IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 22, 2020, accepted August 31, 2020, date of publication September 7, 2020, date of current version September 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3022038

Adaptive Service Function Chain Scheduling in Mobile Edge Computing via Deep Reinforcement Learning

TIANFENG WANG, JIACHEN ZU, GUYU HU, AND DONGYANG PENG
Institute of Command and Control Engineering, Army Engineering University, Nanjing 210007, China
Corresponding author: Guyu Hu (huguyu@189.cn)

ABSTRACT MEC (Mobile Edge Computing) provides both IT service environment and cloud computation on the edge of the network. This technology not only minimizes the end-to-end latency but also increases the efficiency of computing. Some latency-sensitive applications, such as cloud video, online game, and augmented reality, take advantage of the MEC system to provide fast and stable services. Several new network techniques, including the implementation of NFV (Network Function Virtualization), the placement of VNF (Virtual Network Function) and the scheduling of SFC (Service Function Chain), should be considered to be applied in the MEC system. In this paper, we focus on the research about the scheduling of SFC in the NFV enabled MEC system and propose a solution accordingly. First, we make reasonable assumptions on the settings of MEC systems and model the SFC scheduling problem into a flexible job-shop scheduling problem. Since minimizing the latency can significantly improve the quality of service (QoS) and increase the revenue of Internet Service Providers, our optimization goal is to minimize the overall scheduling latency. To solve this optimization problem, a deep reinforcement learning based algorithm DQS is proposed. DQS can detect the variation of the MEC system's environment and perform adaptive scheduling for SFC requests. As the results of the simulation indicate, DQS works better than the other off-the-shelf algorithms in two key indexes: overall scheduling latency and average resource usage. Moreover, DQS can shorten the decision time and schedule SFCs stably with high performance. It is suitable to be extended to an online scheduling algorithm.

INDEX TERMS Service function chain, mobile edge computing, scheduling optimization, deep reinforcement learning.

I. INTRODUCTION

Nowadays, 5G is integrated with different prevalent technologies including cloud computing and big data. It promises to connect people and everything and becomes a key infrastructure for the digital transformation of various industries. MEC [1] (Mobile Edge Computing) is an essential technology in 5G architecture, which not only provides a closer cloud service environment on the mobile network edge but also lightens the load of the backbone transmission network. More importantly, MEC reduces the end-to-end latency and improves QoS of users. The core devices of the MEC system are the MEC servers built on the standard IT hardware platform. These servers are generally integrated within MDCs [2] (Micro-Data Center), which can be placed close to the base stations and provide network services for mobile devices.

Network Function Virtualization (NFV) [3] and Software Defined Network (SDN) [4] are two fundamental technologies in the MEC system. NFV virtualizes network functions by uncoupling hardware and software, making it possible to operate the network service in a standard virtualized platform. By separating network controlling and forwarding, SDN enables the flexible management of network transmission and computing resources. The integration of NFV and SDN promises to enable cooperative control and scheduling of network function instances in MDCs.

In this paper, we take the following assumptions. Firstly, all the service requests arrive in the form of Service Function Chain (SFC) from mobile devices, i.e., network flows need

The associate editor coordinating the review of this manuscript and approving it for publication was Petros Nicopolitidis.

to traverse through a series of network function instances to accomplish specific tasks. Secondly, we assume that all service functions are deployed on the Virtual Machines (VMs) of MEC servers, which are named as Virtual Network Function (VNF) instances. As shown in Fig. 1, the MEC system can provide localized public cloud services through MEC servers deployed within an MDC and it can also provide hybrid cloud services by connecting to other MDCs.

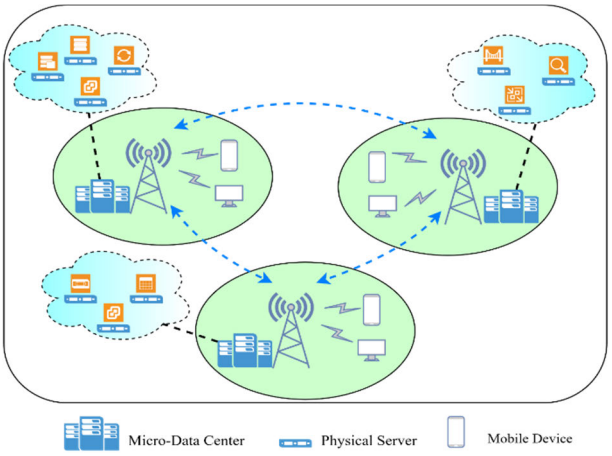


FIGURE 1. Network structure of MEC system.

The network performance of the NFV enabled MEC system is mainly related to NFV implementation, VNF placement and SFC scheduling. For NFV implementation and VNF placement, there exist off-the-shelf solutions. Firstly, for NFV implementation, OpenBox [5] and SNF [6] synthesize multiple VNFs into one equivalent VNF to improve the performance. By contrast, ResQ [7] optimizes the implementation by restricting VNFs on independent physical equipment and balancing cache. Secondly, for VNF placement, Sang *et al.* [8] focus on using the minimum number of VNF instances to serve all flows and Hawilo *et al.* [9] leverage betweenness centrality to embed VNFs into the data center (DC). However, there are few studies on SFC scheduling of the MEC system compared with NFV implementation and VNF placement.

The optimization of SFC scheduling problem aims to minimize the total scheduling latency of service under the network resource constraints, which contributes to improve the throughput of the MEC network and provide high-quality service to users. We show an SFC scheduling example in Fig. 2, where two SFCs need to be scheduled in a MEC system. There are two MDCs in the scenario and three VMs are deployed within each MDC. In SFC1, the processing order is VNF1 → VNF2 → VNF3 → VNF4 → VNF5; in SFC2, the processing order is VNF6 → VNF5 → VNF3 → VNF4 → VNF1. In this example, the computational resources and bandwidth resources are pre-allocated before scheduling. Then, we mainly focus on scheduling the SFCs by matching VNFs and VMs.

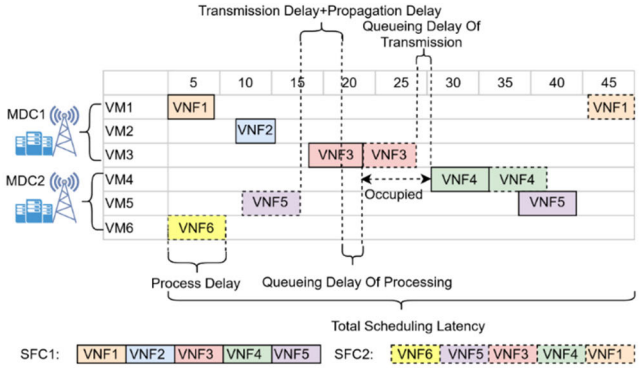


FIGURE 2. SFC scheduling example in MEC.

The horizontal axis represents the scheduling latency of SFCs. The overall scheduling latency includes processing delay, transmission delay, propagation delay, queuing delay of processing and queuing delay of transmission. In Fig. 2, the total scheduling latency of SFC1 is 38 and the total scheduling latency of SFC2 is 45.

The processing delay of a VNF instance depends on its type and the performance of master VM, which is different from each other. In this example, we make a reasonable assumption about the processing delay of various VNFs over VMs. For example, the processing delay of VNF6 is different from VNF1. In the transmission progress between MDCs, the propagation delay cannot be ignored due to the geographical distance, e.g., the transmission between VNF5 and VNF3 in SFC2. Due to the limitation of CPU resources, VNF requests of different SFCs are supposed to be processed by VMs in serial mode. On VM3, the processing of VNF3 in SFC2 needs to wait until the processing of VNF3 in SFC1 is completed, which results in the queueing delay of processing. In this example, we assume that there is only one virtual link between MDC1 and MDC2. The transmission of VNF4 in SFC1 results in the queuing delay of transmission, since SFC2 needs to wait until the link between MDC1 and MDC2 is idle after the completion of the transmission of VNF4 in SFC1.

Once the VNF instances are deployed on VMs, the VM selection and the VNF scheduling order for SFC requests will surely affect the total latency. Existing works formulate the scheduling problem as a flexible job-shop problem [10], [35], [36], which has been proven with no polynomial-time solution. Traditional linear or nonlinear programming methods are ineffective in solving this problem. To address the above challenges, we design a novel deep Q-learning [11] based approach named DQS to schedule SFCs efficiently. Our proposed approach has the following advantages. Firstly, DQS can automatically adapt to the changes in the MEC system without manual intervention. Secondly, DQS is stable in performance for different sizes of scheduling tasks. Furthermore, DQS can be easily extended as an effective online scheduling method. The main contributions of this paper are as follows:

(1) Considering the MEC system's characteristics, we formalize the SFC scheduling problem into a flexible job-shop scheduling problem aiming at minimizing the total latency.

(2) Comprehensively considering the factors affecting the scheduling latency, we propose DQS, an adaptive approach based on DQN. Also, the intelligent agent and training procedure are carefully designed to schedule SFCs in the MEC system automatically.

(3) The performance evaluation shows that DQS achieves better performance in total latency and resource utilization compared to the state-of-the-art solutions. Meanwhile, we analyze the reasons for high efficiency of DQS and put forward some ideas for further improvement.

