Journal of Systems Engineering and Electronics
Vol. 35, No. 4, August 2024, pp.906 – 921

SFC placement and dynamic resource allocation based on VNF performance-resource function and service requirement in cloud-edge environment

HAN Yingchao[1], MENG Weixiao[1], and FAN Wentao[2,*]

1. School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin 150001, China;
2. China Mobile (Suzhou) Software Technology Company Limited, Suzhou 215000, China

Abstract: With the continuous development of network functions virtualization (NFV) and software-defined networking (SDN) technologies and the explosive growth of network traffic, the requirement for computing resources in the network has risen sharply. Due to the high cost of edge computing resources, coordinating the cloud and edge computing resources to improve the utilization efficiency of edge computing resources is still a considerable challenge. In this paper, we focus on optimizing the placement of network services in cloud-edge environments to maximize the efficiency. It is first proved that, in cloud-edge environments, placing one service function chain (SFC) integrally in the cloud or at the edge can improve the utilization efficiency of edge resources. Then a virtual network function (VNF) performance-resource (P-R) function is proposed to represent the relationship between the VNF instance computing performance and the allocated computing resource. To select the SFCs that are most suitable to deploy at the edge, a VNF placement and resource allocation model is built to configure each VNF with its particular P-R function. Moreover, a heuristic recursive algorithm is designed called the recursive algorithm for max edge throughput (RMET) to solve the model. Through simulations on two scenarios, it is verified that RMET can improve the utilization efficiency of edge computing resources.

Keywords: cloud-edge environment, virtual network function (VNF) performance-resource (P-R) function, edge resource allocation.

DOI: 10.23919/JSEE.2024.000092

1. Introduction

With the development of network technology and smart terminal devices, diversified network services and applications continue to emerge such as Internet of Things services [1], health monitoring [2], autonomous driving [3], virtual/augmented reality [4], and remote computing [5]. Meanwhile, Internet users are growing explosively, which will bring huge challenges to both the underlying data forwarding network and network service instances. However, the traditional network service provisioning and management paradigm cannot support such massive services. To overcome this limitation, several promising network innovations, such as Network functions virtualization (NFV) and cloud/edge computing, have been proposed and already implemented in the industry [6].

NFV technology has been considered as a key to solving the bottleneck of the traditional network since it was first proposed by European Telecommunications Standards Institute (ETSI) [7] in 2012. NFV adopts a software-based virtualization approach to replace traditional dedicated middleboxes with VNFs deployed on commercial on-the-shelf (COTS) devices. Based on NFV, a complex network service can be decomposed into multiple independent VNFs, and the user traffic only needs to pass through an ordered VNF sequence, called SFC [8], to complete the required network service. Meanwhile, with the support of software-defined networking (SDN) [9,10] and network slicing technology [11], NFV management and operation (MANO) have become more efficient and concise [12]. Hence, network service providers (NSP) prefer to deploy network functions in the form of VNFs, which causes explosive growth in demand for computing resources.

As a result, cloud service providers (CSP) have been widely deployed cloud and edge computing nodes in the network and provide strong computational power for countless VNFs. Users can purchase cloud and edge computing resources flexibly and conveniently to complete the required network services. Compared with cloud computing nodes, edge computing nodes are closer to users and have extremely low access delay [13]. On the

Manuscript received January 19, 2024.
*Corresponding author.
This work was supported by the Key Research and Development (R&D) Plan of Heilongjiang Province of China (JD22A001).

other hand, since the deployment location is close to users, the cost of edge computing nodes is much higher than cloud computing nodes, greatly limiting the deployment scale of edge computing nodes. Therefore, it is difficult for users to complete complex network services by purchasing only edge computing resources, but they need to purchase edge computing resources to collaborate with cloud computing resources. For users, how to deploy SFCs with limited edge resources and abundant cloud resources to maximize the utilization of edge resources purchased remains a considerable challenge.

