} \tag{18}$$

where $\theta_k(t)$ denotes the remaining packet batch processing time at the VNF being traversed on server n_k at time slot *t*. In (18), the first condition indicates that the state of n_k transits from the packet batch processing state into the idle state if n_k takes a null action, and the remaining packet batch processing time at the VNF is exactly one time slot; The second condition indicates that the state of n_k transits from the idle state to the packet batch processing state if the node takes a non-null action at time slot *t* and the packet batch processing time at the VNF chosen to be traversed is more than one time slot. On the other hand, the state transitions for $\boldsymbol{F}(t)$ are given by

$$\xi_l(t+1) = \begin{cases} 1, & \text{if } \xi_l(t)=0, l \in A(t), \tau_l(t) > 1 \\ 2, & \text{if } \xi_l(t)=0, l \in A(t), \tau_l(t) = 1 \\ 2, & \text{if } \xi_l(t)=1, \tau_l(t)=1 \\ \xi_l(t), & \text{otherwise} \end{cases} \quad (19)$$

where $\tau_l(t)$ denotes the remaining packet batch processing time at f_l at time slot *t*. In (19), the first condition indicates that the state of f_l transits from waiting for being scheduled to being traversed if the agent starts the packet batch processing at time slot *t* and the corresponding packet batch processing time is greater than one time slot. The second condition indicates that the state of f_l transits from waiting for being scheduled to scheduling completion if the agent starts packet batch processing at time slot *t* and the function traversing time is exactly one time slot. The third condition indicates that ξ_l transits from being traversed to scheduling completion if the remaining packet batch processing time at f_l is one time slot at time slot *t*.

D. Reward Function

The objective of VNF scheduling is to minimize the overall makespan $\mathcal{M}$, while satisfying the delay requirements for different services. When an RL approach is applied to achieve delay-aware VNF scheduling, it is required that the accumulated reward (which is the feedback to the agent during the learning process) is consistent with the objective of VNF scheduling. However, the makespan and the completion time instants of all services are not accessible until the system reaches the completion state. Therefore, instead of providing the agent with an instant reward after an action is taken at each time slot, we calculate the accumulated reward for continuous state-action pairs before an episode (i.e., a sequence of agent-environment interactions between initial and completion states) ends. In this study, we follow the reward feedback mechanism used/discussed in [37]–[39], where the reward for a state-action pair is received by the end of an episode. Although this mechanism may slow down the overall learning process, other advanced RL techniques such as reward shaping and parallel computation can be employed to speed up the overall learning process. The accumulated reward is then fed back to the agent to help improve its VNF scheduling policy. Specifically, we design an accumulated reward for a series of state-action pairs in an episode, given by

$$R(s_t, A_t) = c_0/\mathcal{M} + \sum_{i=1}^{|\mathcal{R}|} c_i \delta_i, \ \forall (s_t, A_t) \in \Omega \quad (20)$$

where $\mathcal{M}$ is the makespan for current episode, Ω is a set containing all the state-action pairs in the episode, and δ_i is a binary variable indicating if the delay requirement of S_i is satisfied ($\delta_i = 1$) or not ($\delta_i = 0$). c_0 and c_i are weighting coefficients reflecting the rewards obtained from minimizing the makespan and satisfying delay requirements, respectively.

If the packet batch processing of service S_i is finished before its deadline, the scheduling agent obtains an additional reward c_i. In this way, the scheduling agent tries to maximize the accumulated reward by scheduling the services to be completed before their deadlines. Setting $c_0 = 1$ and $c_i = 0$ implies that one only considers to minimize the overall makespan and ignores the delay requirements of the services, while setting $c_0 = 0$ and $c_i = 1$ means that one aims to have all the delay requirements satisfied no matter how long the makespan is. By setting proper values for c_0 and c_i, we minimize the makespan while having all the delay requirements satisfied. Note that the value of c_i also reflects the priority of services, and high priority services should have large c_i. For delay non-sensitive services, c_i should be set to a small value (or even zero), since the additional reward from satisfying the delay requirement can be obtained. For delay-sensitive services, c_i should be set as a large value to satisfy the delay requirement.

E. Q-Learning Algorithm for Optimal VNF Scheduling

Given the system states, actions, and reward functions, we develop an RL approach for the scheduling agent to learn the optimal scheduling policy π^* that maximizes the accumulated reward over time, given by

$$\pi^* = \max_{\pi} \sum_{t \in [1,T]; \ \mathcal{A}(s_t) \neq \mathcal{A}_{null}} R(s_t, A_t). \quad (21)$$

Q-learning [40] is adopted to allow the VNF scheduler to learn the optimal scheduling policy through continuously interacting with the system. During the learning process, a scheduling policy table (also called *Q*-table) is maintained and the entries (*Q*-values) in the *Q*-table are updated iteratively by [40]

$$\begin{aligned} Q(s_t, A_t) &= (1-\alpha) Q(s_t, A_t) \\ &\quad + \alpha \left[R(s_t, A_t) + \gamma \max_{A_{t+1}} Q(s_{t+1}, A_{t+1}) \right] \end{aligned} \quad (22)$$

where α is the learning rate and γ is the discount factor. Both α and γ are real numbers set between 0 and 1. The *Q*-value at time slot *t*, $Q(s_t, A_t)$, represents the expected reward for the state-action pair (s_t, A_t), and thus can be interpreted as the probability of the agent taking action A_t at sate s_t. It is demonstrated in [42] that *Q*-learning can be used to achieve an optimal policy for discounted reward problems. Suppose that the *Q*-table converges to its optimum Q^* after sufficiently large number of episodes, then the optimal policy π^* can be obtained based on a greedy exploration [43]. That is,

$$\pi^* = \arg\max_{A(t)} Q^*(s_t, A_t). \quad (23)$$

Since the VNF scheduling problem is transformed as an MDP with variable action set, the agent can take null actions at some time instants. Therefore, the general *Q*-value iterative equation in (22) cannot be directly applied to our problem. To make it adapt to the varying time period

