Reliability-aware Dynamic Service Chain Scheduling in 5G Networks based on Reinforcement Learning

Junzhong Jia†, Lei Yang†*, Jiannong Cao‡
† School of Software Engineering, South China University of Technology, Guangzhou, China
‡ Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China
Email: sejjz@mail.scut.edu.cn, sely@scut.edu.cn, jiannong.cao@polyu.edu.hk
*Corresponding Author

IEEE INFOCOM 2021 - IEEE Conference on Computer Communications | 978-1-6654-0325-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOM42981.2021.9488707

***Abstract*—As a key enabler of future 5G network, Service Function Chain (SFC) forwards the traffic flow along a chain of Virtual Network Functions (VNFs) to provide network services flexibility. One of the most important problems in SFC is to deploy the VNFs and schedule arriving requests among computing nodes to achieve low latency and high reliability. Existing works consider a static network and assume that all SFC requests are known in advance, which is impractical. In this paper, we focus on the dynamic 5G network environment where the SFC requests arrive randomly and the computing nodes can redeploy all types of VNF with a time cost. We formulate the problem of SFC scheduling in NFV-enabled 5G network as a mixed integer non-linear programing. The objective is to maximize the number of requests satisfying the latency and reliability constraints. To solve the problem, we propose an efficient algorithm to decide the redundancy of the VNFs while minimizing delay. Then we present a state-of-art Reinforcement Learning (RL) to learn SFC scheduling policy to increase the success rate of SFC requests. The effectiveness of our method is evaluated through extensive simulations. The result shows that our proposed RL solution can increase the success rate by 18.7% over the benchmarks.**

***Index Terms*—Service function chain, 5G network, reliability, reinforcement learning.**

I. Introduction

Recently, Software Defined Networking (SDN) and Network Function Virtualization (NFV) play important roles in network architecture evolution. Traditionally, a network function is implemented by dedicated hardware devices (middle boxes), which increases both the capital and operational cost of network service providers, meanwhile causing the problems of *network coupling*. When a new kind of service comes, relevant hardware devices must be deployed and connected by some order. This manual operation is extremely time-consuming, expansive and error-prone, preventing the operation of service additions and network upgrades. To address this challenge, NFV transfers the implementation of network function from dedicated hardware to software-based component named Virtual Network Function (VNF) [1]. VNFs are abstract network functions running on general commodity (e.g., x86 based systems) servers without specific hardware devices.

In NFV framework, an ordered combination of several VNF instances comprises a Service Function Chain (SFC) [2]. Traffic flows through different VNF sequences will establish multiple SFCs that can support various network services, such as firewall, load balancers, Deep Packet Inspection (DPI), Intrusion Detection System (IDS), and etc. NFV enables the virtualization of software-implemented network functions used in SFC. NFV is adopted by SFC to provide efficient and effective deployment and orchestration of network functions [3]. The IETF SFC working group (RFC 7665) and the Open Network Foundation (ONF) propose the SFC architecture specifications, demonstrating use-cases in operator networks, mobile networks, and datacenter networks.

The SFC scheduling problem has been widely discussed in many researches, but it is challenging to perform SFC scheduling due to the low latency and high reliability requirement in 5G network. In order to enhance the reliability of a network service, an SFC execution needs to have additional redundant VNF instances. The redundant instances occupy more computing resource and hence cause a higher waiting time for other network services. Consequently, it is essential that the SFC scheduler in 5G environment should be more intelligent to balance between latency and reliability.

In this paper, we consider online scheduling in a dynamic 5G network environment where the SFC requests arrive randomly. In this case, a computing node should switch its deployed VNF type to satisfy new types of VNF in the incoming SFC requests. However, the redeployment of various VNFs on a computing node is not cost-free; the node must shut down the previous VNF and set up the new VNF, which introduces additional redeployment time. Previously proposed network models assume that a node can only host a single type of VNF during all the time slots. They does not consider the redeployment of VNFs at the computing nodes. This static deployment strategy leads to an inefficient utilization of network resources under a dynamically varying SFC requests. Moreover, some existing studies consider the redeployment on a computing node but ignore the time cost of the redeployment. If redeployments occur frequently, a lot of time is spent on redeploying, increasing the latency of requests. Thus, the solution should balance the latency and the redeployment.

To solve the problem, we propose an efficient approach by separating the redundancy determining from SFC scheduling. For the former, we propose a heuristic algorithm to decide the redundancy of the VNFs while minimizing delay. For the latter, we develop a Reinforcement Learning (RL) method based on policy gradient to place these VNFs on computing nodes with the objective of maximizing the success rate of

978-1-6654-0325-2/21/$31.00 ©2021 IEEE

SFC requests. Further, we design a novel RL action different from existing studies that apply reinforcement learning to scheduling problems. In related works, the action determines which node should deploy the VNF, where the size of the action space is fixed to the number of computing nodes. The RL action in our method is to simply decide whether to defer the execution of the VNF. Thus, our RL model can scale to arbitrary number of computing nodes, without modifying the RL network model. The simulation result shows that our proposed method can increase the success rate by 18.7% comparing to the benchmark algorithms. The main contributions of this paper are summarized below:

- To the best knowledge of the authors, this paper is the first work to consider the reliability-aware SFC scheduling problem with latency constraint in dynamic network environment where the SFC requests arrive randomly. The computing nodes in this environment can host all types of VNF with a cost of redeployment time when switching the hosted VNF type.
- This paper formulates the reliability-aware scheduling problem as a Mixed Integer Non-Linear Programing (MINLP) problem, which is NP-hard and shows the complexity of the SFC scheduling problem and the difficulty to find a globally optimal solution.
- We propose a reinforcement learning algorithm to provide a feasible scheduling. The effectiveness of our approach is revealed through the comprehensive simulations. The results show that the proposed approach outperforms other benchmarks in term of SFC success rate.

II. Related Work

In recent years, the VNF placement and SFC scheduling has been set up in-depth research. [4] proposed an optimization model for minimizing the use of resources. For each experiment, they randomly placed VNFs and injected traffic based on day/night profiles. Every time this traffic profile is changed (12 hours), the mixed integer linear programming is rerun and the network is reconfigured. In [5], the authors proposed a dynamic VNF placement algorithm for SFC setup in a metropolitan area network. At each moment, a certain number of users request a specific SFC, and VNFs are placed to minimize the blocking probability.