In this paper, from the perspective of CSPs, we aim to maximize the utilization efficiency of limited and precious edge computing resources based on the network service requirements of NSPs in cloud-edge environments. Since the current definition of cloud-edge network is relatively vague, we adopt the view generally accepted by existing literature that a network containing both edge computing nodes and cloud computing nodes can be called a cloud-edge network [14]. This work has two vital innovations:

(i) We consider VNF placement and each SFC as a placement object, and determine whether it should be deployed in the cloud or at the edge. We formulate one new regulation for SFC placement in cloud-edge environments: VNF instances contained in the same SFC should integrally be deployed in the cloud or at the edge. As for the cloud-edge SFC in the actual situation, we believe that the main purpose of the cloud-edge SFC is to compress data at the edge servers to save bandwidth. In this paper, the cloud-edge SFC is divided into two separate SFCs for processing. In the existing researches of VNF placement in cloud-edge scenarios, the core decision-making problem is selecting the most suitable server for each VNF instance. However, the VNFs within one SFC may be deployed simultaneously in edge servers and cloud servers. In this situation, due to the extensive access delays of cloud servers, the overall delay of this SFC must be large, and the edge computing resources utilized by this SFC are wasted. Thus, we need a new regulation to avoid the situation. Moreover, the new regulation also brings a new optimized objective: selecting SFCs most suitable to deploy at the edge.

(ii) We introduce the resource consumption peculiarity of each VNF type into the consideration of resource allocation. We propose the VNF performance-resource (P-R) function to represent the functional relationship between the performance indicator of one VNF instance (e.g., throughput and processing delay) and the allocated resource for the VNF instance in a specific hardware environment. Through the P-R function, we can link resource allocation with the user requirements directly. According to the throughput and quality of service (QoS) requirements of user traffic, P-R function can help us to calculate the least computing resources required by each VNF instance in SFC. Unlike most existing VNF resource allocation studies where fixed computing resources are allocated to VNF instances based on VNF type, this work dynamically determines resource allocation by VNF P-R function and service requirements. This dynamic approach can prevent insufficient or redundant resource allocation caused by fixed approaches, making a big difference in resource-constrained edge scenarios.

Based on the two innovations, we build a network service slice model including cloud and edge computing resources, where we configure every VNF in the slice with its own particular P-R function. We take the utilization efficiency of edge computing resources purchased by users as the key performance indicator (KPI) and take the total throughput of the SFCs deployed at the edge abbreviated as edge throughput (ET), as the measure of utilization efficiency of edge computing resources and set the optimization objective to find a subset of all SFCs which can be deployed in edge server cluster and maximize ET. We name this optimization problem as max ET problem and prove this problem is a non-deterministic polynomical (NP)-hard problem. Algorithms with polynomial time complexity cannot obtain the optimal solution of the NP-hard problem. Therefore, we use the recursive algorithm to solve the max ET problem. Specifically, to simplify the max ET problem, we decompose this NP-hard problem into two sub-problems: how to place an SFC into the edge servers; find the optimal SFC subset satisfying the objective. Respectively, we design a heuristic algorithm based on VNF P-R function and a the recursive algorithm for max ET (RMET) algorithm to solve sub-problems. Moreover, we conduct simulation tests in two edge computing scenarios, and the results verify our main contribution: the RMET algorithm can obtain the optimal solution of the NP-hard problem to maximize ET within an acceptable time complexity in the small-scale scenario.

The remaining sections of this paper are organized as follows. In Section 2, we present related researches and discuss the novelty of our work. Section 3 introduces the research motivation and innovation. In Section 4 and Section 5, the system model, constraints, and objective formula are demonstrated. Finally, Section 6 contains the algorithm simulation and results analysis, while Section 7 concludes this paper.

2. Related work

This section presents the related research results of computing resource placement and dynamic resource alloca-

tion in cloud-edge environments. Moreover, we discuss the differences between the existing researches and this work.