Algorithm 2: *Q*-Learning Algorithm to Find the Optimal Scheduling Policy

```
1  Initialization:
2  Initialize episode counter n as 0;
3  Initialize $n_{max}, \alpha, \epsilon, \gamma$;
4  Set $t \leftarrow 0$;
5  while $n < n_{max}$ do
6      Set $s_t \leftarrow s_{ini}$;
7      Set $\Omega_n \leftarrow \emptyset$;
8      while $s_t \neq s_{ter}$ do
9          $A_t \leftarrow chooseAction(s_t, Q, t)$;
10         Add $(s_t, A_t)$ to $\Omega_n$;
11         $s_{t+1} \leftarrow TakeAction(s_t, A_t, t)$;
12         while True do
13             $t \leftarrow t + 1$;
14             if $s_t = s_{ter}$ then
15                 break;
16             $\mathcal{A}(s_t) \leftarrow findFeasibleActionSet(s_t, t)$;
17             if $\mathcal{A}(s_t) \neq \mathcal{A}_{null}$ then
18                 break;
19             else
20                 $s_{t+1} \leftarrow NullAction(s_t, t)$;
21     $n \leftarrow n + 1$;
22     for all $(s_t, A_t) \in \Omega_n$ do
23         Calculate $R(s_t, A_t)$ according to (20);
24         $Q(s_t, A_t) \leftarrow (1-\alpha)Q(s_t, A_t) + \alpha(R(s_t, A_t) + \gamma \max_{A_{t+\Delta t}} Q(s_{t+\Delta t}, A_{t+\Delta t}))$;
```

between two decision-making states, we modify (22) as follows

$$Q(s_t, A_t) = (1-\alpha)Q(s_t, A_t) + \alpha\left[R(s_t, A_t) + \gamma \max_{A_{t+\Delta t}} Q(s_{t+\Delta t}, A_{t+\Delta t})\right] \quad (24)$$

where $t + \Delta t$ denotes the next time slot when the decision-making state occurs, and Δt depends on both the system state and the packet batch processing time at the VNFs being traversed at time slot *t*. Note that the *Q*-values are only updated at each decision-making state. Between any two consecutive decision-making states, Δt is same for all the NFV nodes. In the implementation of large networks, the logically centralized VNF scheduler can be physically decentralized. In such a circumstance, if we apply *Q*-learning centrally (i.e., maintain the *Q*-table in a single server), then some information exchange among different servers will be required.

The detailed *Q*-learning algorithm to determine the optimal scheduling policy is given in Algorithm 2, where *n* denotes the episode counter; n_{max} denotes the maximum number of episodes for *Q*-learning; and Ω_n represents the set containing all the state-action pairs in the *n*th episode. At the beginning of each episode, the system state is initialized as s_{ini} and Ω_n is empty (Line 6 - Line 7). In each episode, the agent takes an action by using the ϵ-greedy algorithm at the beginning of each

Algorithm 3: Algorithm for Choosing Action at s_t

```
   Input: System state $s_t$, Q-table
   Output: The action to take $A_t$
1  $\mathcal{A}(s_t) \leftarrow findFeasibleActionSet(s_t)$;
2  $\mathcal{N}_{idle}(s_t) \leftarrow findIdleServers(s_t)$;
3  for all $n_k \notin \mathcal{N}_{idle}(s_t)$ do
4      Set $a_k(t) \leftarrow 0$;
5  for all $n_k \in \mathcal{N}_{idle}(s_t)$ do
6      if $\mathcal{A}_k(t) = \{0\}$ then
7          Set $a_k(t) \leftarrow 0$;
8      else
9          With probability $\epsilon$ to choose a random action
           $a_k(t)$ from $\mathcal{A}_k(s_t)$;
10         Otherwise, choose
           $a_k(t) = arg \max_{a_k(t)} Q(s_t, A_t)$;
11 Set $A_t \leftarrow [a_1(t), a_2(t), \ldots, a_{|\mathcal{N}|}(t)]$;
```

time slot (Line 9 - Line 11). The detailed algorithm for the agent to take an action at each state is given in Algorithm 3. If an action is a null action, the system state keeps transiting over time until the next decision-making state occurs (Line 12 - Line 20). Whenever the agent takes a non-null action, the state-action pair (s_t, A_t) is added to Ω_n (Line 10). When an episode ends, we calculate the accumulated reward for all the state-action pairs in this episode according to (20) (Line 23) and update *Q*-values according to (24) (Line 24).

Complexity Analysis: In each episode, the agent first chooses an action at each state using Algorithm 3 and then execute it. The worst-case complexity of this operation is $O(|\mathcal{N}||\mathcal{F}|)$. The state evolves until the completion state s_{ter} is reached. Let *m* represent the maximum number of steps in an episode. Then, the complexity of running an episode is $O(m \cdot |\mathcal{N}| \cdot |\mathcal{F}|)$. Finally, the worst-case running time of updating the *Q*-table for the state-action pairs in each episode is *O*(*m*). Therefore, the total complexity in each episode of RL is upper-bounded by $O(|\mathcal{S}| \cdot |\mathcal{N}| \cdot |\mathcal{F}|)$. The total number of episodes required by RL is n_{max}. So the overall complexity of Algorithm 2 is upper-bounded by $O(n_{max}|\mathcal{S}| \cdot |\mathcal{N}| \cdot |\mathcal{F}|)$.

Convergence Analysis: As shown in Algorithm 1, we introduce an additional mapping which assigns the set of feasible/admissible actions to each state. Each state *s*(*t*) consists of the states of NFV nodes and the states of VNFs. The feasible action set $\mathcal{A}(s_t)$ depends on each state and can be obtained without observing the environment. Therefore, $\mathcal{A}(s_t)$ is finite, state-dependent, and fixed for a given state. The MDP under consideration has stable state-action pairs. When the state and action spaces are finite and stable, various proofs exist that the tabular *Q*-learning does converge to the optimal *Q*-function $Q^*(s_t, A_t)$, under very mild conditions [40]–[43], supposed that all state-action pairs are infinitely often visited.