The rest of this paper is organized as follows. Section II discusses the related work. Section III describes the system model in detail. Section IV formulates the SFC scheduling problem in MEC system and defines the optimization goal. Section V proposes DQS with a detailed description. Section VI presents the evaluations compared to other existing scheduling algorithms. At last, Section VII is the conclusion.

II. RELATED WORK

Driven by the visions of IoT and 5G communications, MEC has become an essential paradigm of mobile computing [12]. There are researches about the MEC system from a variety of perspectives.

Research [13] introduces a taxonomy for MEC applications and analyzes chances and limitations from a technical point of view. To bridge the gap between the industry and the academia, different MEC deployment, discovery, and communication options are discussed [14]. MEC is designed as an infrastructure to support various IoT applications [15], [16]. Meanwhile, some challenges of the MEC system are summarized in [17], [18]. To facilitate the transformation of MEC from theory to practice [19], [20], some issues, such as service migration [21], resource orchestration [22] [23] and service scheduling, need to be taken more seriously. In paper [21], two novel conceptions, live migration for data centers and handover in cellular networks, are proposed to implement service migration. Some underlying design principles of resource orchestration are explored in [24]. Beyond that, MEC orchestration considering both application developers and content providers is discussed [25], [26].

The resource optimization of MEC is an important subject that has been fully discussed in previous work. A flexible framework is proposed which can be used to optimize the service selection process, according to various metrics [27]. Multiple criteria decision analysis [28] is used to complete the dynamic autonomous resource management in computing clouds. To deal with the SLA (Service Level Agreements) based resource allocation problem, an upper bound on the total profit is provided and an algorithm based on force-directed search is proposed [29]. Considering cloud users always assign their VMs with specific PM (physical machine) requirements [30], an efficient solution for VM placement is proposed. Under the cloud deployment cost limitation, an enhanced BnB (branch-and-bound) heuristic approach for virtual function placement is proposed [31]. In [32], the VM placement is defined as a multi-objective optimization problem and a multi-objective memetic algorithm is proposed to solve it. In multi-cloud environment, clouds can be selected to optimize the cost and the speed of service deployment can be increased with appropriate scheme. To achieve these objectives, a P-ART (Predictive-Adaptive Real Time) [33] framework is proposed, which was examined in CloudLab. In addition, performance evaluation of multi-cloud management in software level is discussed. The authors in [34] point out that functions like host creation and polling have significant impact on the performance of the platform software.

Scheduling optimization in DC has been fully discussed in existing works. However, there is little related references on scheduling in the MEC environment. In [35], the first formalization model for the VNF complex scheduling problem is proposed. The scheduling problem is ascribed to a flexible job-shop scheduling problem (FJSP) [10], which is a classic NP-hard problem. Although the above papers have well formalized the SFC scheduling problem, they don't provide specific solutions. Assuming that the VM can process multiple VNFs in order, three greedy algorithms and a tabu search-based method are used for online placement and SFC scheduling [36]. Moreover, the simulations show that the tabu search-based method works only slightly better than the best greedy algorithm. Reducing the scheduling latency enables cloud operators to serve more customers and cater to services with stringent delay requirement, SFC scheduling and traffic steering are jointly formulated as a Mixed Integer Linear Program (MILP) [37]. Meanwhile, a Genetic Algorithm (GA) based method is proposed in [37] to reduce the complexity of SFC scheduling, which guarantees the scheduling efficiency in the small-scale scenario. Authors in [38] further refine the MILP model by comprehensively considering the interactions among NFs mapping onto VNFs, service scheduling and traffic routing. Given the complexity of the SFC scheduling problem, they present a primal-dual decomposition using column generation that solve a relaxed version of the problem. In order to solve the scheduling problem in the online scenario, a matching-based algorithm is proposed, which can guarantee a stable scheduling [39]. For the scheduling optimization in MEC scenario, an affinity-based fair weighted scheduling heuristic [14] is proposed which works better than standard greedy scheduling algorithms.

In the MEC system, the physical servers are typically installed in distributed MDCs to reduce the complexity of centralized orchestration. Multiple VMs are set up in each MDC to implement various VNFs. Compared to the single data center scenario, more factors affecting scheduling need to be taken into account in the MEC scenario, e.g., selection of access point for each SFC and transmission between MDCs. Thus, we formalize the scheduling problem in the MEC system, which can serve as a benchmark for new research and propose an efficient approach, DQS, to solve the problem.

III. SYSTEM MODEL

A. SUBSTRATE NETWORK

The distributed cloud network is generally modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. It typically consists of multiple MDCs. $\mathcal{V}$ is the set of MDCs and E is the set of virtual links. Here, we use $\mathcal{E} = \mathcal{E}^{in} \cup \mathcal{E}^{out}$ to distinguish links within MDCs from links between MDCs. For each link $e \in \mathcal{E}$, its bandwidth capacity is B_e. Since the MDCs are geographically far away from each other in the MEC system, we use D_e to represent the physical distance of link $e \in \mathcal{E}^{out}$ and ignore the propagation delay within one MDC. There are multiple VNF instances deployed on each MDC. Each VNF instance is running on one VM in MDC. For each MDC $\upsilon \in \mathcal{V}$, we use M_v to represent its supported VNF instance number.

B. SERVICE FUNCTION CHAIN REQUESTS

In the distributed network, each flow entering the network is in the form of SFC. A three-tuple is used to symbolize each SFC request $S_i = \{o_i, \varphi_i, size_i\}$. For each SFC $S_i \in S$, $o_i \in \mathcal{V}$ indicates the ingress node of S_i and $size_i$ indicates the stream size of this flow. Furthermore, each S_i is composed of a series of VNFs in a specific order which is defined as $\varphi_i = \{\varphi_{i1}, \varphi_{i2}, \ldots, \varphi_{il}\}$. φ_{ij} represents the j^{th} VNF request of S_i and $l = |\varphi_i|$ represents the VNF length of S_i. Given a set of SFC commodities S, we use $|S|$ to represent the total number of the existed commodities in the network.

IV. PROBLEM FORMULATION

A. PROBLEM DESCRIPTION

Reducing service scheduling latency contributes to increase network throughput and improve the QoS to users. To achieve this goal, the network operators need to joint and optimize SFC placement and SFC scheduling order. Given the network topology, we assume several VNF instances have been deployed in MDCs. Due to the insufficient resources in single MDC, intercommunication between MDCs is an effective solution which means SFC commodities can traverse through different MDCs to complete network services. In this paper, we aim at optimizing the overall scheduling latency of a set of SFC requests, which can be formulated as a flexible job-shop problem.

B. DETAILED FORMULATION

In this section, we formulate the SFC scheduling problem with strict definitions. The main notations are listed in TABLE 1.

Here, $x_e^{S_i}(t)$ indicates whether the SFC S_i is transmitting in link e and $ac_e^{S_i}$ represents the transmission rate of S_i in link e. Firstly, any SFC S_i cannot be shunted as shown in Eq. (1). Secondly, we assume that the transmission on each link is a separate dynamic queue model [40]. For each link e, the total transmission rate of SFCs transmitting on e is limited by B_e as shown in Eq. (2). The length of queue waiting for transmission on link e changes dynamically as shown in Eq. (3).

TABLE 1. Symbols and variables.

Notations	Explanation
Substrate Network	
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Distributed network topology
N	Number of MDCs
M_v	VM set of MDC $\upsilon \in \mathcal{V}$
L_v	Link set of MDC $\upsilon \in \mathcal{V}$
F	VNF type set
$p_{v,m}$	Processing delay of the m^{th} VM of MDC υ
$\mathcal{E}^{in}, \mathcal{E}^{out}$	Internal link set within MDCs and link set between MDCs, where $\mathcal{E} = \mathcal{E}^{in} \cup \mathcal{E}^{out}$
B_e	Bandwidth capacity of link e $\in \mathcal{E}$
D_e	Physical distance of link e $\in \mathcal{E}$
WQ_e	The queue length waiting for transmission on link e.
Service Function Chain	
$S_i = \{o_i, \varphi_i, size_i\}$	$o_i \in \mathcal{V}$, ingress node of S_i; $size_i$, stream size of flow
φ_i	The form of VNFs in S_i, $\varphi_i = \{\varphi_{i1}, \varphi_{i2}, \ldots, \varphi_{il}\}$, $l = \|\varphi_i\|$ represents the VNF number of S_i
$\mu_e^{S_i}$	The transmission rate of S^i in link e
Binary Variable	
$x_e^{S_i}$	Boolean variable that equals 1 if S_i is transmitting in link e and 0 otherwise
$y_{v,m}^{\varphi_{ij}}$	Boolean variable that equals 1 if VNF request φ_{ij} is processing on the m^{th} VM of MDC $\upsilon \in \mathcal{V}$ and 0 otherwise
$z_{v,m}^{f}$	Boolean variable that equals 1 if f type VNF instance is deployed on the m^{th} VM of MDC $\upsilon \in \mathcal{V}$ and 0 otherwise
$w_e^{S_i}$	Boolean variable that equals 1 if S_i is newly arrival SFC for e and 0 otherwise

$$\sum_{j=1}^{|\varepsilon|} x_{\varepsilon_j}^{S_i}(t) \leq 1 \forall i \in [1, |S|] \tag{1}$$

$$\sum_{i=1}^{|S|} x_e^{S_i}(t) . \mu_e^{S_i}(t) \leq B_e \forall e \in \varepsilon \tag{2}$$

$$\begin{aligned} WQ_e(t) &= \max(WQ_e(t-1) - \sum_{i=1}^{|S|} x_e^{S_i}(t-1) . \mu_e^{S_i}(t-1), 0) \\ &+ \sum_{i=1}^{|S|} w_e^{S_i} . size_i \end{aligned} \tag{3}$$