The literature [6] provided an online algorithm for VNF scaling to dynamically set up network services in a data center network. [7] proposed a heuristic method for SFC reconfiguration, and compared its results with the best solution implemented in CPLEX. They do not consider the cost of reconfiguration due to loss of operator revenue due to reduced QoS when reconfiguring SFC. [8] proposed a merge algorithm based on the horizontal scaling technology, in which the processing capacity dedicated to VNF is increased/decreased by instantiating/deleting VNF without changing the processing capacity allocated to VNF.

Failure of network services can be normal and has a huge impact on the performance of applications; hence it is one of the critical problem in SFC scheduling [9]. In [10], the authors proposed a SDN-based replication framework for reliability-aware network services. Sherry et al. [11] developed a log-based recovery model for network services, which can be used to recover failed network services. Fan et al. [12] and Ye et al. [13] studied the problem of reliable service chain embedding, and proposed different heuristic algorithms based on both dedicated backup and shared backup provisioning. The literature [14] proposed a network-agnostic solution for network service backups based on bipartite matching. [15] discussed the SFC reliability problem in a dynamic network to minimize the total placement cost. However, it ignored the latency constraint of the SFC request.

The above efforts mainly focus on static SFC environment where all request information is known in advance, which is not feasible in real network. In this paper, we consider that the SFC requests can come randomly in dynamic network environment. Different from above work, our solution can adapt to real-world scenario. The goals of the current work mostly focus on optimizing network costs or considering only delay. To meet the needs of 5G network, our objective is to decrease the average end-to-end delay and guarantee the reliability at the same time. We develop a state-of-art algorithm based on Reinforcement Learning (RL) for performing dynamic SFC scheduling in low latency and reliability-aware 5G network.

III. System Model and Problem Formulation

In this section, we introduce NFV architecture defined by ETSI and SFC Mapping process in NFV framework. Next, we present the network model in this work.

A. NFV Architecture

VNFs are software modules abstract from the traditional hardware-based middle-boxes and require some physical resources (i.e. CPU, memory, bandwidth, etc.) [16]. They can be deployed on Virtual Machines (VMs) or containers hosted on commodity servers to provide scalable and elastic network services. Generally, there are three components in the NFV architecture: Services, NFV Infrastructure (NFVI) and NFV Management and Orchestration (NFV-MANO), as shown in Figure 1. A brief description of the components is as follows.

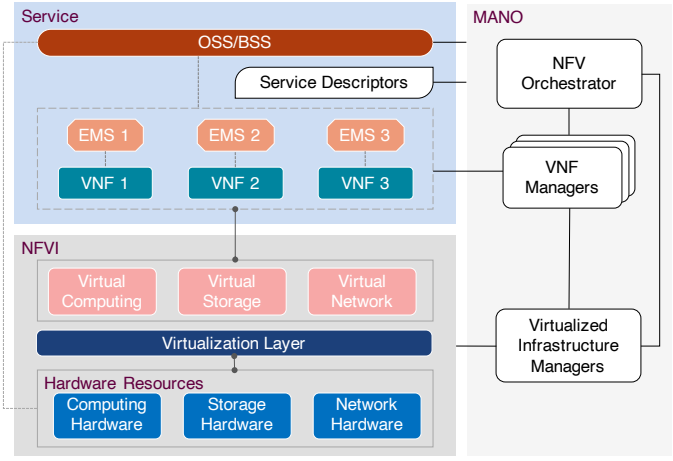


Fig. 1. ETSI NFV Architectural Framework.

Services. A service is a sequence of VNFs, which can be implemented in several virtual machines (computing nodes). A VNF is supervised by an Element Management System (EMS), in charge of its creation, configuration, performance and observation. Besides, EMS can supply the information required by the Operations Support System and Business Support System (OSS/BSS), which help telecom service providers to deploy and manage various telecommunications services [17].

NFVI. NFV Infrastructure is a set of resources used to host and connect VNFs. NFVI is a data center that includes servers, hypervisors, operating systems, virtual machines, virtual switches, and network resources, which is based on widely used and low-cost standardized computing components.

MANO. NFV Orchestrator is responsible for onboarding of new network services and VNFs, service lifecycle management and global resource management, etc. VNF Manager oversees lifecycle management of VNF instances, and coordinates EMS and NFVI. Virtualized Infrastructure Manager (VIM) controls the NFVI computing, storage, and network resources.

B. SFC Mapping

A network service is composed of several network functions. In NFV environment, these are virtualized as various VNFs. The SFC mapping problem aims at deciding where and when should the computing nodes deploy and process the given VNFs. The ETSI defines a network service as a sequence of several ordered VNFs [18], i.e., Service Function Chain (SFC). The packets should pass through a set of VNFs one by one, which are described in the SFC to complete an end-to-end network service. An example of SFC is shown in Fig.2. It presents a typical end-to-end network service $\{$*Firewall* $\rightarrow$ *Load Balancing* $\rightarrow$ *Encryption* $\rightarrow$ *Packet Inspection* $\rightarrow$ *Decryption*$\}$. The orchestrator is in charge of network service chaining and VNF placement. It accepts the network service request, transforms the request into an SFC, and maps the VNFs into the virtual machines provided by the virtualization layer. Physical Machines (PMs) are virtualized through a virtualized layer to provide abstract resources for VNFs.

C. Network Model

In this work, we consider a physical network deployed at the edge of 5G network. Due to the high bandwidth in 5G, the main limitation of network resource is computation capacity instead of the network transmission. Thus, we assume that the physical network is a fully connected network. This assumption is for abstracting details on routing and data transmission scheduling at the network layers, as we focus on the SFC scheduling problem at the application layer. In this case, the end-to-end delay can be defined as the sum of processing time and waiting time. The processing time of a VNF is calculated as: $p_f = w_f/V_k$, where w_f is the computation load of VNF f and V_k is the computation speed of computing node k. Note that the successor VNF cannot begin execution until the predecessor VNF is completed.

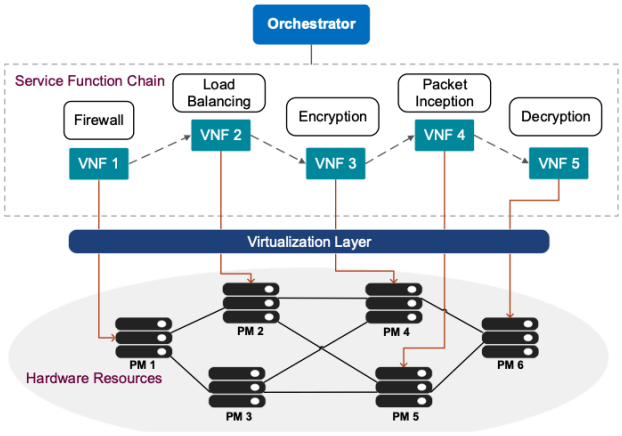


Fig. 2. Service Function Chain mapping in NFV Framework.