VI. Performance Evaluation

In this section, we first verify the convergence of the proposed RL approach then compare its performance with other benchmark algorithms. In the following simulations,

TABLE II
PARAMETER RANGES IN SIMULATION FOR DIFFERENT NETWORK SCALES

Parameter	Small-scale	Medium-scale	Large-scale
$\|\mathcal{N}\|$	[1,5]	[6,10]	[10,20]
$\|\mathcal{R}\|$	[1,5]	[6,15]	[16,30]
$\|\mathcal{F}_i\|$	[2,5]	[2,5]	[2,5]
ρ_{ijk}	[1,5]	[1,5]	[1,5]

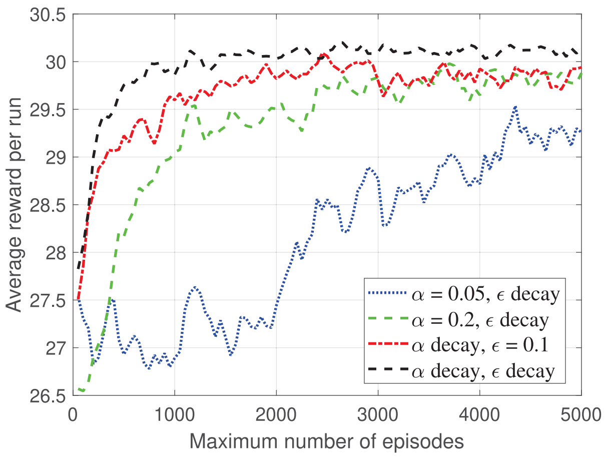


Fig. 4. Convergence of the proposed RL approach over the medium-scale network.

three different network scales (small, medium and large) are considered, which reflects different numbers of supported NFV nodes, network services, and VNFs. The parameter ranges in the simulation of the three network scales are listed in Table II. For all the three network scales, the VNFs are randomly embedded onto the NFV nodes. The number of VNFs for each network service is a random integer between 2 and 5, while the packet batch processing time (in terms of number of time slots) at a VNF is a randomly selected integer from 1 to 5. Note that the size of a problem instance is sampled from parameters in Table II and remains constant across episodes during the whole learning process.

A. *Convergence of the Proposed RL Approach*

We first use a medium-scale network with 8 NFV nodes and 10 services to verify the convergence of the proposed RL approach. For all the simulations presented in this section, we consider all the services as delay non-sensitive ones except for S_3, S_6, and S_{10}. Accordingly, we set the coefficients in the reward function as $c_0 = 600$, $c_3 = 1.0$, $c_6 = 1.0$, $c_{10} = 1.0$, and $c_i = 0$ ($i = 1, 2, 4, 5, 7, 8, 9$). The corresponding delay constraints are specified as $D_3 = 18$, $D_6 = 20$, and $D_{10} = 21$ (all in time units).

1) Impact of Learning Rate (α) and Exploration Rate (ϵ): Fig. 4 illustrates the convergence of the learning process with different parameters. The discount factor γ is set to be 0.8. We consider a constant learning rate ($\alpha = 0.05, 0.2$) and a decayed one (decaying from 0.8 to 0.1 with a decay rate of 0.999) and compare their performance. From the figure, we can see that a decayed learning rate helps the RL approach converges faster than a constant one does. This is because a decayed α allows the learning algorithm to take bigger steps during the initial phase to have a fast learning, but to take smaller steps as learning approaches convergence.

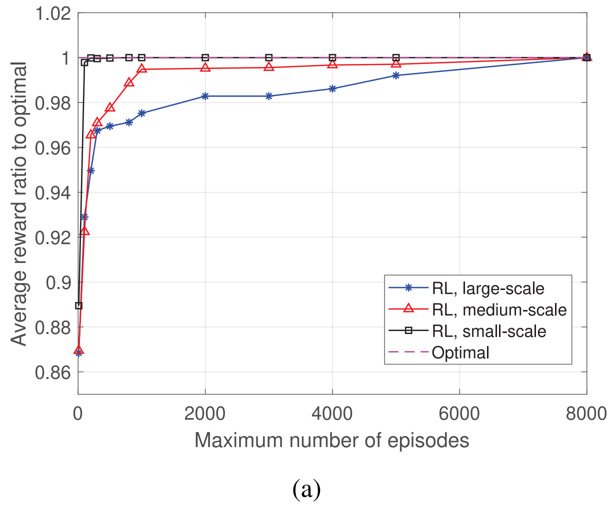
On the other hand, the choice of the value for ϵ reflects the trade-off between exploration and exploitation [40]. Two cases of ϵ values are considered, i.e., $\epsilon = 0.1$ and ϵ decays from 1.0 to 0.1 with a decay rate 0.99 (referred to as "ϵ decay"). It is observed that the RL approach with a decayed ϵ has a higher convergence rate. This is because a decayed ϵ has a higher chance do exploration at the beginning but decreases the possibility dedicated for exploration as time goes by. Fig. 4 also shows that the proposed RL approach converges after about 1000 episodes with the optimal set of learning parameters among the four cases presented.

2) Impact of Network Scale: Next, we demonstrate the convergence of the proposed RL algorithm over all the three network scales, as shown in Fig. 5. The selected small-scale network is with 4 NFV nodes and 5 network services, while the selected large-scale network is with 15 NFV nodes and 20 network services. For the small-scale network, we set the coefficients in the reward function as $c_0 = 350$, $c_1 = 1.0$, $c_2 = 0$, $c_3 = 1.0$, $c_4 = 0$, and $c_5 = 1.0$. The delay constraints are specified as $D_1 = 18$, $D_3 = 17$, and $D_5 = 8$ (all in time units). For the large-scale network, we set $c_0 = 2000$, $c_9 = 1.0$, $c_{15} = 1.0$, $c_{17} = 1.0$, and $c_i = 0$ ($i = 1 \ldots 20, i \neq 9, 15, 17$). The delay constraints are specified as $D_9 = 17$, $D_{15} = 41$, and $D_{17} = 20$ (all in time units).