We assume that the VNF requests of different SFCs are processed sequentially on VMs according to FIFO (first in first out) principle. We use $y_{v,m}^{\varphi_{ij}}$ to indicate whether VNF request φ_{ij} is processing on the m^{th} VM of MDC $\upsilon \in \mathcal{V}$.

$$\sum_{i=1}^{|S|} \sum_{j=1}^{|\varphi_i|} y_{v,m}^{\varphi_{ij}}(t) \leq 1 \quad \forall m \in M_v \; \forall v \in \mathcal{V} \tag{4}$$

For each VM in MDCs, only one type of VNF $f \in F$ is allowed to be deployed as shown in Eq. (5). Meanwhile, the VNF types of VMs on the same MDC are supposed to be different from each other as shown in Eq. (6). We use $z^f_{v,m}$ to indicate whether the f type VNF instance is deployed on the m^{th} VM of MDC $\upsilon \in \mathcal{V}$.

$$\sum_{f \in F} z^f_{v,m} = 1 \quad \forall m \in M_v \forall v \in \mathcal{V} \tag{5}$$

$$\sum_{f \in F} z^f_{v,m} . z^f_{v,m'} = 0 \quad \forall m \neq m' \ \forall v \in \mathcal{V} \tag{6}$$

C. DEFINITION OF THE SCHEDULING LATENCY

For each SFC S_i, the total scheduling latency consists of transmission delay, propagation delay, processing delay and queueing delay of processing. The notations are listed in TABLE 2.

TABLE 2. Symbols and variables.

Notations	Explanation
T_{s_i}	Total scheduling latency of SFC S_i
$\mathbb{V}_{s_i}$	Set of traversed VMs of SFC S_i
$\mathbb{W}_{s_i}$	Set of traversed links of SFC S_i
$d^{tran}_{s_i}$	Transmission delay
$d^{prop}_{s_i}$	Propagation delay
$d^{proc}_{s_i}$	Processing delay
$d^{qdp}_{s_i}$	Queueing delay of processing

We use $d^{tran}_{s_i}$ to represent the total transmission delay of S_i, which is related to the bandwidth allocation policy in queue model and the stream size of S_i. Here, $d^{tran}_{s_i}$ includes the extra delay caused by link queueing.

Since the VMs in the same MDCs are geographically close, we only consider the propagation delay between MDCs. $d^{prop}_{s_i}$ is defined to represent the total propagation delay of S_i.

$$d^{prop}_{s_i} = \sum_{e \in \mathbb{W}_{s_i} \cap \varepsilon^{out}} \frac{D_e}{c} \tag{7}$$

The processing delay of the VNF instance is determined by the computing capacity of VM and the type of VNF. Thus, the processing delay may vary from VM to VM. $d^{proc}_{s_i}$ is defined to represent the total processing delay.

$$d^{proc}_{s_i} = \sum_{(v,m) \in \mathbb{V}_{s_i}} p_{v,m} \tag{8}$$

Each VM has a cache set to store incoming SFC. The queueing delay of processing is proportional to the cache size of the VM. We use $d^{qdp}_{s_i}$ to represent the total queueing delay of processing.

Now, we define T_{s_i} as the overall scheduling latency of SFC S_i.

$$T_{s_i} = d^{tran}_{s_i} + d^{prop}_{s_i} + d^{proc}_{s_i} + d^{qdp}_{s_i} \tag{9}$$

In an offline scheduling task, given a set of SFC commodities, we aim at minimize the global scheduling latency for all incoming SFC requests. The objective function is defined as:

$$\begin{aligned} &\min \ \max_i T_{s_i} 1 \leq i \leq |S| \\ &Subject\ to\ Eq.\ (1) - Eq.(9) \end{aligned} \tag{10}$$

V. PROPOSED ALGORITHM

We propose a deep Q-learning based scheduling approach named DQS to solve the problem, which allows us to learn the optimal scheduling policy without any information of dynamic network statistics. In DQS, the agent gets the state information from the MEC environment and automatically performs actions. After that, the MEC environment transfers the reward to the agent and related strategies are updated according to the reward. In this procedure, scheduling process and agent training are synchronized. We can get the optimal scheduling policy after sufficient iterations of reinforcement training.

In this section, we introduce the proposed DQS approach in detail from three aspects: agent design, adaptive scheduling process, agent training procedure.

A. AGENT DESIGN

Before we introduce the proposed approach, we need to further discuss the details in the model and make some reasonable simplifications. Firstly, to reduce the complexity of the problem, we assume that there is only one physical link between any pair of MDCs. Within each MDC, any pair of VMs are connected by a single virtual link that is independent of each other. We simplify the bandwidth allocation strategy in the dynamic queuing model. In DQS, once the SFC starts transmitting, it takes up all the bandwidth resources until the transmission ends. Secondly, we ignore the delay generated from the links between VM and the access node in the same MDC, i.e., the transmission of VMs between MDCs is only related to the transmission on links between MDCs. Thirdly, we assume that VM handles requests in a FIFO principle, which means the priority of request processing is dependent on the order of the arrival time. Finally, forwarding is not considered in the scheduling process.

Here, we introduce the representations of state, action and reward in details. All the defined symbols and notions are listed in TABLE 3.

1) STATE

In our model, the components of state $\mathbb{S}$ include VM state $\mathbb{S}^v$ and link state $\mathbb{S}^e$. We number all the VMs in the distributed network and use $\mathbb{S}^v = \left\{\mathbb{S}^v_1, \mathbb{S}^v_2, \ldots, \mathbb{S}^v_g\right\}$ to represent the total VMs state. $c^{\varphi_{ij}}_n$ is symbolized to indicate whether the VNF request φ_{ij} is cached on the n^{th} VM and $rpt^{\varphi_{ij}}$ is defined to represent the residual processing time of VNF request φ_{ij}.

$$\mathbb{S}^v_n = \sum_{i=1}^{|S|} \sum_{j=1}^{|\varphi_j|} \left(c^{\varphi_{ij}}_n . p_n + y^{\varphi_{ij}}_n . rpt^{\varphi_{ij}} \right) \tag{11}$$

We number all the links and use $\mathbb{S}^e = \left\{\mathbb{S}^e_1, \mathbb{S}^e_2, \ldots, \mathbb{S}^e_k\right\}$ to represent the total link state. We define rtt^{S_i} that equals the

TABLE 3. Symbols and variables.

Notations	Explanations
$\mathbb{S}^v$	The overall VMs state set, $\mathbb{S}^v = \{\mathbb{S}_1^v, \mathbb{S}_2^v, \ldots, \mathbb{S}_g^v\}, g = \lvert\mathcal{V}\rvert$, where $\mathbb{S}_n^v$ represents the state of n^{th} VM
$\mathbb{S}^e$	The overall link state set, $\mathbb{S}^e = \{\mathbb{S}_1^e, \mathbb{S}_2^e, \ldots, \mathbb{S}_k^e\}, k = \lvert\mathcal{E}\rvert$, and $\mathbb{S}_n^e$ represents the link state of the n^{th} link
A	The action set, $A = (a_1, a_2, \ldots, a_l), l = N^2$
$\mathcal{F}$	The probability distribution of actions, $\mathcal{F} = (\rho_1, \rho_2, \ldots, \rho_l), l = N^2$
R	The reward set, $R = w_1 R_1 + w_2 R_2$
LD	The load state of MDCs, $LD_n = \sum_{\mu=i}^{j} \mathbb{S}_\mu^v$, where $(\mathbb{S}_i^v, \ldots, \mathbb{S}_j^v)$ represent the states of VMs within n^{th} MDC
$rpt^{\varphi_{ij}}$	Residual processing time of VNF request φ_{ij}
rtt^{S_i}	Residual transmission time of S_i
$c_n^{\varphi_{ij}}$	Boolean variable that equals 1 if VNF request φ_{ij} is cached on the n^{th} VM and 0 otherwise
$T_{\varphi_{i,j}}$	Scheduling latency of VNF $\varphi_{i,j}$

residual transmission time of S_i if S_i is being transmitted on the link and 0 otherwise.

$$\mathbb{S}_n^e = \sum_{i=1}^{|S|} x_n^{S^i} . rtt^{S_i} \tag{12}$$

2) ACTION

The design of action needs to achieve two goals: one is that the agent can completely control the scheduling process through actions without human intervention; another is that the actions should be efficient and without ambiguity so that the agent can update the action strategy to reduce the scheduling latency. Here, we begin to describe the actions of the agent. Once the previous VNF φ_{ij-1} has been processed, the SFC S_i needs to be sent to the next VM to process φ_{ij}. There are two issues to consider: timing selection and destination VM selection. At each timeslot, we stipulate the agent to send the waiting SFCs as soon as possible, which is proved to be an effective heuristic strategy. The timing strategy will be further discussed in section VI. Based on the above strategy, only the choice of destination VM needs to be considered at each step. Therefore, we define the selection of the destination VM as an action, which can be intuitively represented as a tuple (VM_s, VM_d). However, taking the VM pairs as actions may result in a huge action space, which will make it difficult to learn the ideal action strategy for agents. Therefore, according to Eq. (5)-(6), we replace tuple (VM_s, VM_d) with tuple (MDC_s, MDC_d) to reduce the action space without ambiguity. Moreover, we number the action tuples and define $A = (a_1, a_2, \ldots, a_l)$ as the action set. It should be mentioned that $\mathcal{F} = Q(\mathbb{S}, A)$ is symbolized as the probability distribution to take actions, which represents the priority of each action.