Reliability is a key performance metric in 5G network. The execution of a VNF may be interrupted since the hardware of software failures (e.g., unexpected restart/shutdown of a physical machine, network disconnection, software bugs, etc) [19]. We adopt the active-active redundancy scheme to increase the reliability, which deploys multiple instances of same VNF running concurrently. According to the reliability model defined by ETSI [20], the reliability of a VNF is the probability that at least one of the redundant subcomponents is available. We define θ as the reliability of a computing node, which specifies the probability of successful completion of a VNF on the node. Then, the reliability of a VNF is defined as:

$$r_f = 1 - \prod_{i}^{R_f}(1-\theta) \tag{1}$$

, where r_f is the reliability of VNF f and R_f is the redundancy of f. With respect to an end-to-end service, the end-to-end reliability is calculated as the product of the reliability of the VNFs comprising the SFC. Thus, the reliability of an SFC is:

$$R_i = \prod_{v \in s}\left[1 - \prod_{j}^{R_v}(1-\theta)\right] \tag{2}$$

, where v specifies one of the VNF in the SCF s and and R_v is the redundancy of v. In this case, the more redundant instances are instantiated, the higher reliability SFC is. However, an extremely large number of VNF instances in the network costs a log queue time, resulting in a greater end-to-end delay due to the limitation of network resources. Therefore, it is important to balance the trade-off between the redundancy and latency.

In this paper, we consider a Virtual Machine (VM) as a computing node, which can run on the physical server. For simplicity, we consider that each node can only host one VNF at a time. When an SFC request arrives, our problem is to decide the redundancy of each VNF and place these VNF instances on the nodes. A node can switch its type of VNF (i.e. redeployment) with a redeployment delay Δ. Further, we assume that all the nodes have the same reliability θ. To increase the reliability of a VNF, we can deploy multiple VNF instances called redundant VNFs. The redundant VNFs can run

TABLE I
TERMS IN MATH EXPRESSIONS

Terms	Explanation
N	The set of computing nodes, $k \in N$
S	The set of SFC requests
s_i	The i-th SFC request
s_{ij}	The j-th VNF of request s_i
F	The set of VNF types in S, $f \in F$
Θ_i	The reliability requirement of s_i
Φ_i	The end-to-end deadline time of s_i
d_i	The end-to-end delay of s_i
R_i	The reliability of SFC s_i
B_{ij}	The redundancy of VNF s_{ij}
w_f	The computation load of VNF f
V_k	The computation speed of node k
θ	The reliability of a computing node
T	The set of time slots
Δ	The delay of deploying a VNF instance
R_m	The maximum redundancy

on different nodes or rerun on the same node.

The notations used in this paper are shown in Table I. The time is divided into discrete time slots. Let N be a set represents all computing nodes in the network. Each node represents a Virtual Machine (VM), where the computation speed of a node k is V_k. An SFC consists of an ordered sequence of VNFs, where each VNF in a chain are different from each other. Let S be a set of SFC requests, where s_{ij} is the j-th VNF in the chain of request s_i. Note that the end-to-end delay of s_i should not exceed its deadline Φ_i.

The assumptions in our system model are as follows. First, the types of VNF in an SFC are different from each other. Second, SFC is a controlling flow including a sequence of VNFs. In an SFC, a VNF usually has very limited amount of data to the subsequent VNF. Thus, the data transmission delay can be ignored. Moreover, the network connecting the compute nodes has relatively abundant bandwidth. We assume that the computation cost and the redeployment cost on the resource constrained environment significantly impacts the performance, while the data transmission delay among two connective VNFs is ignored. Third, for simplicity, we consider that the first deployment delay of a computing node can be ignored. Further, each node can only host one VNF at a time. Last, the processing of a VNF should be continuous (non-preemptive), which cannot be interrupted by other VNFs.

D. Problem Formulation

Decision Variables. We use $\hat{x}_{ij}^{\delta k}$ to indicate that the computing node k starts to process VNF s_{ij} at time slot δ:

$$\hat{x}_{ij}^{\delta k} = \begin{cases} 1, & \text{node } k \text{ starts to process } s_{ij} \text{ at time slot } \delta \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

We also define an auxiliary variable to indicate VNF s_{ij} is processing on node k at time slot δ:

$$x_{ij}^{\delta k} = \begin{cases} 1, & \text{node } k \text{ is processing } s_{ij} \text{ at time slot } \delta \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The relation between $x_{ij}^{\delta k}$ and $\hat{x}_{ij}^{\delta k}$ is given as follows:

$$\frac{w_{ij}}{V_k} \sum_{\delta}^{|T|} \hat{x}_{ij}^{\delta k} \leq \sum_{\delta}^{|T|} x_{ij}^{\delta k}, \quad \forall i \in S, j \in s_i, k \in N. \tag{5}$$

$$\begin{gathered} x_{ij}^{\delta k} \geq \hat{x}_{ij}^{\delta' k}, \\ \forall i \in S, j \in s_i, k \in N, \delta, \delta' \in T : \delta \leq \delta' \leq \delta + \tfrac{w_{ij}}{V_k}. \end{gathered} \tag{6}$$

End-to-End Delay Constraints. The following constraints guarantee the end-to-end delay of a chain should meet its deadline requirement.

$$d_i = \sum_{\delta}^{|T|} \left(\delta + \frac{w_{i|s_i|}}{V_k} - 1 \right) \hat{x}_{i|s_i|}^{\delta k} \leq \Phi_i, \quad \forall i \in S, k \in N. \tag{7}$$

Reliability Constraints. The following constraint shows that the number of VNF redundant instances should ensure the SFC request reliability requirement.

$$\begin{gathered} R_i = \prod_{j}^{|s_i|} r_{ij} = \prod_{j}^{|s_i|} \left[1 - \prod_{k}^{|N|} \left(1 - \theta \sum_{\delta}^{|T|} \hat{x}_{ij}^{\delta k} \right) \right] \geq \Theta_i, \\ \forall i \in S, j \in s_i, k \in N. \end{gathered} \tag{8}$$

VNF Deployment Delay Constraint. There is a redeployment delay Δ when a computing node shut down its running VNF instance and deploy a different type of VNF instance.