For each network scale, we first tune the RL parameters to approach the optimum, and then apply to different maximum number of episodes n_{max} to observe a convergence. For each value of n_{max}, we run the RL approach 50 times and calculate the average reward and makespan per run for each network scale. These rewards and makespans are then used to find the ratios to optimality of different learning configurations. The optimal solutions are found by solving the ILP model given by ($P2$) using Gurobi solver. Fig. 5(a) and Fig. 5(b) show the convergence of the RL algorithm in terms of average reward per run and average makespan per run, respectively. As can be seen from the two figures, the proposed RL approach is able to produce the optimal solution upon convergence after a certain number of learning episodes. Also, the convergence rate of the RL approach decreases as the size of problem instance expands. Specifically, RL converges to the optimal solution after around 400, 1000, and 2000 episodes for the small, medium, and large problem instances, respectively. The reason for this is that a large problem instance has a bigger size of solution space, and thus the RL agent needs more episodes to explore the optimal policy.

B. *Verification of Delay-Guaranteed VNF Scheduling Using the Proposed RL Approach*

We focus on the medium-scale network to verify the proposed RL approach in guaranteeing the delay constraints. To this end, we consider two types of services, i.e., delay sensitive ones and delay non-sensitive ones, and two cases of service type assignments. In Case 1, S_3, S_6, and S_{10} are considered to be delay sensitive services, and all the



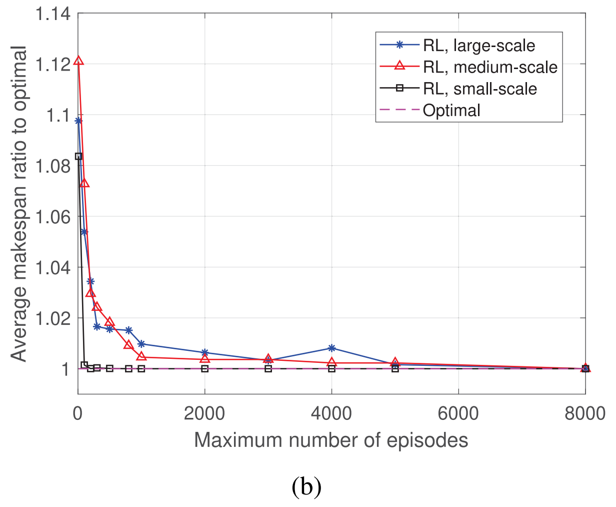
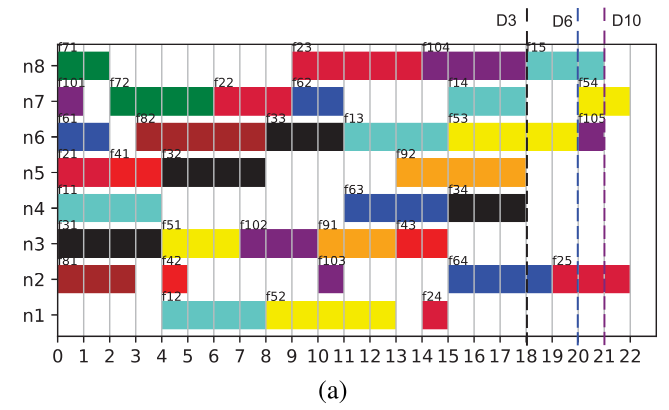


Fig. 5. Convergence of the RL algorithm over the small-scale, medium-scale and large-scale networks: (a) Average reward ratio to optimal, and (b) Average makespan ratio to optimal.



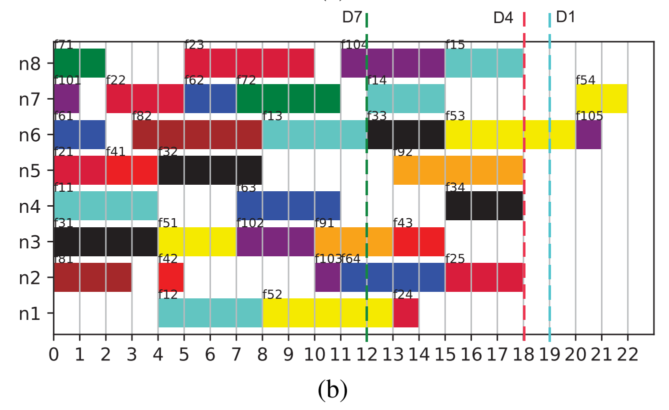


Fig. 6. Verifying delay-guaranteed VNF scheduling using the proposed RL approach over the medium-scale network: (a) Case 1 with $c_3 = 1.0$, $c_6 = 1.0$, $c_{10} = 1.0$, and (b) Case 2 with $c_1 = 1.0$, $c_4 = 1.0$, $c_7 = 1.0$.

remaining services are considered to be delay non-sensitive services. Accordingly, the coefficients in the reward function are set as $c_3 = 1.0$, $c_6 = 1.0$, $c_{10} = 1.0$, and $c_i = 0$ ($i = 1, 2, 4, 5, 7, 8, 9$). The corresponding delay constraints are specified as $D_3 = 18$, $D_6 = 20$, and $D_{10} = 21$ (all in time units). In Case 2, however, S_1, S_4, and S_7 are considered to be delay sensitive services, so we set $c_1 = 1.0$, $c_4 = 1.0$, $c_7 = 1.0$, and $c_i = 0$ ($i = 2, 3, 5, 6, 8, 9, 10$). The corresponding delay constraints are specified as $D_1 = 19$, $D_4 = 18$, and $D_7 = 12$ (all in time units). For both cases, we set $c_0 = 600$, and run the RL approach for 1000 episodes (i.e., $n_{max} = 1000$) to allow a guaranteed convergence.

The detailed VNF scheduling results for the two cases are presented in Fig. 6, where the interval between two vertical lines represents a time slot, the bricks in the same color represent the VNFs in a network service, and the vertical dashed lines represent the delay constraints of the delay sensitive services. As can be seen from the figure, the scheduling outcomes of the two cases have the same makespan but the detailed scheduling sequence of the VNFs are different. In both cases, the proposed RL approach is able guarantee the delay constraints of delay sensitive services, while achieving the optimal makespan. This demonstrates that the proposed approach achieves delay-aware VNF scheduling effectively and can support services with different priorities.