2.1 Resource allocation in cloud-edge environment

In the network scenario containing cloud-edge computing resources, study of VNF placement and resource allocation study differs from that in traditional data center networks (DCN). After reviewing and evaluating the typical resource allocation researches in DCN [15], we deem that the resource allocation schemes for DCN cannot be directly adopted to the cloud-edge network since the differences between the edge and the cloud. Here, we present relevant research on the collaborative allocation of cloud and edge computing resources. Du et al. [16] designed a greedy heuristic algorithm to optimize computing task offloading in edge computing systems. Its optimization goal is to minimize the total cost of edge systems, including computing, communication and transmission costs. Huang et al. [17] proposed a cloud edge collaborative task offloading mechanism and designed a heuristic service scheduling algorithm to reduce the total service delay and energy consumption at the edge. Long et al. [18] proposed a cloud edge collaborative architecture to minimize the energy consumption of the cloud edge network, and designed a greedy annealing algorithm to obtain an optimized computing task offloading strategy. Slim et al. [19] designed a distributed low-cost service offloading algorithm. Edge nodes directly select feasible offloading nodes based on local information to offload services. The distributed offloading algorithm will centrally offload the services to one or several currently optimal offloading nodes, which is likely to cause network congestion and computational exhaustion of the offloading nodes. Peng et al. [20] optimized the computing offload of Internet of Things (IoT) service applications in cloud edge networks, aiming to reduce the computing delay and energy consumption of IoT service applications at the same time, and introduced constrained multi-objective evolutionary algorithms to solve the optimization problem. Wu et al. [21] constructed a cloud edge collaborative multitask computing offloading model. With the optimization goal of minimizing both delay and energy consumption, Wu et al. [21] used particle swarm optimization algorithm to solve the optimization problem. In [22], the branch and bound algorithm was introduced into the problem of computing task offloading. By optimizing task offloading, the computing and energy consumption of computing devices in cloud edge networks were minimized.

In [23], a general framework of 5G network slice, which jointly contains both cloud and edge servers with different amounts of resources, was proposed for the first time. This framework provides a fundamental model for the cloud-edge resource placement problem in the 5G slice network. Moreover, Zhang et al. [23] considered the interference between VNF inside the server and designs a heuristic placement algorithm to maximize the overall network throughput.

Randriamasinoro et al. [24] aimed to optimize the resource allocation problem in the cloud-edge network environment, which divided the server clusters that provide computing services into three categories: edge computing nodes, central clouds, and public clouds. The delays of three kinds of computing nodes depend on the distance between the node and the user. Randriamasinoro et al. [24] presented a completed formulation of the optimized resource allocation problem in the edge-cloud environment and designed algorithms to minimize the overall computing resource cost in the network while meeting QoS constraints.

Zhang et al. [23] and Randriamasinoro et al. [24] both took the problem of collaborative allocation of cloud and edge computing resources into consideration and achieved the optimization goal well. However, in these models, the computing resources required by different types of VNF are set as fixed values, and the relationship between the packet processing performance of VNF and the allocated computing resources is not considered. In this paper, we dynamically allocate computing resources to VNF as an important optimization method to improve the utilization efficiency of edge computing resources and reduce service delay.

2.2 Dynamic computing resource allocation for VNF

In many models related to the allocation of network computing resources, the computing resource required by a specific type of VNF instance is usually configured as a constant, which is quite helpful in reducing the complexity of modelling. However, in actual deployment, the computing resources of VNF instances are dynamically allocated by network managers. In NFV management platforms, such as OpenStack [25], the computing resource of any VNF can be customized.

Agarwal et al. [26] proposed the concept of flexible allocation of computing resources for VNF instances for the first time. In this paper, the m/m/1 service queue model represents the packet processing behavior of a VNF instance. The service rate of the queue corresponds to the packet processing rate of the VNF instance. Based on this, a heuristic algorithm is designed to optimize the VNF placement and improve the service QoS. However, Agarwal et al. [26] did not clearly define the relationship between the VNF processing rate and the allocated

resources but directly quantified the server's processing capacity by request/s and allocates it to different types of VNF. Although this suggests that the computing resources required by different types of VNF for processing one request are the same, the cost by different types of VNF instances for processing one request is different. Thus, we discuss this issue in Section 3 in detail.