$$\begin{gathered} \delta \hat{x}_{ij}^{\delta k} \geq (\delta' + \Delta + 1)\, x_{i'j'}^{\delta' k}, \\ \forall \delta, \delta' \in T : \delta' \leq \delta - 1, i, i' \in S, \\ j \in s_i, j' \in s_{i'} : f_{ij} \neq f_{i'j'}, k \in N. \end{gathered} \tag{9}$$

Sequence Order Constraint. The following constraint ensures the order of the VNF chain in request s_i. The VNF s_{ij} should complete when the next VNF s_{ij+1} starts to process.

$$\begin{gathered} \textstyle\sum_{\delta}^{|T|} \left(\delta + \frac{w_{ij-1}}{V_k} \right) \hat{x}_{ij-1}^{\delta k'} \leq \sum_{\delta}^{|T|} \hat{x}_{ij}^{\delta k} \delta, \\ \forall i \in S, j \in s_i, k, k' \in N. \end{gathered} \tag{10}$$

VNF Processing Constraint. Each VNF s_{ij} should be hosted in some computing node. The following constraint ensures that a VNF is instantiated more than once:

$$\sum_{k}^{|N|} \sum_{\delta}^{|T|} \hat{x}_{ij}^{\delta k} \geq 1, \quad \forall i \in S, j \in s_i, k \in N. \tag{11}$$

In this paper, we aim to maximize the number of successful SFC requests with given network resources. A request s_i is successful if its delay d_i does not exceed its deadline Φ_i. The objective is defined as follows, where $\mathbb{I}$ is *indicator function*.

$$\max \sum_{k}^{|N|} \mathbb{I}(d_i \leq \Phi_i) \tag{12}$$

The reliability-aware service function chain scheduling model is a Mixed Integer Non-Linear Problem (MINLP) with a series of constraints, which is very complex to solve. When we consider the offline scheduling, set redeployment delay $\Delta = 0$, ignore the SFC reliability requirement, and cancel the delay constraint, the problem in this paper can be reduced to a

Flexible Job-shop Scheduling Problem (FJSP). FJSP is proved to be NP-hard, and hence our problem is NP-hard. Thus, a global optimal solution is not to be obtained in polynomial time and we present a reinforcement-learning-based algorithm to generate a suboptimal solution.

IV. Solutions to Reliability-aware SFC Scheduling

According to [21], the basic SFC scheduling problem can be regarded as an extended Flexible Job-shop Scheduling Problem (FJSP) which is proved to be NP-hard. Moreover, the reliability-aware SFC scheduling problem with redeployment cost in a dynamic network is more complicated than the basic case. Hence it is too difficult to find an optimal solution in polynomial time. To simplify the problem, the reliability-aware SFC scheduling procedure is decomposed into two sub-problems. The first sub-problem is determining the optimal number of VNF redundancies to guarantee the reliability requirement. For this problem, we propose a heuristic approach to determine the optimal number of redundancies while avoiding high latency. The second sub-problem is mapping these VNFs into appropriate computing nodes with the goal of maximizing the number of successful SFC requests. We develop an intelligent RL-based algorithm to handle the SFC mapping problem.

A. Overview of Reliability-aware Dynamic SFC Scheduling Approach

To solve the problem, we propose a Reliability-aware Dynamic SFC Scheduling Approach (**RDSSA**). Fig.3 shows the workflow of this approach. It includes three critical steps. When there comes an SFC request, the **Redundancy Determining** is used to calculate the optimal redundancy of VNFs according to the required SFC reliability. It specifies which VNF in the SFC should be replicated and how many redundant VNFs should be deployed. These unscheduled VNFs are waiting in a queue. The priority of a VNF is based on the remaining time to the deadline in the corresponding SFC. The **Rule-based Approach for Node Selection** provides a strategy that selects an appropriate computing node for the VNF at head of the queue. In this process, we introduce a mechanism named deferred execution, which means the scheduler can defer the execution of an input VNF. The motivation is to reduce the computing resources consumed by the redundant VNFs. In this case, the **Reinforcement Learning for Dynamic SFC Scheduling** learns a deferring policy according to the change of network environment. If it decides to defer a VNF, the VNF is returned to the queue. Otherwise, the scheduler adopts the rule-based approach to allocate the node for the VNF.

B. Redundancy Determining

Note that the more redundant instances are deployed, the higher reliability SFC becomes. However, due to the limitation of computing capacity, end-to-end delay will increase as the number of redundancies grows. In order to obtain an optimal reliability-aware scheduling without exceeding the deadline, it is required to determine the minimum number of redundancies and decide which VNFs in the SFC should be replicated.

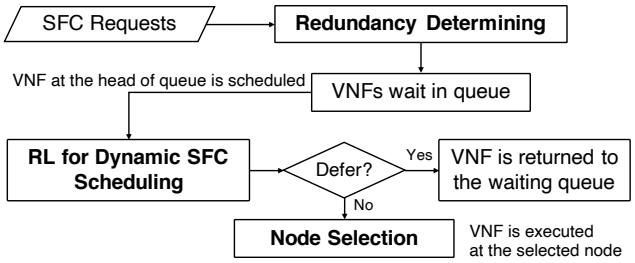


Fig. 3. Workflow of proposed RDSSA.

Algorithm 1: Redundancy Determining Algorithm

Input : $R_m, L, S = \{s_i\}, \theta, \Theta$
Output: `Redundancy(`s_i`)`: the redundancy of s_i

Set $a_1 ... a_L \leftarrow R_m$;
if `Rel(`$\{a_l\}$`)` $<\Theta$ **then**
  return *cannot satisfy the given reliability* ;
else
  Set $a_1 ... a_L \leftarrow 1$;
  Set $index \leftarrow 1$;
  while `Rel(`$\{a_l\}$`)` $<\Theta$ **do**
    $a_{index} \leftarrow a_{index} + 1$;
    $index \leftarrow (index + 1)\%L$;
  end
  $S' \leftarrow$ `AscSortByComputationLoad(`$\{s_i\}$`)`;
  for $i \leftarrow 1$ **to** L **do**
    `Redundancy(`s'_i`)` $\leftarrow a_i$;
  end
end

We propose an optimal redundancy determining algorithm as shown in Algorithm 1. Given that the maximum redundancy number of a VNF R_m, the length of the SFC L, the service function chain S, the reliability of the a computing node θ, and the reliability requirement of the SFC request Θ, the algorithm outputs a redundancy list A that indicates the number of redundancies for every VNF in the SFC, where $Redundancy(s_i)$ is VNF instances number of s_i, and $Rel(A)$ calculates the reliability of A. Recall that we consider all computing nodes have the same reliability θ during their executions. For example, if there is an SFC, of which VNFs and required reliability Θ are shown in Fig.4. The reliability of the a computing node θ is set to 0.96. The redundancy list A is initialized to 1, then increments from left to right in a loop until it meets the required reliability Θ. After that, we sort the VNFs by computation load in ascending order. In this case, the number of redundant VNFs corresponds to the number in the redundancy list. Algorithm 1 ensures that the difference between two number in the redundancy list A is less than 1, which can proved to be the maximum reliability when a fixed amount of redundancies is given.