C. Running Time Comparison

We first compare the running time between the proposed RL approach and the MILP model to illustrate the time efficiency of the proposed approach. For the MILP formulation, we employ the Gurobi optimization solver to determine the optimal solution. For the RL approach, we tune its parameters to approach the optimal values. Table III compares the running time, number of episodes, and the corresponding average reward of the proposed RL approach with that of the MILP model. Both methods are run on a Intel i5-7200U CPU @2.5GHz. We can find that, for the medium-scale network, solving the MILP model using Gurobi solver takes about 5.4 s with a reward of 30.273 (calculated using the same reward function as RL). In comparison, the solutions from the proposed RL approach after 500 (or 1000) episodes are only 2.26% (or 0.52%) worse than the optimality, while the running times are both lower than that of the MILP model. As for the large-scale network, the solution from the RL approach with 2000 episodes already has a very small gap (1.71%) to the optimality, while the running time is almost half of that of the MILP model. It is also noticed that the advantage of the RL approach becomes greater as the problem instance size

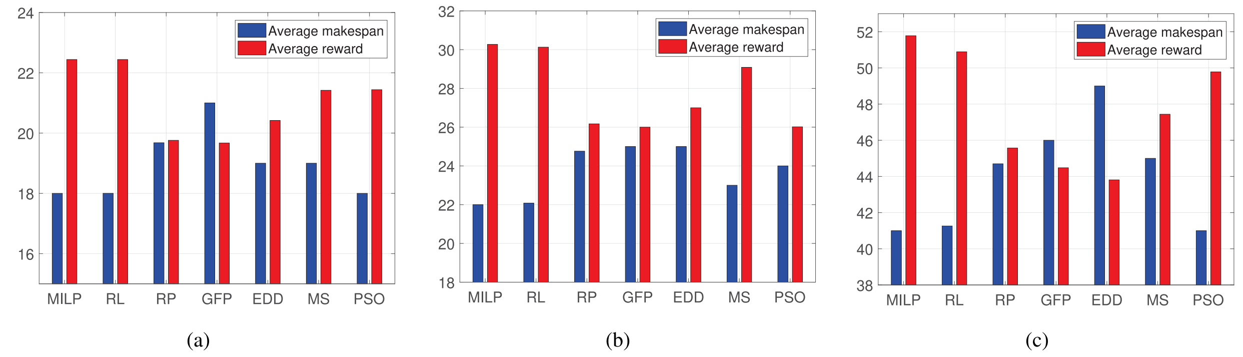


Fig. 7. Performance comparison of the four methods in terms of the average reward and average makespan over the three scale networks: (a) small-scale network, (b) medium-scale network, and (c) large-scale network.

TABLE III
RUNNING TIME COMPARISON BETWEEN RL APPROACH AND MILP

Problem Instance	Proposed RL Approach		MILP Model
	Number of episodes	CPU time (Average reward)	CPU time (Reward)
Medium ($\|\mathcal{N}\|$=8, $\|\mathcal{R}\|$=10)	1	0.0039s	5.4103s (30.273)
	10	0.0486s (26.321)	
	100	0.3462s (27.924)	
	500	1.9865s (29.590)	
	800	2.6526s (29.928)	
	1000	3.0028s (30.115)	
	2000	5.9922s (30.128)	
Large ($\|\mathcal{N}\|$=15, $\|\mathcal{R}\|$=20)	1	0.0109s	41.3272s (51.780)
	10	0.1052s (44.967)	
	100	1.3739s (48.102)	
	1000	10.7934s (50.497)	
	2000	20.7799s (50.892)	
	4000	40.2382s (51.063)	
	5000	52.9593s (51.369)	
	10000	106.6593s (51.780)	

expands. In summary, we can conclude that the proposed RL approach is time-efficient with negligible loss of accuracy.

D. Performance Comparison With Heuristic Algorithms

In this section, we compare the performance of the proposed RL approach with the optimal solutions obtained from the Gurobi solver and those from classical heuristic/metaheuristic algorithms. In particular, we consider four greedy algorithms [44]: 1) Greedy fast processing (GFP) in which the VNF with the shortest processing time is always selected first; 2) Earliest due date first (EDD) in which the VNF with the earliest due time is always selected first; 3) Minimum slack first (MS) which always selects the VNF with the minimum slack (defined as $D_i - t_{cur} - \rho_{ij}$, where t_{cur} is the current system time); and 4) Random policy (RP) in which the agent randomly schedules a feasible VNF for processing at each time slot. In addition, we also consider a particle swarm optimization (PSO) based algorithm [45], [46] for the purpose of comparison. According to [46], we modify the way to evaluate the fitness function for each particle by adding a penalty term. For a given particle, if one delay constraint is not satisfied, a penalty is added to the fitness function.

We compare the performance of the seven methods in terms of the (average) reward and the (average) makespan over the three scale networks. For the proposed approach, we set n_{max} to be 400 (0.343 s), 1000 (3.002 s), and 2000 (20.78 s) for the small, medium, and large problem instances, respectively, to allow the RL approach to converge. For the PSO-based algorithm, the number of particles, the number of iterations, and the corresponding running time for the three network scales are (100, 200, 0.65 s), (300, 1000, 10.11 s), (500, 1000, 32.377 s), respectively. The performance comparison for the seven methods are shown in Fig. 7. It is seen that for all the three problem instances, the average reward and makepsan from the RL approach stay very close to the optimal ones obtained from MILP, which indicates that in most cases the minimum makespan is found and the delay constraints are all satisfied. In contrast, using the PSO algorithm, in the small-scale network, the minimum makespan can be found with a delay constraint being violated. Note that the maximum reward can only be achieved when the PSO parameters are increased to (500, 200, 3.04 s), which indicates that the minimum makespan is found and the delay constraints are all satisfied. A similar conclusion can be drawn for the large-scale network. With 500 particles and 1000 iterations, PSO cannot obtain the maximum reward (with two delay constraints being violated). Even if we use 1000 particles and 2000 iterations, which takes 116.14 s, the minimum makespan still cannot be found with all the delay constraints being satisfied. This demonstrates that the proposed RL approach can achieve the minimum makespan while satisfying all the delay constraints using a shorter time than the PSO-based algorithm.