Alameddine et al. [27] discussed dynamic task offloading and scheduling in edge computing. In [27], researchers set a minimum computing resource p_m^a for each application a in the edge server. The actual allocated computing resource of application a is p_a, a dynamic value that exceeds p_m^a. The processing delay of application a is calculated as the ratio of user workload to p_a. We can find that two types of applications with the same computing resources will have the same processing delay. Therefore, [26] and [27] have the same problem, which is not considering the differences in resource consumption rates of different applications. Hence, in this work, the functional relationship between the delay and the allocated resource of each VNF type is unique and only bound to the VNF type and hardware environment.

Li et al. [28] discussed the relationship between central processing unit (CPU) allocation and VNF processing performance, and designs the finedge platform, which can dynamically allocate computing resources for VNFs to achieve cost-efficient purposes. The focus of [28] is to utilize the least CPU core to achieve the best VNF processing performance. The working mode of the finedge platform is based on a state machine and real-time decision-making. From the user traffic entering the traffic monitoring mode, Finedge assigns CPU cores to VNFs in the corresponding SFC according to the traffic attribute. Our work is proactive, and VNF placement and resource allocation are performed based on known network service requirements, including all VNFs, SFCs, and QoS requirements.

2.3 VNF performance testing

How to measure the processing performance of VNF is the basis for linking VNF performance with allocated resources. In [28], researchers conducted a performance test for one VNF instance, basic monitoring, and the results indicate that the performance of basic monitoring improves with the increase of allocated computing resources. Van Rossem et al. [29] introduced an approach to VNF profiling that regards one VNF under test as a black box and proposes a performance metric to quantify the VNF performance precisely. Researchers test several VNF types and record the trend of packet rate and packet loss rate with different assigned computing resources. During the experiment, Van Rossem et al. [29] proposed the concept of saturated resources boundary. According to the test results, the VNF performance can hardly improve before the assigned computing resource reaches the saturated resources boundary but will increase significantly when the assigned resource exceeds the boundary. This concept provides a theoretical basis for one parameter in the proposed system model: VNF start-up resource.

3. Motivation and innovation

In this section, we present the motivation and innovation of this work in two aspects. We first stipulate an SFC placement regulation to avoid wasting the edge computing resource. Moreover, we propose the VNF P-R function as a basis for the dynamic allocation of computing resources to VNF instances according to the service requirements.

3.1 A regulation for SFC placement in cloud-edge environments

In network services, the most fundamental and vital optimization objective is to reduce service delay. The original intention of edge computing is to reduce user access delay by decreasing the distance between computing resources and users. Because of the much higher deployment cost, edge computing resources are more precious and limited.

In existing VNF placement researches, the presentations of optimization results are the mappings from VNF instances to servers. In cloud-edge environments, if we only focus on selecting servers for VNFs, it turns out that, in one same SFC, part of the VNF instances will be deployed in cloud servers and the others in edge servers. However, SFC crossing cloud and edge servers will cause a waste of computational resources. Here, we demonstrate the explanation with a simple cloud-edge scenario.

In the cloud-edge scenario shown in Fig. 1, the user's smartphone initiates a service request, and the SFC corresponding to this request is VNF1-VNF2-VNF3-VNF4. The four lines with different colors represent four forwarding paths of this SFC. For the red path, all VNF instances are deployed in edge servers. On the contrary, in the blue path, all VNF instances are in cloud servers. Green and orange paths contain VNF instances deployed in both cloud and edge servers. We use T to denote the transmission delay between the user device and the cloud server. To simplify the modeling complexity, we assume that the delays between servers in the same cluster (cloud/edge) are the same and VNF instances of the same type have the same processing rate. Then, we use delay-color to represent the total delay of different color paths and show the relationship among the overall delays of the four SFC paths:

$$\text{Delay}-\text{blue}=\text{Delay}-\text{red}+T,$$

$$\text{Delay}-\text{green}=\text{Delay}-\text{red}+T,$$

$$\text{Delay}-\text{orange}=\text{Delay}-\text{red}+2T.$$

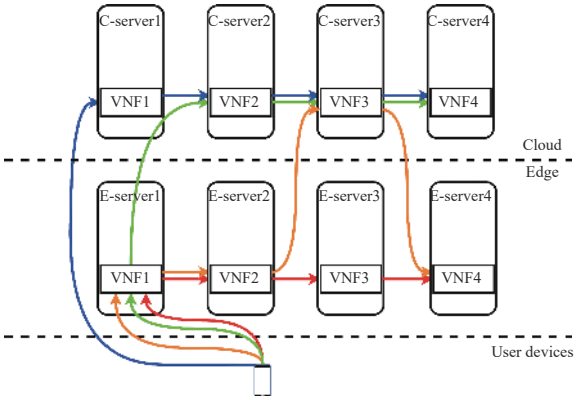


Fig. 1 Four forwarding paths in cloud-edge scenario

We can find that although the green and orange paths occupy part of the edge computing resources, their overall delays are still higher than or equal to the latency of the blue path. The high transmission delay between the cloud and the edge conceals the low-latency feature of edge computing.

Based on the above explanation, we set the following regulations: VNF instances in the same SFC should integrally be deployed in the cloud or at the edge.

In actual scenarios, there exist cloud-edge SFCs that need to be deployed on both cloud and edge servers. For such cloud-edge SFCs, we believe that the main purpose of the SFCs lies in saving bandwidth by firstly compressing data at edge servers. In this paper, we divide the cloud-edge SFC into two separate SFCs. As for the large-bandwidth SFC without data compression, the algorithm in this paper will deploy the unprocessed large-bandwidth SFC on edge servers to maximize ET. After being compressed by edge servers, the data is processed through the other SFC. Therefore, the processing results are consistent with the actual situation.

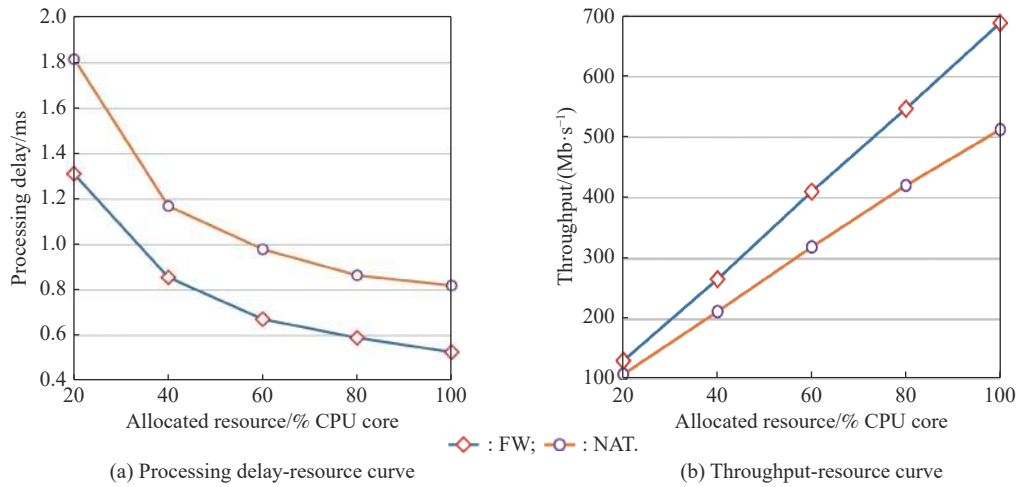
Based on this regulation, one main objective of this research is to select SFCs most suitable to be deployed at the edge.

3.2 VNF P-R

As mentioned in Section 2, in [26] and [27], VNF instances of different types with the same computing resources have the same processing rates. We deem that this setting does not conform to the actual working situation of VNF instances and propose the following new hypothesis:

Hypothesis 1 For every VNF instance, there is a functional relationship, $P=\text{function}(R)$, between its processing performance (P) and allocated computing resource (R), and this function is only determined by the VNF type and hardware environment.