3) REWARD

The reward is generated until the end of the transmission action. Reward R is defined as a weighted sum of three factors:

(1) Whether the VNF φ_{ij} is processed timely. For each VNF φ_{ij}, the processing delay on the VM is a fixed value. The transmission delay, propagation delay and queueing delay of VNF φ_{ij} are determined by action selection. $T_{\varphi_{i,j}}$ is defined to represent scheduling latency of VNF $\varphi_{i,j}$. We use $\xi . d_{\varphi_{ij}}^{pro}$ as the base scheduling latency for VNF $\varphi_{i,j}$ and calculate R_1, where ξ is an empirical value obtained in experiments.

$$T_{\varphi_{i,j}} = d_{\varphi_{ij}}^{tran} + d_{\varphi_{ij}}^{prop} + d_{\varphi_{ij}}^{proc} + d_{\varphi_{ij}}^{qdp} \tag{13}$$

$$R_1 = \xi . d_{\varphi_{ij}}^{proc} - T_{\varphi_{i,j}} \tag{14}$$

(2) Whether the destination VM is a scheduling bottleneck. We observe the state migration in a VM granularity. At the end of each action, we consider whether the destination VM becomes a scheduling bottleneck. R_2 is obtained by comparing the state of destination VM named $\mathbb{S}_d^v$ with the states of other VMs. If the destination VM is not a scheduling bottleneck, the agent will get a positive reward.

$$R_2 = \max_n \mathbb{S}_n^v - \mathbb{S}_d^v, n \in [1, |\mathcal{V}|] \cap n \neq d \tag{15}$$

(3) Load balancing between MDCs in the system. Load imbalance of MDCs will cause transmission congestion, which affects the total scheduling time. We define $LD_n = \sum_{\mu=i}^{j} \mathbb{S}_\mu^v$ to symbolize the load state of the n^{th} MDC, where $(\mathbb{S}_i^v, \ldots, \mathbb{S}_j^v)$ represent the states of VMs within n^{th} MDC. R_3 is equal to the standard deviation of the load state set of MDCs.

$$R_3 = \sqrt{\frac{1}{N-1} \sum_{n=1}^{N} \left(LD_n - \bar{LD}\right)^2} \tag{16}$$

B. ADAPTIVE SCHEDULING PROCESS

The pseudocode of the adaptive scheduling process is described in Algorithm 1. Firstly, we initialize the MEC environment and group the SFCs according to their distance from the MDC. Each group of SFCs selects the corresponding MDC as the access point. We assume that the first VNF of each SFC can be processed in the access MDC. Secondly, we prioritize SFCs according to the number of requests which means the SFC with more VNFs has a higher priority. Next, we steer SFCs to the VMs deployed the corresponding VNFs according to the priority. After that, the scheduling order of SFCs is completely determined by the agent.

State-action function Q is the brain of the agent. The transition process from $\mathbb{S}$ to $\mathcal{F}$ is divided into two steps. Firstly, we input the VM state $\mathbb{S}^v$ into dense layers and the link state $\mathbb{S}^k$ into convolutional layers. Secondly, we concatenate the extracted features and feed them into dense layers to calculate $\mathcal{F} = Q(\mathbb{S}, A)$.

For each timeslot, the agent selects a random strategy $\hat{\mathcal{F}} = \tilde{\mathcal{F}}$ with probability ε or selects strategy $\hat{\mathcal{F}} = Q(\mathbb{S}, A)$ with

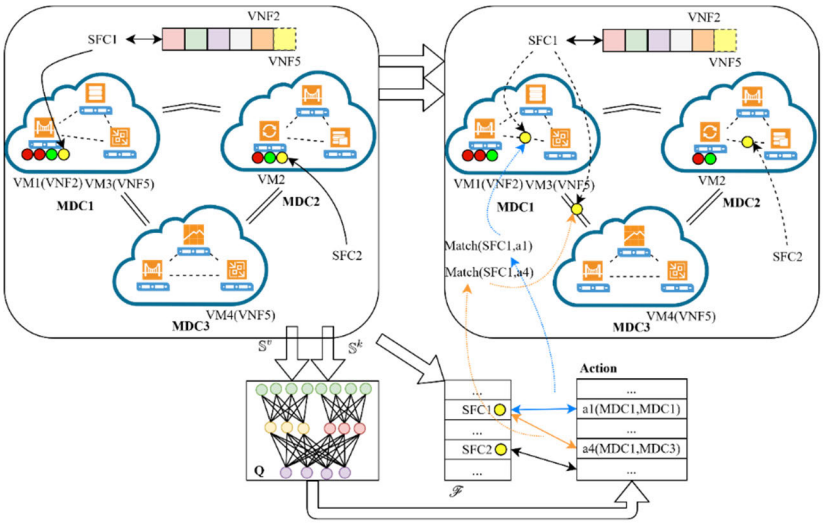


FIGURE 3. Action matching in adaptive scheduling process.

Algorithm 1 Adaptive Scheduling Process

1: *Initiate MEC system state and steer SFCs in access VMs*
2: **for** *t=1,T* **do**
3: *r= Rand()*
4: **if** $r < \varepsilon$ **then**
5: *Select random strategy* $\hat{\mathcal{F}} = \tilde{\mathcal{F}}$
6: **else**
7: *Select strategy* $\hat{\mathcal{F}} = Q(\mathbb{S}, A)$
8: **end if**
9: $A' = \hat{\mathcal{F}}(A)$
10: *Collect waiting SFCs in MEC, named as* S_W
11: *Sort*S_W
12: **for** S_i *in* S_W **do**
13: **for** a_j *in* A' **do**
14: **if** *Match* $(S_i, a_j) == True$ **then**
15: *Perform transmission action according to* (S_i, a_j)
16: *Store* $(\mathbb{S}, a_j)$ *in pre-memory*
17: **break**
18: **end if**
19: **end for**
20: **end for**
21: **for** $(\mathbb{S}^-, a^-)$ *in pre-memory***do**
22: **if** a^- *is done* **then**
23: *Delete* $(\mathbb{S}^-, a^-)$ *in pre-memory*
24: *Calculate* $R = w_1R_1 + w_2R_2 + w_3R_3$ *for* a^-
25: *Store* $(\mathbb{S}^-, a^-, \mathbb{S}, R)$ *in* $\mathbb{M}$
26: **end if**
27: **end for**
28: **end for**

probability $(1 - \varepsilon)$ corresponding to line 3∼8. In line 9, we prioritize the actions in A according to $\hat{\mathcal{F}}$ and figure out A'. From line 10 to line 11, we collect the SFCs waiting to be sent in the network, named as S_W, and sort them in descending order according to the number of unprocessed VNFs. Next, the agent takes actions by matching $(\{S_W\}, A')$ and store two-tuples $(\mathbb{S}, a)$ in pre-memory corresponding to line 12∼20, which will be described by a concrete example. Finally, for each tuple $(\mathbb{S}^-, a^-)$ which the corresponding action a^- is done, we expand it into four-tuple $(\mathbb{S}^-, a^-, \mathbb{S}, R)$ and store the new tuple in $\mathbb{M}$.

To illustrate the process of action matching more clearly, we display an example in Fig. 3. In Fig. 3, we use circles to represent SFCs and use different colors to represent the state of SFC. Red represents the blocked state, green represents the processing state and yellow represents the waiting state. At the current timeslot, S_1 and S_2 are waiting to be sent. We take S_1 as an example to illustrate the process of action matching. There are three necessary conditions for a successful action matching between a waiting SFC and an action. A matching is effective if the first element of the action tuple corresponds to the MDC where the SFC is located, and the second element of the tuple corresponds to the MDC where deployed the required VNF. Moreover, the required transmission link is not occupied. The agent calculates the action strategy $\mathcal{F}$ according to $Q(\mathbb{S}, A)$. For S_1, both a_1 and a_4 meet the first and the second conditions. Finally, the agent executes a_1 to send S_1 because a_1 has a higher priority than a_4 according to $\mathcal{F}$.

C. AGENT TRAINING PROCEDURE

The pseudocode of agent training procedure is described in Algorithm 2.

From line 1 to line 3, we initialize neural network Q, replay memory $\mathbb{M}$, and target neural network $\tilde{Q}$. Model training and adaptive scheduling are carried out simultaneously, which is illustrated in Fig. 4.