Lemma 1: Given two redundancy a, b, where $a, b \geq 2$ and $a + b \equiv R_m$, if $0 \leq a - b \leq 1$ than $Rel(a, b)$ maximizes.

Proof: To prove it by contradiction try and assume that

the statement is false. Let $p = 1 - \theta$,where $0<\theta<1$, then $[1-(1-\theta)^a] \cdot [1-(1-\theta)^a] = (1-p^a) \cdot (1-p^b)$. Therefore,

$$\begin{aligned}(1-p^a) \cdot (1-p^b) &< (1-p^{a+1}) \cdot (1-p^{b-1}),\\ p^a(1-p) &< p^{b-1}(1-p),\\ p^a &< p^{b-1},\\ a &< b-1.\end{aligned}$$

Here, it arrives to a contradiction with $a - b \geq 0$. ■

Moreover, we prefer the VNF with a lower-load to have more redundancies because it costs a shorter processing time compared to the VNF with a high load. In this way, we use the minimum number of redundancies to guarantee the reliability requirement and reduce the extra processing time caused by the redundancies.

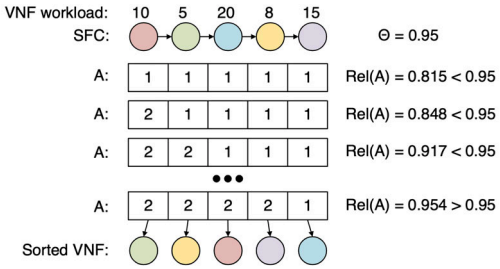


Fig. 4. An example for redundancy determining.

C. Rule-based Approach for Node Selection

Dispatching rules are used to select a computing node for a VNF. Existing studies propose some carefully designed rule-based algorithm. Zhang et al. [22] propose a priority-driven weighted algorithm to solve the VNF placement problem, which can find a near optimal solution cost-effectively. In [23], the authors develop a two-phased algorithm to minimize the total cost of VNF deployment. However, existing solutions to SFC scheduling are not applicable to the problem in this paper because they consider a static network and all SFC requests are known in advance. In this case, we consider the rule-based methods for FJSP, such as Earliest Finish First (EFF) and Earliest Start First (ESF). EFF selects the node that is able to finish the VNF earlier. ESF selects the node that starts with the VNF earlier. These simple rule-based approaches only provide feasible solutions, and we will propose reinforcement learning methods to improve their performance later.

D. Reinforcement Learning for Dynamic SFC Scheduling

We develops a Reinforcement Learning (RL) that uses a neural network referred to as the policy network to dynamically optimize SFC scheduling. The RL agent input the current state of the SFCs and outputs a scheduling action. Our RL framework is shown as Figure 5. A scheduling *agent* observes SFC requests and computing nodes *state* to decide a scheduling *action* under the NFV-based 5G network *environment*. The agent obtains a *reward* after performing an action, and updates the network parameters. Here, the agent uses an embedding network to encode the original state information vectors into low-dimensional features for RL networks.

Scheduling Events. Our scheduling model is task triggered. The unscheduled VNFs are waiting in a queue. The RL agent should make a scheduling decision when the VNF waiting queue is not empty. The behaviors of the queue are as follow.

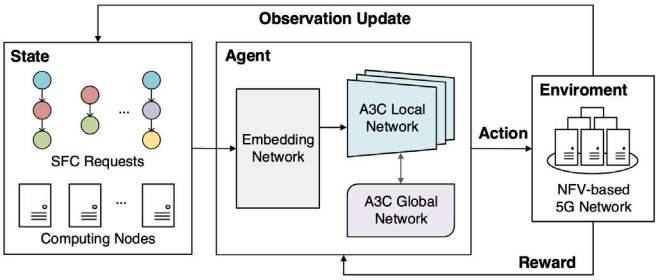


Fig. 5. Proposed reinforcement learning framework.

- The priority of the queue is based on the remaining time from the deadline. The VNF of an SFC with less time remaining has higher priority in the queue.
- The first VNF (with its all redundancies) of the newly arrived SFC will be added to the queue.
- When a VNF (with its all redundancies) is completed, its successor will be added to the queue unless the former is the last task. Furthermore, if all the redundancy of a VNF fails, we consider that the SFC of this VNF is aborted and subsequent VNFs will not be added to the queue.
- If one of the VNF redundancies is completed, the others will be removed from the queue, and if they have already been deployed on nodes, their executions will be aborted.
- VNF at the head of the queue will be scheduled first by the scheduler (or RL agent).

State Observation. State is the input of the RL agent. In this paper, the state consists of the VNF type, the number of VNF remaining in the chain, the length of the chain, the VNF computation load, the remaining length of the SFC, the remaining time to deadline, the VNF type that the node maintains, and the advanced time on each node. The VNF advanced time T_a is calculated as the difference between weighted VNF deadline T_w and the estimated completion time T_e, i.e., $T_a = T_w - T_e$. The estimated completion time T_e is defined as the sum of current time T_n, redeployment time T_δ and processing time T_p, i.e., $T_e = T_n + T_\delta + T_p$. The processing time of VNF i at node k is calculated as $T_p = w_{ij}/V_k$. The redeployment time is 0 when the type of VNF s_{ij} is the same as node k. Otherwise, there will be a redeployment delay Δ.

$$T_\delta = \begin{cases} 0, & \text{if VNF type } f_i \text{ is the same as type of node } k \\ \Delta, & \text{otherwise.} \end{cases} \tag{13}$$

The weighted VNF deadline T_w is defined as the computation-load-weighted average value of SFC deadline:

$$T_w = \frac{\sum_{u \leq j} w_{iu}}{\sum_{v \leq |S_{ij}|} w_{iv}} \Phi_i. \tag{14}$$

T_w indicates the expected completion time of a VNF. If a VNF is completed before its weighted VNF deadline, it is a proper scheduling. Otherwise, if the completion time is after its weighted VNF deadline, it implies that this SFC is lagging behind in execution and it may exceed the deadline in the end.