Proof We conduct a verification experiment in a server with CPU Intel Xeon E5-2 609 v4. We test two common VNF types: firewall (FW) and network address translation (NAT). We create independent instances with one CPU core for both VNF types and use the same data plane development kit (DPDK) [30] settings to accelerate hardware forwarding. Then, we use the trex tool to test and record the latency and throughput of VNF instances with different CPU utilization rates. In this experiment, when a certain amount of computing resources is allocated to one VNF instance, the VNF performance will fluctuate within a stabilized interval and cannot be completely stabilized at a fixed value. Therefore, we take the performance lower limit of the interval as the value of the VNF performance under the CPU allocation situation, i.e., we take the upper limit in the delay test and the lower limit in the throughput test. The experimental results are shown in Fig. 2.



(a) Processing delay-resource curve (b) Throughput-resource curve

Fig. 2 Results of confirmatory experiment

Thus, the proof is complete. □

Results indicate that, with the same hardware environment and CPU allocation, VNF instances of two types have different processing delays and throughputs. With the increase in computing resources, the delay is gradually decreasing, and the throughput is gradually increasing. The curve of processing performance is smooth and can be approximately fitted to a continuous function within a specific interval. Moreover, this experiment proves that simple network test tools can test and measure the function in Hypothesis 1, $P = \text{function}(R)$.

The confirmatory experiment confirms that VNF instances of different types have different resource consumptions for processing one request. Therefore, we hope that the VNF type can be taken into consideration in terms of computing resource allocation. Based on Hypothesis 1, we propose VNF P-R function to assist the precise resource assignment, defined as Performance Indicator (f,ψ).

Performance Indicator refers to a certain VNF instance performance indicator, such as delay or throughput. f is the VNF type of the VNF instance, and ψ denotes the computing resource allocated to the instance. This function represents the performance indicator that an f-type VNF instance can achieve with ψ assigned computing resources.

P-R function accomplishes the link between the VNF performance and the allocated computing resource, and the VNF performance needs to meet the service requirements. Therefore, we can accurately allocate computing resources for each VNF instance according to the P-R functions of all VNF types in the service slice and the service requirements of all SFCs. In the subsequent sections, we discuss how to place VNF instances, allocate resources based on service requirements with the help of P-R Function, and determine whether each SFC should be deployed in the cloud or at the edge. Additionally, this paper considers two service requirements: throughput and delay QoS.

4. System model

We consider a cloud-edge network environment as depicted in Fig. 3, which consists of three parts: cloud, edge, and user devices. The network and computing resources of the cloud and edge are divided into multiple network slices. Each network slice supports one network service and simultaneously includes cloud and edge resources. In this section, the proposed system model focuses on VNF placement and resource allocation within one network service slice. We comprehensively define the hardware resource parameters and the network service parameters inside the network service slice and describe the mapping between the network service and the service deployment on hardware.

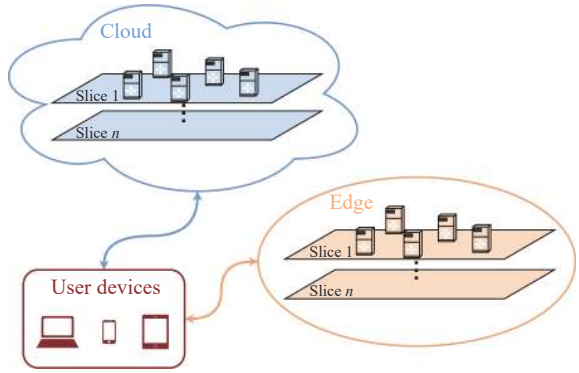


Fig. 3 System model diagram

We use an undirected graph, $G(S,L)$, to represent the network topology of the network slice where S is the set of all servers and L is the set of all links. For $S = S_c + S_e$, S_c and S_e represent the set of cloud servers and the set of edge servers, respectively. The set of all links is denoted by $l(u,v) \in L$ and $l(u,v)$ indicates the virtual link between server u and v. For each server $s \in S$ and each link $l \in L$, k_s represents the computing capacity of s; $b_l(u,v)$ represents the bandwidth of l. In this model, we default that the computing capacities of all servers are the same, i.e., k_s is a constant. The transmission delay of $l(u,v)$ is denoted by $\tau(u,v)$.