An episode in the training model is a complete scheduling process for all the SFC requests. In each episode, we firstly initiate the MEC system and SFCs placement. From line 7 to

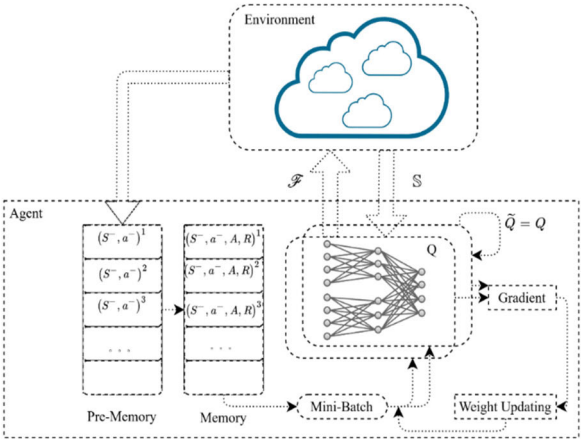


FIGURE 4. Agent training procedure.

Algorithm 2 Agent Training Procedure

1: *Initialize neural network* Q *with random weights* θ
2: *Initialize replay memory* $\mathbb{M}$
3: *Initialize target neural network* $\tilde{Q}$ *with weights* $\tilde{\theta} = \theta$
4: **for** *episode* $=1$, P **do**
5: *Initiate MEC system state and steer SFCs in access VMs*
6: **for** $t=1$, T **do**
7: *Calculate* $\hat{\mathcal{F}}, A', S_W$
8: *Take actions by matching* (S_W, A')
9: *Store tuples* $(\mathbb{S}^-, a^-, \mathbb{S}, R)$ *in* $\mathbb{M}$
10: *Sample a random mini-batch* $\tilde{\mathbb{M}} \subseteq \mathbb{M}$
11: **for** $(\mathbb{S}^{\tau-1}, a^{\tau-1}, \mathbb{S}^{\tau}, R^{\tau})$ *in* $\tilde{\mathbb{M}}$ **do**
12: $\theta = \theta + \alpha(R + \gamma \max_{a_\tau} \tilde{Q}\left(\mathbb{S}^{\tau}, a^{\tau}; \tilde{\theta}\right) - Q(\mathbb{S}^{\tau-1}, a^{\tau-1}; \theta))\nabla Q(\mathbb{S}^{\tau-1}, a^{\tau-1}; \theta)$
13: **end for**
14: **if** $t\%\eta == 0$ **then**
15: *Update* $\tilde{Q}$ *by* $\tilde{\theta} = \theta$
16: **end if**
17: **end for**
18: **end for**

line 9, the agent takes actions based on observation of the environment and store the tuples (state, action, reward) into the memory $\mathbb{M}$, which is described in detail in Algorithm 1. According to the experience replay technique, the agent randomly samples a mini-batch $\tilde{\mathbb{M}} \subseteq \mathbb{M}$ to train Q corresponding to line 10∼13. After every η timeslots, we update $\tilde{Q}$.

VI. PERFORMANCE EVALUATION

In this section, we demonstrate the performance of DQS. Firstly, the simulation configuration is illustrated. Next, we compare the performance of DQS with four existing algorithms and analyze the factors affecting the scheduling latency. Finally, we discuss the influence of delayed action strategy on DQS, which has been mentioned in section V.

A. SIMULATION CONFIGURATION

In our model, we use a fully connected network topology containing five MDCs to simulate a town-scale MEC system. The detailed parameter settings are shown as follow:

TABLE 4. MEC parameter settings.

	Parameter	Setting
MEC system	Number of MDCs	5
	Number of VMs in each MDC	[5,10]
	Bandwidth within MDCs	100 Mb/s~ 200 Mb/s
	Bandwidth between MDCs	100 Mb/s ~ 200 Mb/s
	Propagation speed between MDCs	5 us/km
	Geographical distance between MDCs	5 km ~ 10 km
	VNF processing time	4 ms ~ 7 ms
SFC	SFC length	4~6
	Size of SFC	1 Mb ~ 2 Mb
	Number of SFCs	[10~50,20~100]

Number of VMs in each MDC: The technical report about the 5G radio access network [41] and MEC shows that the number of physical devices would not be very large. In the simulation, we set the number of VMs to 5 and 10 respectively, which means MDCs have different sizes of computing resources.

Bandwidth allocation: Bandwidth allocation scheme is one of the factors affecting total scheduling latency. In this paper, both bandwidth within MDCs and bandwidth between MDCs are set in the range of 100 M to 200 M.

Propagation speed between MDCs: The propagation speed of links between MDCs is set to 5 us/km.

Geographical distance between MDCs: The geographical distance between is set in the range of 5 km to 10 km, which is in line with town size.

VNF processing time: This parameter is determined by the computing capacity of VM. Here, we set VNF processing time in the range of 4 ms to 7 ms.

SFC length: The SFC length is chosen uniformly at random in the range of 4 to 7, which is the expected length of SFC in real deployments.

Size of SFC: The size of SFC is set in the range of 1 Mb to 2 Mb, which is the size of a common network stream.

Number of SFCs: For the scenario that each MDC is deployed with 5 VMs, we set the number of SFCs in the range of 10 to 50; for another scenario that each MDC is deployed with 10 VMs, we set the number of SFCs in the range of 20 to 100.

We use a Python-based framework Tensorflow to construct the architecture of DQS and its deep neural network. All experiments are conducted on a computer with Intel (R) Core (TM) i7 processor and 8GB memory. The statistics are the average results.

B. TRAINING CONFIGURATION

The parameter settings of model training are shown in TABLE 5. Among them, the exploration rate ε is represented

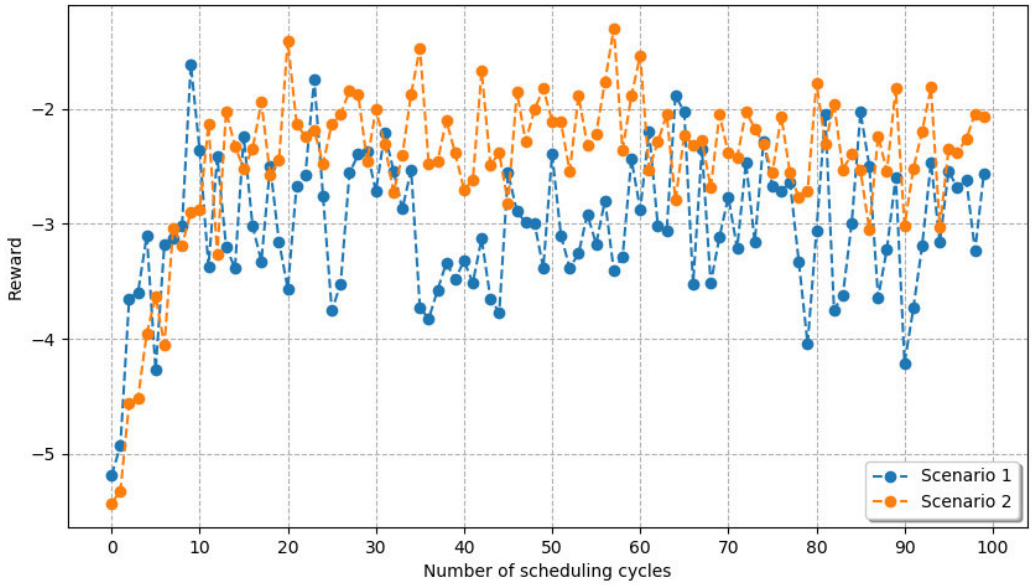


FIGURE 5. The rewards with different number of scheduling cycles.

TABLE 5. Model training parameter settings.

	Parameter	Setting
Training	Memory size $\mathbb{M}$	2000
	Discount rate γ	0.95
	Exploration rate ε	(1, 0.01,0.995)
	Learning rate α	0.001
	Update interval η	100
	ξ in R_1	2.5
	Weights of reward (w_1, w_2, w_3)	(1, 0.2, -0.2)

by a three tuple, with the first element representing the initial exploration rate, the second element representing the minimum exploration rate and the third element representing the decline rate for ε. The weights of the reward are the empirical values obtained from the experiments, which are set to 1, 0.2, -0.2 respectively.

The training process of the model is also the exploration process of the agent. The agent will observe the state from the environment and take action according $\boldsymbol{\varepsilon}$ greedy strategy combining exploration and exploitation. At the beginning of the training, we set the exploration rate $\boldsymbol{\varepsilon}$ to 1. After each agent training, we decrease the exploration rate with discount rate 0.995. Meanwhile, we set the minimum exploration rate as 0.01 to prevent the model from overfitting. The memory size $\mathbb{M}$ of agent is set to 2000. If the memory limit is exceeded, the old logs will be overwritten. The batch size of training is set to 32, which means model training begins after the number of logs in memory exceeds 32. For each 100 epochs of training, we will update the parameters of $\tilde{\boldsymbol{Q}}$.

Fig. 5 shows the evolution of the reward gained by the agent. Each MDC is deployed with 5 VMs in scenario 1 and each MDC is deployed with 10 VMs in scenario 2. Here, the ordinate represents the average reward of each scheduling cycle. We can find that the rewards gained by the agent gradually increase and eventually tend to be stable.