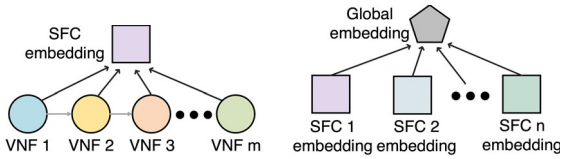
Action. Action is the output of the RL agent, which determines where the VNF should be deployed. Here, we

introduce a mechanism called *deferred execution*. When there is a VNF to be deployed, the scheduler or RL agent can *defer* this VNF until next scheduling event. The defer execution can make some redundant VNFs not be executed at first and stay waiting in the queue. At this time, the previously started VNF may be successfully completed and these deferred VNFs can be removed from the queue, reducing the computing resource occupied by the redundant VNFs as well as improving the success rate. With this motivation, we evaluate the impact of the deferred execution in SectionV-B. The results in Fig.7 shows that the deferred execution significant affects the success rate, which supports our motivation. However, the deferred execution with fixed probability cannot adapt to the dynamic available compute resources. Thus, we consider develop a method to provide a deferred execution with varying probability according to the dynamic environment. In this paper, we adopt a different action from other related work using reinforcement learning [2] [24]. Our network outputs a *defer rate*, i.e., the probability that the input VNF will be deferred. If a VNF is deferred, the scheduler will not allocate node to deploy it and the VNF will stay at the queue waiting next scheduling event. Otherwise, the VNF will be deployed at a node based on a rule-based approach. Importantly, with this action design, our agent can adapt to different number of computing nodes without changing the model.

Reward. Reward is the feedback from the environment after an action is performed. To guide the agent, we design a reward r based on the SFC scheduling objective. Each VNF must complete before its SFC deadline in order to increase success rate. We consider that if each VNF in a chain could complete before its VNF deadline T_w, the entire chain would satisfy the deadline; hence we use the VNF advanced time as the reward: $r = T_w - T_e$. When the term r is a negative number, we penalize the agent with a negative reward because of lagging completion of the VNF. While the term r becomes positive, the agent will be rewarded due to the VNF completion ahead of its VNF deadline. Note that, in the experiment, we bound the large-absolute-value reward to prevent the model from diverging caused by the large feedback signal.

E. SFC Embedding

RL agent converts the observation information into features to pass to the policy network. Traditionally, the feature vector is designed to contain all the state information. However, this approach cannot handle with the situation that both the amount of arrival SFCs and the length of SFCs are arbitrary. In this paper, we use a convolutional neural network called TextCNN [25] to encode the state information into embedding vectors to overcome the difficulty of arbitrary inputs and achieve scalability. TextCNN is a useful method for sentence classification tasks. We adopt it as an encoder for state information, which can compress the high-dimensional sparse vectors into low-dimensional dense vectors. Our embedding layer inputs the SFCs information and outputs two levels of embeddings, as shown in Fig.6. The first level is *SFC embedding* as shown in Fig.6(a) , which aggregates information about the whole chain of VNFs. The SFC embedding network transforms the per-VNF features in an SFC into a vector. The other level is *global embedding*, which summaries all SFCs information in the system. Similarly, the global embedding network inputs all SFC embeddings and outputs a global summarization vector. Note that, the size of SFC embedding vector is 16, while the size of global embedding vector is 64.



(a) SFC embedding. (b) Global embedding.

Fig. 6. Two levels of embedding networks.

F. Training

Our RL agent uses policy gradient for training, i.e., Asynchronous Advantage Actor-Critic (A3C). Unlike some techniques which are simply based on either value-iteration or policy-gradient, A3C combines the advantages of both the methods, which means A3C predicts value function V as well as the optimal policy function π. The value function is used to evaluate the action and the policy function is used to generate an optimal action. The agent use A3C algorithm to update the parameters θ of policy $\pi(a_i|s_i;\theta')$ and the parameters θ_v of value function $V(s_i;\theta'_v)$ [26] by the following equations:

$$\mathrm{d}\theta \leftarrow \mathrm{d}\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v)) \tag{15}$$

$$\mathrm{d}\theta_v \leftarrow \mathrm{d}\theta_v + \frac{\partial (R - V(s_i;\theta'_v))^2}{\partial \theta'_v}. \tag{16}$$

Here, R is time-discounted reward and $(R - V(s_i;\theta'_v))$ estimates how much better (or worse) the reward is in a step compared to the long term reward. $\nabla_{\theta'} \log \pi(a_i|s_i;\theta')$ indicates the direction to increase the probability of making action a_i at state s_i. Consequently, the Equation (15) teaches the agent to choose those actions that can obtain above-average reward. Equation (16) aims to reduce the difference between estimated reward value and actual reward. A3C makes use of multiple agents (or workers) with each agent having its own local parameters and a copy of the environment. These agents interact with their local environments asynchronously, and upload updates to the global network periodically. In this way, the global network can explore the parameter space more adequately. We implement A3C networks using 3 hidden layers, with 64, 32, 16 hidden units on each layer.

V. Evaluation

In this section, we first present the setup in our simulation. Then we conduct simulations to evaluate the performance of our proposed RDSSA for solving the SFC scheduling problem.

A. Simulation Setup

SFC simulator. We implement a Python-based SFC simulator and use PyTorch machine learning framework to build and train the agent networks. All the simulations are executed on a machine with Intel Core i9 2.3GHz and 32GB RAM.

Service function chains. According to [27], the length of SFCs in our simulation ranges from 2 to 6. Each VNF has its own computation load among {100, 150, 180, 200, 250, 300}, which refers to the amount of calculation. The reliability requirement of an SFC is randomly selected among {0.95, 0.99, 0.999, 0.9995}. The arrival of SFCs follows a Poisson process with an average interval of 15 time units.

Computing nodes. The process speed of computing nodes is randomly selected among {8, 12, 15, 18}. Each computing node has the same reliability $\theta = 0.96$, which indicates the probability of successful execution of an VNF at the node. The failure of a computing node can occur at any time throughout the VNF execution. We assume that the failure is one-off incident which would not affect subsequent task.

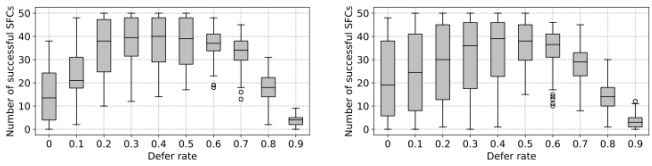
Baseline algorithms. Due to the similarity to the flexible job-shop scheduling problem, we choose two robust benchmark algorithms (i.e., Earliest Finish First and Earliest Start First), which are widely used in FJSP. Further, we implement a Deep Q Network (DQN) model as a comparison method.