As for the other four heuristic algorithms, we observe that their solution qualities depend on the objective of the problem, and for different problem instances their performance cannot be guaranteed. Among the four heuristic algorithms, the MS algorithm performs the best in all the three network scales. The reason is that the MS algorithm aims to minimize the maximum lateness and takes the delay constraints into consideration when selecting the VNF for processing. However, performance gap exists between the solution from the MS algorithm and the optimal solution, and this gap increases as network scale becomes larger. Considering that the optimality

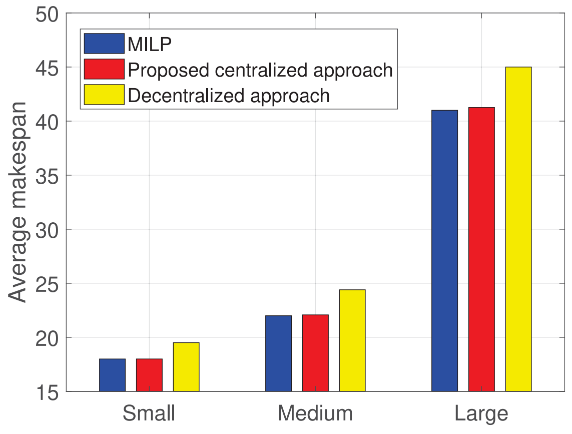


Fig. 8. Comparison between the proposed centralized approach and the decentralized approach in terms of makespan.

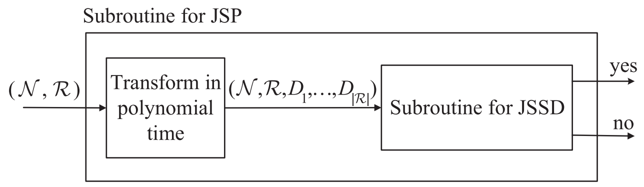
gap between the RL solutions and the optimal ones are consistently small, we can conclude that the proposed RL approach outperforms classical heuristic algorithms, and also well competes with the MILP model over all of the three networks. In addition, the performance gaps between the RL approach and the heuristic algorithms in terms of the reward are bigger than those for the makespan for all the three network scales. This is because the RL approach achieves delay-aware VNF scheduling where not only the makespan is minimized but also the delay constraints are satisfied. Therefore, the additional rewards for satisfying the delay constraints are obtained. Note that the makespan differences in Fig. 7 indicate the packet batch processing time gaps between RL and the heuristic algorithms in one scheduling cycle. Once the scheduling sequence is found, each cycle of VNF scheduling proceeds repeatedly for future packet batch processing, which accumulates the gap and leads to a considerable performance difference between the RL approach and the other five algorithms in comparison.

E. Performance Comparison With Decentralized Solution

In this section, we compare the proposed centralized approach and the decentralized one [23]. For the decentralized approach, we attach to each NFV server an agent which improves its VNF scheduling policy independently with the help of RL. Each agent only has a local view of the system, which contains the status of the NFV node it is associated with and the status of all the VNFs on that node. Similar to [23], the immediate cost $C(s_t, a_t)$ for taking action a_t at state s_t is given by

$$C(s_t, a_t) := \sum_{k=1}^{|\mathcal{N}|} \left|\{f_{ij} \,|\, f_{ij} \text{ is queued at NFV node } n_k\}\right|. \tag{25}$$

Similar to the reward function $R(s_t, A_t)$ in this paper, this cost function $C(s_t, a_t)$ is utilized in updating the *Q*-table maintained within each learning agent. The intuition behind this cost function is that a high utilization of the resources (i.e., NFV nodes) implies a minimal makespan.



Subroutine for JSP
$(\mathcal{N},\mathcal{R})$
Transform in polynomial time
$(\mathcal{N},\mathcal{R},D_1,\ldots,D_{|\mathcal{R}|})$
Subroutine for JSSD
yes
no

Fig. 9. Illustration of reducing JSP to JSSD.

Since delay constraints are not considered in the decentralized approach, we set $c_i = 0$ $(i = 1, \ldots, |\mathcal{R}|)$ in the reward function in our approach and compare the performance in terms of makespan only. Fig. 8 shows the comparison between the two approaches over three scale networks. It can be seen that the proposed centralized approach outperforms the decentralized one. Our centralized approach has improved performance by taking advantage of a global view of the status of NFV nodes and services in the whole system.

VII. Conclusion and Future Work

In this paper, we have investigated a VNF scheduling problem under an SDN/NFV-enabled network architecture, with the objective of minimizing the overall makespan of services while satisfying differentiated E2E delay requirements. The problem has been first formulated as an MILP and then reformulated as an MDP problem. A reinforcement learning approach has been proposed to obtain near-optimal solutions to the VNF scheduling problem with high efficiency and accuracy. Simulation results have been presented to demonstrate that the proposed approach outperforms other heuristic algorithms and can achieve near-optimal solutions. The proposed approach facilitates QoS-guaranteed service provisioning in SDN/NFV-enabled networks. Since the *Q*-table was learnt and applied on one fixed problem instance. As our future work, we will investigate a more general RL approach which can learn a set of training problem instances but apply to unknown instances with similar patterns. We will also consider stochastic arrivals of service requests in the formulation of the VNF scheduling problem.