C. COMPARED ALGORITHM

In the experiment, the performance of DQS is compared with four other algorithms. We use a fast and effective algorithm, the greedy-based SFC scheduling algorithm (GFP) [36], as the baseline. The first step of GFP is to determine the deployment of SFCs according to the distance between virtual nodes and the second step is to determine the scheduling order of SFCs according to the number of unprocessed VNFs. In GFP, the complexity of the SFC deployment can be calculated as $O(|\mathcal{E}|.|S|.|T|)$ and the complexity of the scheduling order can be calculated as $O(|S|.|L|.|T|)$. Here, L represents the number of VNFs in the SFC and T represents the maximum tolerance for scheduling latency. Because $|L| \ll |\mathcal{E}|$, the total scheduling complexity can be represented as $O(|\mathcal{E}|.|S|.|T|)$.

The second algorithm is a Genetic Algorithm based SFC scheduling method (GA) [37]. In GA, two integer arrays O_1 and O_2 are used as chromosomes to represent the VNF assignment and the sequence of scheduling. In each iteration, the parental generations with shorter scheduling latency are selected to perform crossover and mutation operations. At the end of iterations, GA outputs the population with the shortest scheduling latency. In GA, the complexity of decoding is calculated as $O(|S|.|L|.|T|.|I|)$, where I represents the number of iterations. The complexity of crossover and mutation operations is $O(|S|.|L|.|I|)$. Hence, the total complexity can be calculated as $O(|S|.|L|.|T|.|I|)$.

The third algorithm is a matching based SFC scheduling algorithm (Matching) [39]. Matching can guarantee stable

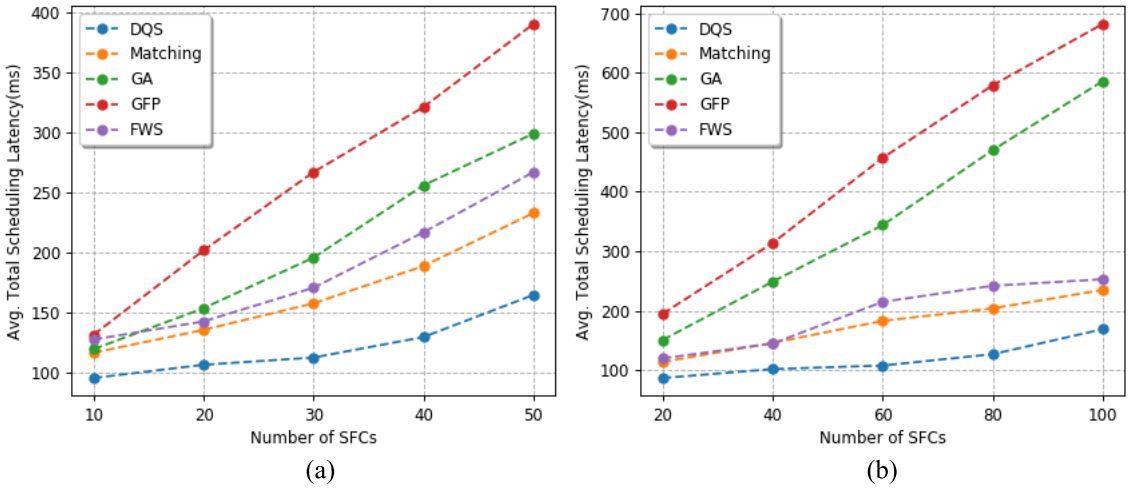


FIGURE 6. The performance of total scheduling latency.

scheduling, which means all resource nodes and VNFs are satisfied with the assignment. In Matching, the assignment of resource nodes at each timeslot is considered as an outcome of the one-to-one matching game. The complexity of matching progress can be calculated as $O(|\mathcal{E}|.|S|.|T|)$.

The fourth algorithm is a fair weighted scheduling method (FWS). In FWS, the VNF that needs to be executed first gets a priority as per the arrival time. The number of VNFs in SFC and the waiting time spent by the SFC in the queue are two other factors that determine the scheduling order. Meanwhile, the affinity between VNFs is taken into consideration in FWS. Two VNFs belonging to the same SFC are considered to have higher affinity and we try to place them on the same MDC. The complexity of priority computing can be calculated as $O(|S|.|L|.|T|)$. And the complexity of routing selection can be calculated as $O(|S|.|N|.|T|)$, where N represents the number of total VMs. Because $|L| \ll |N|$, the total scheduling complexity can be represented as $O(|S|.|N|.|T|)$.

The scheduling of DQS in each timeslot mainly includes computing of $\mathcal{F}$ and action matching. The computing of $\mathcal{F}$ is a linear mapping process with little time cost. The complexity of action matching can be calculated as $O(|\mathcal{E}|.|S|.|T|)$. Among all the scheduling algorithms, GA has a significantly higher time cost than other algorithms and other algorithms have similar time complexity.

D. PERFORMANCE EVALUATION

1) COMPARISON OF TOTAL SCHEDULING LATENCY

We first compare the performance of the five algorithms in total scheduling latency, which is the most important evaluation index. Fig. 6(a) describes the performance evaluation of average total scheduling latency among the five algorithms in the small-scale scenario, where 5 VMs are deployed in each MDCs and total number of VMs is 25. The number of SFCs is set to the range of 10 to 50. In DQS training, the number of SFCs for scheduling task is 30. The results show that DQS can adapt to the change of SFC number. With the increase of the SFC number, the advantages of DQS over other algorithms are gradually revealed. The scheduling latency generated by the five algorithms to process 10 SFCs is similar. However, the scheduling efficiency of DQS is 41.2 % higher than Matching, 81.2 % higher than GA, 136.4 % higher than GFP, 61.8% higher than FWS when the number of SFCs is 50. Fig. 6(b) shows the evaluation result in a large-scale scenario, where 10 VMs are deployed in each MDCs and total number of VMs is 50. We set the number of SFCs in the range of 20 to 100. In DQS training, the number of SFCs for scheduling task is 60. The results show that Matching is a relatively stable algorithm when the number of SFCs increases significantly. FWS has a similar performance in total latency with Matching. By contrast, with the rapid increase of solution space, GA is difficult to find a suitable scheduling scheme in limited time. When the number of SFCs reaches 100, the scheduling efficiency is 39.1% higher than Matching, 246.2% higher than GA, 303.6% higher than GFP, 49.7% higher than FWS. Since the trained agent in DQS can work well on different tasks, we speculate that DQS can be used as a feasible online scheduling approach for MEC system with fixed parameter settings.

2) COMPARISON OF VM UTILIZATION

In addition to the scheduling latency, the VM utilization is also an effective index to evaluate the efficiency of the scheduling algorithm. Fig. 7(a) and Fig. 7(b) show the VM utilization of each algorithm in two scenarios respectively, where the ordinate represents the average number of VMs in the working state at each time slot. The results show a positive correlation between the average VM utilization and scheduling efficiency. With the increase of arrival SFC numbers, both GA and GFP cannot make full use of VM resources. In the small-scale scenario, the average VM utilization of DQS is

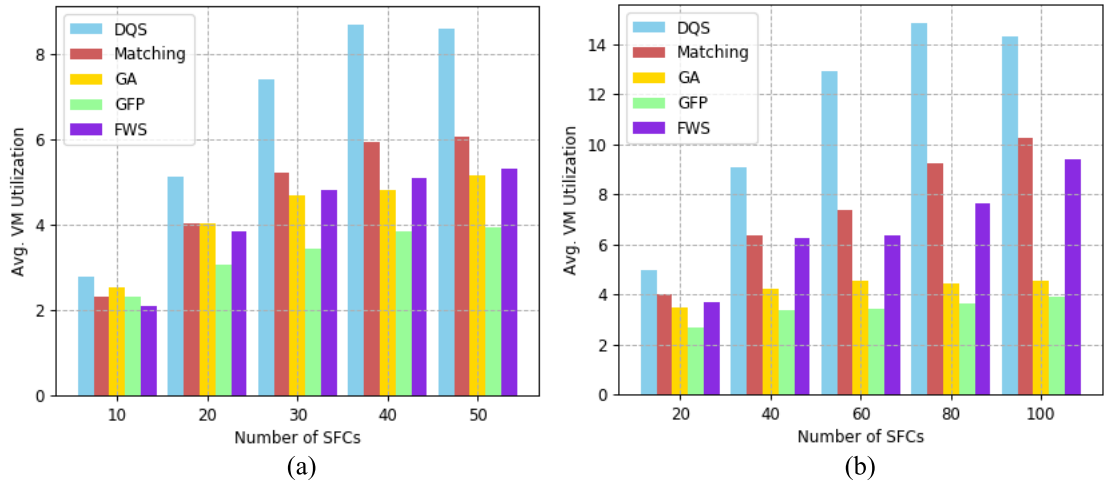


FIGURE 7. The performance of average VM utilization.