F represents the set of VNF types in the network slice. $f \in F$ represents a specific VNF type. This model considers the two most fundamental network service requirements: throughput and delay. We configure each VNF type with two exclusive P-R functions, $\text{Delay}(f,\psi)$, $\text{Throughput}(f,\psi)$, respectively representing the processing delay and throughput of the f-type VNF instance with ψ computing resource. For the deployed VNF instance, we use $\psi(f,s)$ to indicate the computational resource assigned to the f-type VNF instance in server s. The remaining unallocated computing resource of server s is denoted by k_s^r.

To simplify the model and distinguish the resource consumption features of different VNF types more clearly, in our model, $\text{Throughput}(f,\psi)$ and $\text{Delay}(f,\psi)$ are respectively approximated as linear functions and power functions, we have

$$\text{Throughput}(f,\varphi) = \alpha_f(\varphi - \beta_f), \tag{1}$$

$$\text{Delay}(f,\varphi) = \varphi^{\gamma_f}, \gamma_f \in (-\infty, 0). \tag{2}$$

In (1), α_f is the rate of throughput change with assigned computing resource, and β_f represents the start-up resource of f-Type VNF instance, i.e., the minimum computing resource required for f-type VNF instance to

start processing. The setting of parameter β_f is out of consideration for the compute-intensive VNF types, which cannot operate normally unless the assigned resource reaches the start threshold. As for lightweight VNF types, such as FW and NAT, β_f can be configured as 0. In (2), γ_f is the exponent in the power function, and $\gamma_f \in (-\infty, 0)$.

$c \in C$ denotes the set of all SFCs in the network slice. Each SFC c is composed of several VNF types arranged in sequence. $f_c(i)$ represents the ith VNF in c, $f_c(i) \in F$. We use a two-tuple $< \lambda_c, D_c^{\text{QoS}} >$ to denote the service requirement of SFC c, where λ_c represents the peak flow rate of c, and D_c^{QoS} represents the delay QoS requirement of c. To clearly describe the deployment of each SFC, we use p_c to represent the transmission path of c, $|p_c| = |c|$, and $p_c(i) \in S$ denotes the server where the ith VNF in c is deployed. Section 3 stipulates that one SFC must be deployed integrally in the cloud or at the edge. We use $\delta(c)$ to indicate the deployment status of SFC c, set $\delta(c)$ as 0 if c is deployed in the cloud, and 1 otherwise.

In this system model, we define the relationship between VNF instance, server, and SFC: (i) There can be instances of multiple VNF types in one server, but there can only be one VNF instance of the same type. (ii) The instances of one same VNF type can simultaneously exist in multiple servers. (iii) Each VNF type in one SFC only corresponds to one VNF instance such that the transmission path of each SFC is unique, regardless of traffic bifurcation. The key notations used in this paper are shown in Table 1.

Table 1 Key notations

Notation	Description
$G(S, L)$	Network topology
$S = S_c + S_e$	Set of servers, where S_c is the set of cloud servers, and S_e is the set of edge servers
L	Set of links. $l(u,v) \in L$ represents the link between server u, v $(u, v \in S)$
F	Set of VNF types
C	Set of SFCs
k_s	Computing capacity of server s, $s \in S$
k_s^r	Remaining computing resource of server s, $s \in S$
$b_l(u,v)$	Bandwidth of link $l(u,v)$
$\psi(f,s)$	Computing resource assigned to f-type VNF instance in server s
$\tau(u,v)$	Transmission delay of link $l(u,v)$
$f_c(i)$	The ith VNF type in SFC c, $f_c(i) \in F$
$p_c(i)$	Server in which $f_c(i)$ is deployed
α_f	Slope of f-type VNF throughput-resource function
β_f	Intercept of the horizontal axis of f-type throughput-resource function
γ_f	Exponent of f-type VNF delay-resource function
$< \lambda_c D_c^{\text{QoS}} >$	Service requirement of SFC c, where λ_c is the peak flowrate of c, and D_c^{QoS} is the delay QoS of c
$\delta(c)$	0 if SFC c is deployed in the cloud, and 1 if c is deployed at the edge

5. Constraints and objective

In this section, we describe the constraints and the objective (11) and start by discussing the constraints.