Appendix

The classical JSP is NP-complete [47], and hence it is strongly believed that JSP cannot be solved in polynomial time. In our work, we will show that JSP with deadlines (JSSD) is NP-hard, if JSP is NP-complete, i.e.,

$$(JSP \notin P) \Rightarrow (JSSD \notin P). \tag{26}$$

where *P* represents the class of problems that can be solved in polynomial time. We will prove the contrapositive:

$$(JSSD \in P) \Rightarrow (JSP \in P). \tag{27}$$

Assume that we have access to a polynomial time subroutine JSSD $(\mathcal{N}, \mathcal{R}, D_1, \ldots, D_{|\mathcal{R}|})$. The inputs to the subroutine are a set of NFV nodes $\mathcal{N}$, a set of services $\mathcal{R}$, and their corresponding delay constraints D_i, $i = 1, \ldots, |\mathcal{R}|$. The output of this subroutine is *true* if a feasible schedule exists such that the makespan is smaller than or equal to *M* (where *M* is a positive integer) and all the delay constraints are satisfied, and is *false* otherwise [48]. Obviously, a problem instance ($\mathcal{N}$, $\mathcal{R}$) for JSP can be transformed in polynomial time into an instance $(\mathcal{N}, \mathcal{R}, D_1, \ldots, D_{|\mathcal{R}|})$ for JSSD (see Fig. 9). We can also observe that both problems need to answer whether a schedule exists such that the makespan is smaller than or equal to M. Let $D_1, D_2, \ldots, D_{|\mathcal{R}|}$ all be equal to M. Then, the outputs of JSP and JSSD are consistent. Suppose (towards a contradiction) that a polynomial time algorithm for JSSD exists, then we can use this algorithm to solve JSP in polynomial time, thus yielding a contradiction. Therefore, (27) is true, and as the contrapositive of (27), (26) is also true. This shows that the JSSD can be reduced from JSP in polynomial time and thus is NP-hard.

REFERENCES

[1] J. Li, W. Shi, N. Zhang, and X. Shen, "Reinforcement learning based VNF scheduling with end-to-end delay guarantee," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC'19)*, Changchun, China, Aug. 2019, pp. 572–577.

[2] "5G; Study on scenario and requirements for next generation access technologies, V14.2.0," 3GPP, Sophia Antipolis, France, Rep. TR 38.913, 2018.

[3] Q. Ye, J. Li, K. Qu, W. Zhuang, X. Shen, and X. Li, "End-to-end quality of service in 5G networks: Examining the effectiveness of a network slicing framework," *IEEE Veh. Technol. Mag.*, vol. 13, no. 2, pp. 65–74, Jun. 2018.

[4] A. J. Gonzalez, G. Nencioni, A. Kamisinski, B. E. Helvik, and P. E. Heegaard, "Dependability of the NFV orchestrator: State of the art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3307–3329, 4th Quart., 2018.

[5] J. Li, N. Zhang, Q. Ye, W. Shi, W. Zhuang, and X. Shen, "Joint resource allocation and online virtual network embedding for 5G networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM'17)*, Singapore, Dec. 2017, pp. 1–6.

[6] *5G Vision: The 5G Infrastructure Public Private Partnership: The Next Generation of Communication Networks and Services*, Infrastruct. Assoc., Alexandria, VA, USA, Feb. 2015.

[7] S. Zhang, W. Quan, J. Li, W. Shi, P. Yang, and X. Shen, "Air-ground integrated vehicular network slicing with content pushing and caching," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 9, pp. 2114–2127, Sep. 2018.

[8] G. Baggio, R. Bassoli, and F. Granelli, "Cognitive software-defined networking using fuzzy cognitive maps," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 3, pp. 517–539, Sep. 2019, doi: 10.1109/TCCN.2019.2920593.

[9] J. Chen *et al.*, "SDATP: An SDN-based traffic-adaptive and service-oriented transmission protocol," *IEEE Trans. Cogn. Commun. Netw.*, early access, Dec. 31, 2019, doi: 10.1109/TCCN.2019.2963149.

[10] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu, "MicroNF: An efficient framework for enabling modularized service chains in NFV," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1851–1865, Aug. 2019, doi: 10.1109/JSAC.2019.2927069.

[11] N. Zhang *et al.*, "Software defined networking enabled wireless network virtualization: Challenges and solutions," *IEEE Netw.*, vol. 31, no. 5, pp. 42–49, Apr. 2017.

[12] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[13] J. F. Riera, E. Escalona, J. Batallé, E. Grasa, and J. A. García-Espín, "Virtual network function scheduling: Concept and challenges," in *Proc. IEEE Int. Conf. Smart Commun. Netw. Technol. (SaCoNeT)*, Vilanova i la Geltrú, Spain, Jun. 2014, pp. 1–5.

[14] J. F. Riera, X. Hesselbach, E. Escalona, J. A. García-Espín, and E. Grasa, "On the complex scheduling formulation of virtual network functions over optical networks," in *Proc. IEEE 16th Int. Conf. Transp. Opt. Netw. (ICTON)*, Graz, Austria, Jul. 2014, pp. 1–5.

[15] M. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, pp. 117–129, May 1976.

[16] 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.

[17] 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 1st Conf. Netw. Softw. (NetSoft)*, London, U.K., Apr. 2015, pp. 1–9.

[18] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.

[19] H. A. Alameddine, L. Qu, and C. Assi, "Scheduling service function chains for ultra-low latency network services," in *Proc. IEEE 13th Int. Conf. Netw. Serv. Manage. (CNSM)*, Tokyo, Japan, 2017, pp. 1–9.

[20] C. Pham, N. H. Tran, and C. S. Hong, "Virtual network function scheduling: A matching game approach," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 69–72, Jan. 2018.

[21] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016. [Online]. Available: arXiv:1611.09940.

[22] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. 28th Int. Conf. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2692–2700.

[23] T. Gabel and M. Riedmiller, "Adaptive reactive job-shop scheduling with reinforcement learning agents," *Int. J. Inf. Technol. Intell. Comput.*, vol. 24, no. 4, pp. 1–30, 2008.

[24] T. Gabel and M. Riedmiller, "Distributed policy search reinforcement learning for job-shop scheduling tasks," *Int. J. Prod. Res.*, vol. 50, no. 1, pp. 41–61, 2012.

[25] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. 17th Int. Conf. Mach. Learn. (ICML'00)*, 2000, pp. 535–542.

[26] M. M. Ling, K. A. Yau, J. Qadir, and Q. Ni, "A reinforcement learning-based trust model for cluster size adjustment scheme in distributed cognitive radio networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 1, pp. 28–43, Mar. 2019.

[27] S. Riedmiller and M. Riedmiller, "A neural reinforcement learning approach to learn local dispatching policies in production scheduling," in *Proc. 16th Int. Joint Conf. Artif. Intell. (IJCAI)*, vol. 2, 1999, pp. 764–771.

[28] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, Ottawa, ON, Canada, May 2015, pp. 98–106.