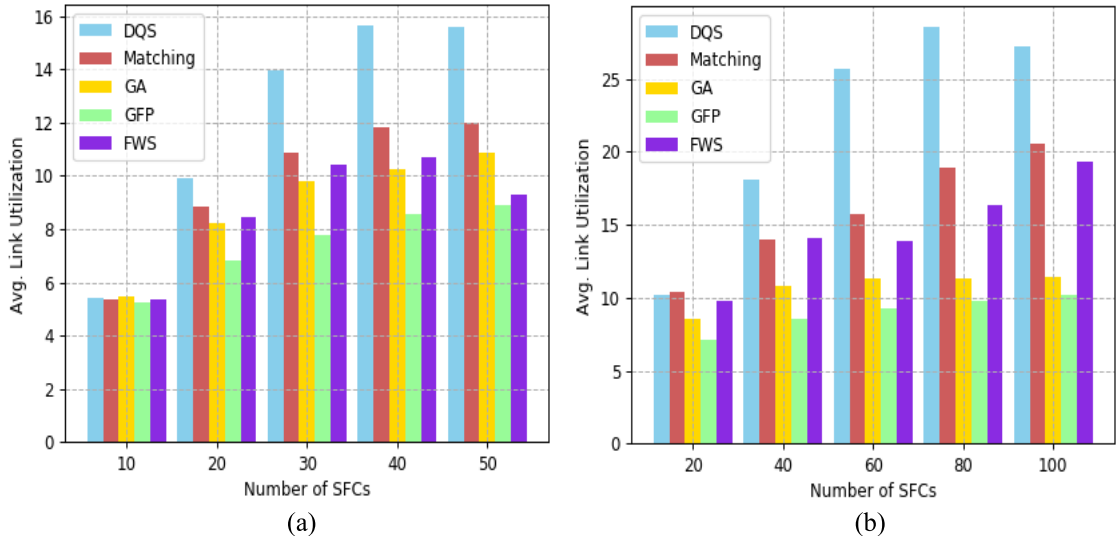


FIGURE 8. The performance of average link utilization.

41.4% higher than Matching, 66.6% higher than GA, 117.2% higher than GFP, 61.3% higher than FWS when the number of SFCs is 50. In the large-scale scenario, the average VM utilization of DQS is 39.5% higher than Matching, 216.1% higher than GA, 267.2% higher than GFP, 52.7% higher than FWS when the number of SFCs is 100. In DQS, we consider load balancing of VMs, which reduces the average queueing delay of SFCs. While in the other four algorithms, the SFCs are more likely to be allocated to the VM that contains more caching.

3) COMPARISON OF LINK UTILIZATION

Link utilization is an index closely related to queueing delay of transmission. Fig. 8(a) and Fig. 8(b) show the link utilization of each algorithm in two scenarios respectively, where the ordinate represents the average number of occupied links at each timeslot. The results show that the link utilization of DQS is higher than the other four algorithms. In the small-scale scenario, the average link utilization of DQS is 30.1% higher than Matching, 43.7% higher than GA, 75.6% higher than GFP, 67.3% higher than FWS when the number of SFCs is 50. And in the large-scale scenario, the average link utilization of DQS is 32.4% higher than Matching, 138.1% higher than GA, 167.9% higher than GFP, 41.4% higher than FWS when the number of SFCs is 100. DQS works better than other algorithms because load balancing among MDCs is considered, which effectively reduces the queueing delay of transmission caused by link congestion.

4) SCHEDULING BOTTLENECK ANALYSIS

To show the simulation results more clearly, we display a concrete example to visualize the scheduling process of

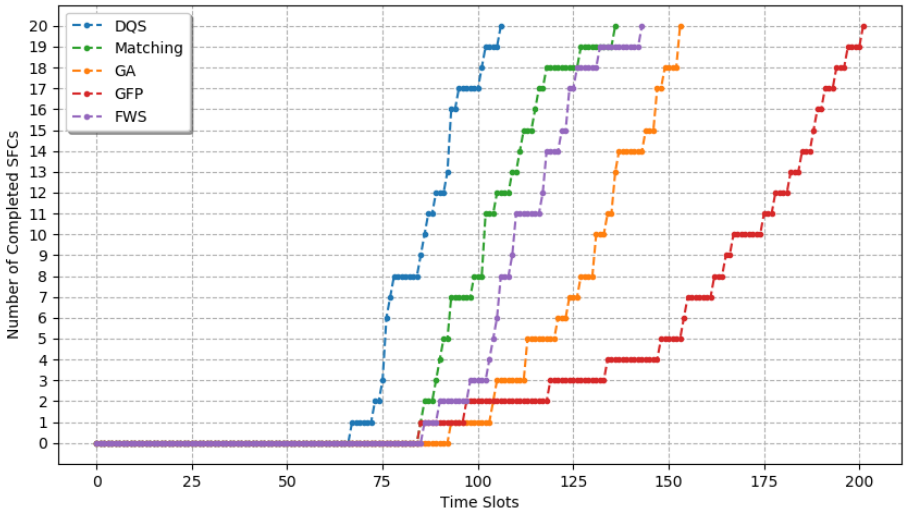


FIGURE 9. The number of completed SFCs.

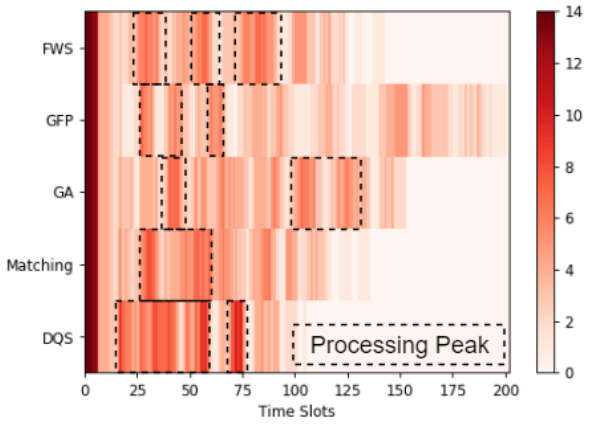


FIGURE 10. The number of SFCs in processing.

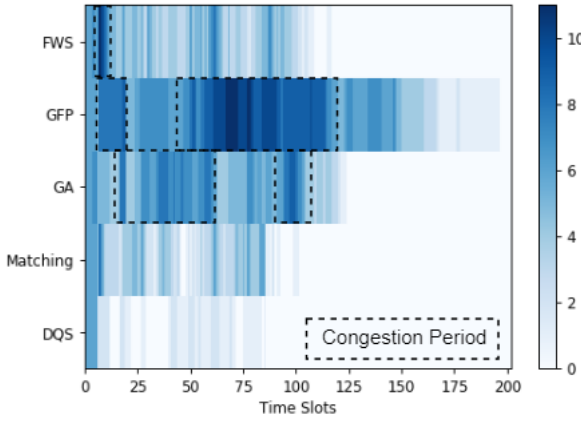


FIGURE 11. The number of SFCs in waiting.

each algorithm. In this example, each MDC contains 5 VMs and the number of SFCs is 20. Fig. 9 shows the progress of SFC completing of each algorithm. Fig. 10 and Fig. 11 are heat maps showing the SFC distribution in different states. In Fig. 10, the shade of red represents the amount of SFC being processed; in Fig. 11, the shade of blue represents the amount of SFC in the waiting state. We find DQS can percept the idle resources in the MEC environment. Compared with other algorithms, DQS reaches the processing peak earlier and seldom blocks. At the same time, we find that the congestion caused by an unreasonable scheduling sequence is the most important factor affecting efficiency.

5) INFLUENCE OF DELAYED ACTION STRATEGY IN DQS

As introduced in Section V, we adopt a timely sending strategy for the SFC in the waiting state. In fact, the timing for sending SFCs is an issue that may further improve scheduling efficiency. At the end of the simulation, we conduct an algorithm parameter analysis and prove that the delay strategy is indeed effective. Fig. 12 shows the results obtained in the large-scale scenario experiment. In the experiment, we randomly block a certain proportion of the SFCs waiting for sending at each timeslot. The results show that this strategy is effective when the number of SFCs is set to 80 or 100. And the total scheduling latency is reduced by 7 ms and 16 ms respectively. As shown in Fig. 7(b) and Fig. 8(b), the resource utilization of DQS increases when the number of SFCs is less than 80. At this point, the timely sending strategy is more efficient. Once the number of SFCs exceeds 80, the resource utilization of DQS reaches saturation. Delay strategy alleviates resource pressure and expands solution space, which is proven to be an effective sending strategy. In the future, we are going to explore the exact relationship between the delay ratio and the number of SFCs, which may further reduce the scheduling latency when the resource utilization reaches saturation.

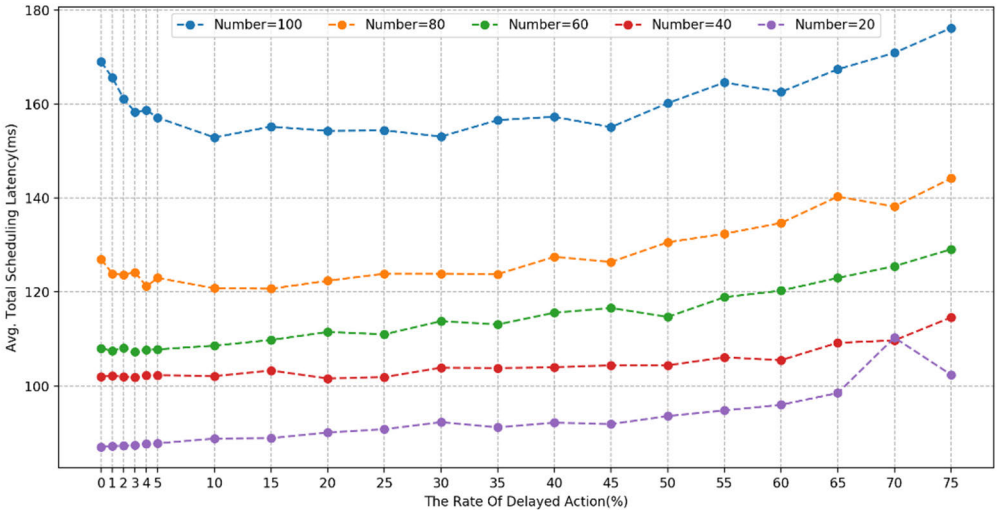


FIGURE 12. Comparison of total scheduling latency under different action delay ratios.