- **Earliest Finish First:** EFF is a greedy algorithm that allocates the node that can finish it at the earliest time.
- **Earliest Start First:** ESF allocates a VNF to the computing node that can start the VNF at the earliest time.
- **Deep Q Network:** The DQN implementation of RL agent, only uses the value function to make actions. To ensure training convergence, we use the same embedding layers and reward function here as we do. The action of DQN follows the existing works [28] [29] that determines which VNF should be selected, instead of the defer rate.

In addition, we introduce a defer rate into EFF and ESF, which means that a VNF will be deferred at a certain probability.

B. Performance Results

Impact of the defer rate. We first evaluate the performance of EFF and ESF as the defer rate varies. The defer rate is the probability that the input VNF will be deferred. With low defer rate, the scheduler prefers to allocate nodes for the VNFs as quickly as possible. This results in a lot of redundant VNFs executing at the same time, which occupies a lot of computing resources. Moreover, with high defer rate, the scheduler will not execute redundant VNFs at first and can remove the redundant VNFs in the queue when the previously started VNF is successfully completed. This causes the redundant VNFs to be executed after the previous VNF fails, increasing the waiting time. Fig.7 shows the distributions of average number of successful SFCs over 100 experiments. For EFF, the appropriate defer rate should range from 0.3 to 0.5. For ESF, the average number of successful SFCs reaches a peak when the defer rate is around 0.5. It suggests that the defer rate plays an important role in the success rate when there are a lot of redundant VNFs in the network. However, the traditional approaches with constant defer rate cannot adjust the defer rate according to the dynamic available compute resources., which leads to a low success rate. In this case, we propose a learning-based approach that can provide a better defer rate as the network state changes.



(a) EFF with varying defer rate. (b) ESF with varying defer rate.

Fig. 7. The number of successful SFCs with different defer rate.

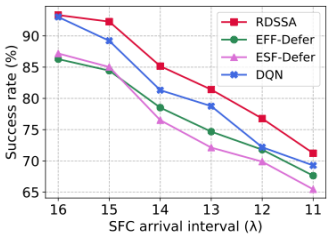


Fig. 8. Success rate with different SFC arrival intervals.

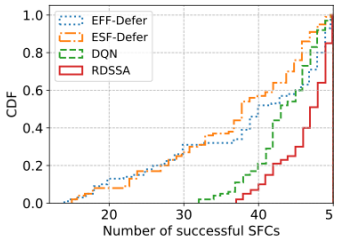


Fig. 9. CDF of the successful SFC.

Impact of the SFC arrival interval. SFC arrival interval denotes the average interval between two continuous SFC requests, which implies the intensity of the job arrival. To investigate the frequency of the SFC arrivals that influence the success rate of our proposed algorithm, we vary the request intervals and test the SFC success rate. Here, the redeployment time Δ is set to 150 time units. Note that, **EFF-Defer** (or **ESF-Defer**) is EFF (or ESF) method with deferred execution. The defer rate of EFF-Defer and ESF-Defer is set to 0.4 and 0.5 respectively, based on the maximum success rate shown in Fig.7. Fig.8 depicts the result and shows that the success rate decreases with the small interval. Apparently, the same computing nodes can provides a limited service capability to deploy VNFs. As the interval decreases, the density of the SFC request increases. In this case, the success rate goes down because the limited computing resources cannot afford large concurrent requests. The simulation shows that our proposed algorithm obtains far better performance than the benchmarks at all test intervals. The ESF-Defer and ESF-Defer defer the VNFs with a fixed probability cannot adapt to the varying request intervals. The action of DQN is to decide which node the VNF is executed at without the deferred execution. This causes a large amount of redundant VNFs are executed on the nodes, occupying computing resources and thus reducing the success rate. Further, as shown in Fig.9, RDSSA is more stable and has a higher average success rate than other benchmarks.

Impact of the number of computing nodes. We evaluate the performance with different number of computing nodes as shown in Fig.10. The result shows that the success rate of all methods increases with the increase of nodes. All methods can reach a high success rate when there are enough nodes. Note that, with the same arrival of SFC requests, the success rate is affected by the computing capability. The computing capability can be influenced by both nodes and scheduling policies. Obviously, with more computing nodes, the computing resources are sufficient for fixed number of SFCs, resulting in a higher SFC success rate. RDSSA can make full use of the computing resources by analyzing the information of the requests, reserve the required computing resources (certain

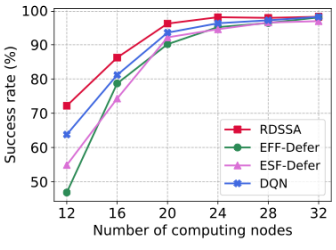


Fig. 10. Success rate with different number of computing nodes.

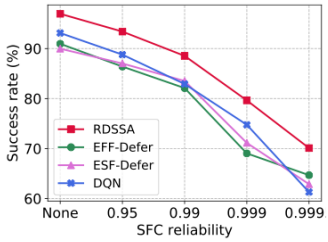


Fig. 11. Success rate w.r.t reliability.

VNF type of nodes), and allocate appropriate nodes to adapt to the request deadline, to achieve a high success rate globally. We can find that our proposed approach can learn a better policy when the number of nodes is relatively small.

Impact of the reliability. We next examine the impact of the required SFC reliability. To get a reasonable success rate, the arrival interval is set to 12 time units here. A better scheduling policy should defer the redundant VNFs wisely; that is to say, when there is already a VNF running on a computing node, its backups should be deferred to not occupy too much computing resources. Moreover, when the execution of the VNF fails, the redundant VNF should be deployed and be completed as quickly as possible because the time has been spent by the failed VNF. Fig.11 shows that our proposed approach outperforms other benchmarks, especially at the high reliability requirement (at 0.999). The result shows that when the required SFC reliability increases, the success rate decreases. This is because the higher reliability, the more redundant VNFs in the system, resulting in competition for computing resources, and eventually the SFC cannot be completed within the deadline.

Impact of redeployment delay. We further compare the success rate with different redeployment delay as shown in Fig.12. The computing nodes in our network environment can deploy all types of VNF with a time cost of redeployment when switching the hosted VNF type. To reduce the redeployment time cost is to enhance the efficient utilization of computing resources. The result shows that the success rate goes down as the redeployment delay increases. In the case of higher redeployment delay, the success rate of RL is improved more, which means RL agent can learn how to reuse the same type of VMs avoiding too much redeployment. Our method increases the success rate by at least 18.7% over rule-based methods at $\Delta = 120$.