[29] O. Alhussein *et al.*, "A virtual network customization framework for multicast services in NFV-enabled core networks," *IEEE J. Sel. Areas Commun.*, early access, Apr. 8, 2020, doi: 10.1109/JSAC.2020.2986591

[30] Z. Ye, X. Cao, J. Wang, H. Yu, and C. Qiao, "Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization," *IEEE Netw.*, vol. 30, no. 3, pp. 81–87, May/Jun. 2016.

[31] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint optimization of service function chaining and resource allocation in network function virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016.

[32] R. Li, Z. Zhao, X. Chen, J. Palicot, and H. Zhang, "TACT: A transfer actor-critic learning framework for energy saving in cellular radio access networks," *IEEE Trans. Wireless. Commun.*, vol. 13, no. 4, pp. 2000–2011, Apr. 2014.

[33] Y. Hua, R. Li, Z. Zhao, H. Zhang, and X. Chen, "GAN-based deep distributional reinforcement learning for resource management in network slicing," in *Proc. IEEE Global Commun. Conf. (Globecom'19)*, Big Island, Hawaii, USA, 2019, pp. 1–6.

[34] J. C. R. Bennett and H. Zhang, "WF^{2}Q: Worst-case fair weighted fair queueing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM'96)*, San Francisco, CA, USA, 1996, pp. 120–128.

[35] C. A. Floudas and X. Lin, "Continuous-time versus discrete-time approaches for scheduling of chemical processes: A review," *Comput. Chem. Eng.*, vol. 28, no. 11, pp. 2109–2129, 2004.

[36] C. Szepesvari, "Algorithms for reinforcement learning," *Syn. Lec. Artif. Intell. Mach. Learn.*, vol. 4, no. 1, pp. 1–103, 2010.

[37] R. Mijumbi, J. Gorricho, J. Serrat, M. Claeys, F. D. Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *Proc. IEEE Netw. Oper. Manag. Symp. (NOMS)*, Krakow, Poland, May 2014, pp. 1–9.

[38] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[39] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, U.K: Cambridge Univ. Press, 2011.

[40] C. J. Watkins and P. Dayan, "*Q*-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[41] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural Comput.*, vol. 6, no. 6, pp. 1185–1201, Nov. 1994.

[42] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 2, pp. 257–265, Jun. 2018.

[43] F. S. Melo, "Convergence of Q-learning: A simple proof," Dept. Inst. Syst. Robot., Instituto Superior Técnico, Lisbon, Portugal, Rep., pp. 1–4, 2001.

[44] M. Pinedo, *Planning and Scheduling in Manufacturing and Services*. New York, NY, USA: Springer, 2005.

[45] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Perth, WA, Australia, 1995, pp. 1942–1948.

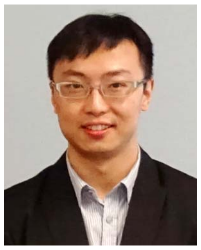
[46] K. Masuda, K. Kurihara, and E. Aiyoshi, "A penalty approach to handle inequality constraints in particle swarm optimization," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Istanbul, Turkey, Oct. 2010, pp. 2520–2525.

[47] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.

[48] G. Mosheiov, "Complexity analysis of job-shop scheduling with deteriorating jobs," *Discrete Appl. Math.*, vol. 117, nos. 1–3, pp. 195–209, Mar. 2002.

Junling Li (Graduate Student Member, IEEE) received the B.S. degree from Tianjin University, Tianjin, China, in 2013, and the M.S. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2016. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. Her current research interest include SDN, NFV, network slicing for 5G networks, machine learning, and vehicular networks. She received the Best Paper Award at the IEEE/CIC International Conference on Communications in China in 2019.

Weisen Shi (Graduate Student Member, IEEE) received the B.S. degree from Tianjin University, Tianjin, China, in 2013, and the M.S. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2016. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His interests include drone communication and networking, NFV, and vehicular networks. He received the Best Paper Award at the IEEE/CIC International Conference on Communications in China in 2019.



Ning Zhang (Senior Member, IEEE) received the Ph.D. degree from the University of Waterloo, Canada, in 2015.

He was a Postdoctoral Research Fellow with the University of Waterloo and the University of Toronto, Canada. Since 2017, he has been an Assistant Professor with Texas A&M University Corpus–Christi, USA. He received the Best Paper Awards from the IEEE Globecom in 2014, the IEEE WCSP in 2015, the *Journal of Communications and Information Networks* in 2018, the IEEE ICC in 2019, the IEEE Technical Committee on Transmission Access and Optical Systems in 2019, and the IEEE ICCC in 2019. He serves as an Associate Editor for the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING, IEEE ACCESS, *IET Communications*, and *Vehicular Communications* (Elsevier) and a Guest Editor for several international journals, such as IEEE WIRELESS COMMUNICATIONS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and the IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING. He also serves/served as a track chair for several international conferences and a co-chair for several international workshops.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990.

He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular ad hoc and sensor networks. He received the R. A. Fessenden Award in 2019 from IEEE, Canada, the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Joseph LoCicero Award in 2015, and the Education Award in 2017 from the IEEE Communications Society. He has also received the Excellent Graduate Supervision Award in 2006 and the Outstanding Performance Award 5 times from the University of Waterloo and the Premier's Research Excellence Award in 2003 from the Province of Ontario, Canada. He served as the Technical Program Committee Chair/Co-Chair for the IEEE Globecom'16, the IEEE Infocom'14, the IEEE VTC'10 Fall, the IEEE Globecom'07, the Symposia Chair for the IEEE ICC'10, the Tutorial Chair for the IEEE VTC'11 Spring, and the Chair for the IEEE Communications Society Technical Committee on Wireless Communications. He was the Editor-in-Chief of the IEEE INTERNET OF THINGS JOURNAL and the Vice President on Publications of the IEEE Communications Society. He is a Registered Professional Engineer of Ontario, Canada. He is a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society. He is a fellow of the Engineering Institute of Canada, the Canadian Academy of Engineering, the Royal Society of Canada, and the Chinese Academy of Engineering Foreign.