VII. CONCLUSION

In this paper, we study the SFC scheduling problem in MEC scenario. A deep Q-learning based approach DQS is proposed to implement dynamic scheduling. Compared with other existing algorithms, DQS obtains great performance improvement in total scheduling latency and resource utilization. Moreover, we show the difference of scheduling process between several algorithms in the form of heat map and analyze the factors affecting scheduling.

In practice, NFV implementation and VNF placement are relevant to SFC scheduling, i.e., resource allocation may constrain overall performance. In addition, some novel SDN management frameworks of MEC like LayBack [42] have been proposed, thus considering decomposition of SFC may improve the performance of DQS. Combining with the points mentioned above, we will expand DQS and apply it to an online scheduling scenario.

REFERENCES

[1] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 47–54.

[2] W. Xiao, W. Bao, X. Zhu, and L. Liu, "Cost-aware big data processing across geo-distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3114–3127, Nov. 2017.

[3] *Network Functions Virtualization*. Accessed: Oct. 17, 2014. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[4] *SDN Architecture Overview*. Accessed: Oct. 11, 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-111120-14.02.pdf

[5] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A software-defined framework for developing, deploying, and managing network functions," in *Proc. Conf. ACM SIGCOMM*, 2016, pp. 511–524.

[6] G. P. Katsikas, M. Enguehard, M. Kuźniar, G. Q. Maguire, Jr., and D. Kostić, "SNF: Synthesizing high performance NFV service chains," *PeerJ Comput. Sci.*, vol. 2, p. e98, Nov. 2016.

[7] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. J. Argyraki, S. Ratnasamy, and S. Shenker, "Resq: Enabling slos in network function virtualization," in *Proc. NSDI*, 2018, pp. 283–297.

[8] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9.

[9] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 643–655, Mar. 2019.

[10] J. F. Riera, E. Escalona, J. Batalle, E. Grasa, and J. A. Garcia-Espin, "Virtual network function scheduling: Concept and challenges," in *Proc. Int. Conf. Smart Commun. Netw. Technol. (SaCoNeT)*, Piscataway, NJ, USA, Jun. 2014, pp. 1–5.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[12] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Proc. 10th Int. Conf. Intell. Syst. Control (ISCO)*, Coimbatore, India, Jan. 2016, pp. 1–8.

[13] M. T. Beck, M. Werner, S. Feld, and T. Schimper, "Mobile edge computing: A taxonomy," in *Proc. Int. Conf. Adv. Future Internet*, 2014, pp. 48–54.

[14] D. Bhamare, M. Samaka, A. Erbad, R. Jain, and L. Gupta, "Exploring microservices for enhancing Internet QoS," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, p. e3445, Nov. 2018.

[15] Q.-V. Pham, F. Fang, V. Nguyen Ha, M. Jalil Piran, M. Le, L. Bao Le, W.-J. Hwang, and Z. Ding, "A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art," 2019, *arXiv:1906.08452*. [Online]. Available: http://arxiv.org/abs/1906.08452

[16] H. Tanaka, M. Yoshida, K. Mori, and N. Takahashi, "Multi-access edge computing: A survey," *J. Inf. Process.*, vol. 26, pp. 87–97, Feb. 2018.

[17] E. Ahmed and M. H. Rehmani, "Mobile edge computing: Opportunities, solutions, and challenges," *Future Gener. Comput. Syst.*, vol. 70, pp. 59–63, May 2017.

[18] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[19] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[20] Y. Zhao, W. Wang, Y. Li, C. Colman Meixner, M. Tornatore, and J. Zhang, "Edge computing and networking: A survey on infrastructures and applications," *IEEE Access*, vol. 7, pp. 101213–101230, 2019.

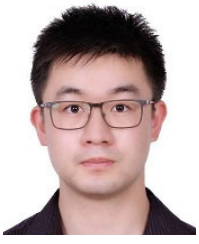
[21] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23511–23528, 2018.

[22] M. J. Kaur, "A comprehensive survey on architecture for big data processing in mobile edge computing environments," in *Edge Computing*. Cham, Switzerland: Springer, 2019, pp. 33–49.
[23] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, Aug. 2019.
[24] V. Nivethitha and G. Aghila, "Survey on architectural design principles for edge oriented computing systems," *J. Comput. Theor. Nanosci.*, vol. 16, no. 4, pp. 1617–1624, Apr. 2019.
[25] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: Architecture, applications, technical issues, and future directions," *Wireless Commun. Mobile Comput.*, vol. 2019, Feb. 2019, Art. no. 3159762.
[26] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.
[27] S. Bakiras, "Approximate server selection algorithms in content distribution networks," in *Proc. IEEE Int. Conf. Commun.*, May 2005, pp. 1490–1494.
[28] Y. O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady, "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, Jul. 2010, pp. 91–98.
[29] H. Goudarzi and M. Pedram, "Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems," in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, Jul. 2011, pp. 324–331.
[30] K. Su, L. Xu, C. Chen, W. Chen, and Z. Wang, "Affinity and conflict-aware placement of virtual machines in heterogeneous data centers," in *Proc. IEEE 12th Int. Symp. Auto. Decentralized Syst.*, Mar. 2015, pp. 289–294.
[31] D. Bhamare, A. Erbad, R. Jain, M. Zolanvari, and M. Samaka, "Efficient virtual network function placement strategies for cloud radio access networks," *Comput. Commun.*, vol. 127, pp. 50–60, Sep. 2018.
[32] F. L. Pires and B. Baran, "Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach," in *Proc. IEEE/ACM 6th Int. Conf. Utility Cloud Comput.*, Dec. 2013, pp. 203–210.
[33] L. Gupta, R. Jain, A. Erbad, and D. Bhamare, "The P-ART framework for placement of virtual network services in a multi-cloud environment," *Comput. Commun.*, vol. 139, pp. 103–122, May 2019.
[34] L. Gupta, R. Jain, M. Samaka, A. Erbad, and D. Bhamare, "Performance evaluation of multi-cloud management and control systems," *Recent Adv. Commun. Netw. Technol.*, vol. 5, no. 1, pp. 9–18, Dec. 2016.
[35] J. F. Riera, X. Hesselbach, E. Escalona, J. A. Garcia-Espin, and E. Grasa, "On the complex scheduling formulation of virtual network functions over optical networks," in *Proc. 16th Int. Conf. Transparent Opt. Netw. (ICTON)*, Piscataway, NJ, USA, Jul. 2014, pp. 1–5.
[36] 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. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Piscataway, NJ, USA, Apr. 2015, pp. 1–9.
[37] 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.
[38] H. A. Alameddine, S. Sebbah, and C. Assi, "On the interplay between network function mapping and scheduling in VNF-based networks: A column generation approach," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 4, pp. 860–874, Dec. 2017.
[39] 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.
[40] S. Tao, L. Gu, D. Zeng, H. Jin, and K. Hu, "Fairness-aware dynamic rate control and flow scheduling for network function virtualization," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 1–6.
[41] P. Marsch, I. D. Silva, O. Bulakci, M. Tesanovic, S. E. E. Ayoubi, T. Rosowski, A. Kaloxylos, and M. R. Boldi, "5G radio access network architecture: Design guidelines and key considerations," *IEEE Commun. Mag.*, vol. 54, no. 11, pp. 24–32, Nov. 2016.
[42] P. Shantharama, A. S. Thyagaturu, N. Karakoc, L. Ferrari, M. Reisslein, and A. Scaglione, "LayBack: SDN management of multi-access edge computing (MEC) for network access services and radio resource sharing," *IEEE Access*, vol. 6, pp. 57545–57561, 2018.

TIANFENG WANG received the B.S. degree in computer science and technology from Army Engineering University, Nanjing, China, in 2018, where he is currently pursuing the master's degree with the Department of Network Engineering. His research interests include network function virtualization, mobile edge computing, and satellite networks.

JIACHEN ZU received the B.S. degree in electronic science and technology from Shanghai Jiao Tong University, in 2017. He is currently pursuing the Ph.D. degree with the Department of Network Engineering, Army Engineering University, Nanjing, China. His research interests include network function virtualization, service function chain, and satellite networks.

GUYU HU received the B.S. degree in radio communication from Zhejiang University, Hangzhou, China, in 1983, and the M.Sc. degree in computer application technology and the Ph.D. degree in communications and information systems from the Nanjing Institute of Communication, Nanjing, China, in 1989 and 1992, respectively. In 1990, he devotes to the research on network management. Since 1997, he has been a Full Professor with Army Engineering University, Nanjing. His research interests include computer networks, maintenance and administration of the satellite networks, and intelligent network management.



DONGYANG PENG received the B.S. degree in computer science and technology from Army Engineering University, Nanjing, China, in 2018, where he is currently pursuing the master's degree with the Department of Network Engineering. His research interests include satellite networks and network management.