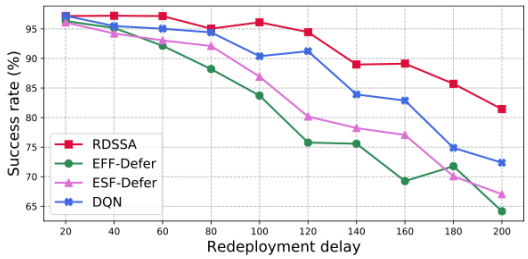


Fig. 12. Success rate with different redeployment delay.

Convergence of proposed method. At last, we compare our proposed method with DQN. Fig.13 shows the learning curves

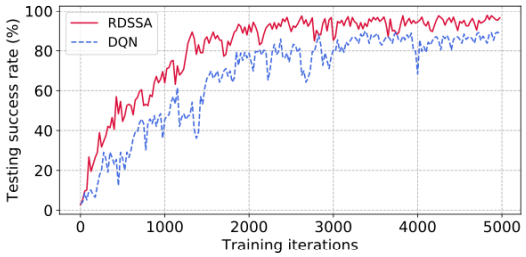


Fig. 13. The comparison of learning curve.

of the two models, testing the of models every 25 iterations. We can find that our method outperforms DQN in term of training efficiency. RDSSA consists of multiple independent workers (local networks) with their own parameters, using the technique of threads running on multicore CPUs in parallel. Each worker interacts with a copy of the environment, and updates the gradients periodically to the global/master network. Thus, these workers can explore a lager state-action space in less time. Further, unlike the traditional DQN model based on value-iteration methods, the model we used combines both the value function $V(s)$ and the policy function $\pi(s)$. The agent uses the output of $V(s)$ to guide the $\pi(s)$ by providing the state value, and updates the probability of choosing action in this state. The result shows that our method is better than DQN in both final success rate and convergence speed.

VI. Conclusion

This paper studies the reliability-aware SFC scheduling problem in NFV-based 5G network. To achieve the trade-off between high reliability and low latency, we formulate the problem as a MILP problem which is NP-hard. Our approach consists of two steps, namely redundancy determining and SFC scheduling. For former step, we develop an efficient algorithm to determine the minimum number of redundancies to guarantee the reliability without much processing time. In the latter step, we propose a scheduling algorithm based on reinforcement learning, as well as an embedding technique for encoding the features into low-dimensional vectors. Finally, the simulation results show the effectiveness of proposed approach in increasing the success rate of dynamically arriving SFCs, especially at a high redeployment time. As further future work, we would like to consider a multi-resource network environment (i.e., bandwidth, CPU and memory), and to solve the SFC scheduling problem within this real-life model.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61972161, in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2020A1515011496, in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2019B010154004 and in part by the Hong Kong RGC General Research Fund under Grant PolyU 15217919 and Grant PolyU 152133/18.

REFERENCES

[1] D. Zheng, C. Peng, X. Liao, and X. Cao, "Toward optimal hybrid service function chain embedding in multiaccess edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6035–6045, 2020.

[2] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao, "Service function chain embedding for nfv-enabled iot based on deep reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 11, pp. 102–108, 2019.

[3] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, 2016.

[4] M. R. Raza, M. Fiorani, A. Rostami, P. Ohlen, L. Wosinska, and P. Monti, "Benefits of programmability in 5g transport networks," in *2017 Optical Fiber Communications Conference and Exhibition (OFC)*, 2017, pp. 1–3.

[5] L. Askari, A. Hmaity, F. Musumeci, and M. Tornatore, "Virtual-network-function placement for dynamic service chaining in metro-area networks," in *2018 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE, 2018, pp. 136–141.

[6] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau, "Online vnf scaling in datacenters," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 140–147.

[7] T. Wen, H. Yu, G. Sun, and L. Liu, "Network function consolidation in service function chaining orchestration," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[8] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 255–260.

[9] R. Yu, G. Xue, and X. Zhang, "Qos-aware and reliable traffic steering for service function chaining in mobile networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2522–2531, 2017.

[10] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–15.

[11] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo *et al.*, "Rollback-recovery for middleboxes," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 227–240.

[12] J. Fan, M. Jiang, O. Rottenstreich, Y. Zhao, T. Guan, R. Ramesh, S. Das, and C. Qiao, "A framework for provisioning availability of nfv in data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2246–2259, 2018.

[13] 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 Network*, vol. 30, no. 3, pp. 81–87, 2016.

[14] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz, "Optimizing virtual backup allocation for middleboxes," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2759–2772, 2017.

[15] M. Karimzadeh-Farshbafan, V. Shah-Mansouri, and D. Niyato, "A dynamic reliability-aware service placement for network function virtualization (nfv)," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 318–333, 2019.

[16] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latré, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, 2016.

[17] J. Quittek, P. Bauskar, T. BenMeriem, A. Bennett, and M. Besson, "Network functions virtualisation (nfv)-management and orchestration," *ETSI NFV ISG, White Paper*, 2014.

[18] G. ETSI, "Network functions virtualisation (nfv): Architectural framework," *ETsI Gs NFV*, vol. 2, no. 2, p. V1, 2013.

[19] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in nfv-enabled enterprise datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 554–568, 2017.

[20] N. Isg, "Network functions virtualisation (nfv); reliability; report on models and features for end-to-end reliability," *ETSI GS NFV-REL*, vol. 1, p. v1, 2016.

[21] J. F. Riera, E. Escalona, J. Batalle, E. Grasa, and J. A. Garcia-Espin, "Virtual network function scheduling: Concept and challenges," in *2014 international conference on smart communications in network technologies (SaCoNeT)*. IEEE, 2014, pp. 1–5.

[22] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 731–741.

[23] L. Gu, J. Hu, D. Zeng, S. Guo, and H. Jin, "Service function chain deployment and network flow scheduling in geo-distributed data centers," *IEEE Transactions on Network Science and Engineering*, 2020, doi: 10.1109/TNSE.2020.2997376.

[24] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[25] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[27] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," IETF Draft, 2015.

[28] G. Li, H. Zhou, B. Feng, G. Li, and S. Yu, "Automatic selection of security service function chaining using reinforcement learning," in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018.

[29] X. Chen, Z. Li, Y. Zhang, R. Long, H. Yu, X. Du, and M. Guizani, "Reinforcement learning based qos/qoe-aware service function chaining in software-driven 5g slices," *arXiv preprint arXiv:1804.02099*, 2018.