5.1 Constraints

Firstly, the total computing resource assigned to VNF instances inside server s must be less than the computing capacity k_s. Therefore, we have

$$\sum_{f \in F} \varphi(f,s) \leqslant k_s, \quad \forall s \in S. \tag{3}$$

Then, we need to ensure that the total traffic through link $l(u,v)$ cannot exceed the bandwidth of $l(u,v)$, which can be expressed as

$$\sum_{c \in C} \sum_{i=1}^{|c|-1} (\lambda_c | p_c(i) = u, p_c(i+1) = v) \leqslant b_1(u,v), \quad \forall l \in L. \tag{4}$$

Due to the QoS requirement of network service, we must avoid congestion when user traffic passes through VNF. Congestion will cause higher delay and increase the possibility of packet loss. Therefore, we hope that the processing rate of each VNF instance is higher than the arrival rate of data packets. Based on the P-R function defined in Section 3, the processing rate, i.e., throughput, of the f-type VNF instance in the server s is Throughput$(f, \psi(f,s))$. Then, the sum of flowrates of all

SFCs passing through this VNF instance cannot exceed the throughput of this instance, which can be indicated as

$$\sum_{c\in C}\sum_{i=1}^{|c|}(\lambda_c|f_c(i)=f,p_c(i)=s)\leqslant \text{Throughput}(f,\varphi(f,s)),\quad \forall f\in F;\ \forall s\in S;\ \varphi(f,s)>0. \tag{5}$$

Based on the placement regulation in Subsection 3.1, all VNF instances in one SFC must be integrally located in the cloud or at the edge. Thus, we can obtain

$$\begin{cases}\prod_{i=1}^{|c|}(p_c(i)\in S_e)=1,\ \delta(c)=1\\ \prod_{i=1}^{|c|}(p_c(i)\in S_c)=1,\ \delta(c)=0\end{cases} \tag{6}$$

where $\forall c\in C$, $(p_c(i)\in S_e)$, $(p_c(i)\in S_c)$ are Boolean expressions whose values are 1 for true and 0 for false.

Finally, we discuss the most vital constraint, the SFC QoS constraint. The total delay of SFC c consists of three parts: transmission delay, processing delay, and access delay. We use D_c^t to represent the overall transmission delay of c, which is the sum of link delays of all the links in the transmission path. For each link, the link delay consists of a queueing delay and a transmission delay. When (4) is satisfied, there is no queuing delay on the link between servers. Therefore, the link delay equals the link transmission delay. Due to the extremely close physical distance between nodes in cloud data center, the link transmission delay is extremely low. We set the average link transmission delay between servers as t. Thus, the average link transmission delay of c is denoted by D_c^t and can be calculated as

$$\begin{cases}D_c^t=\sum_{i=1}^{|c|-1}x_i\tau(p_c(i),p_c(i+1))\\ x_i=\begin{cases}0,\ p_c(i)=p_c(i+1)\\ 1,\ \text{else}\end{cases}\end{cases}. \tag{7}$$

The overall processing delay of c is denoted by D_c^p, which is composed of the processing delays of all the VNF instances in its transmission path, i.e.,

$$D_c^p=\sum_{i=1}^{|c|}\text{Delay}(f_c(i),\varphi(f_c(i),p_c(i))). \tag{8}$$

The third part is the access delay, denoted by D_c^a. If SFC c is deployed at the edge, the access delay is 0. If the service chain is deployed in the cloud, the access delay is the transmission delay between the user terminal and the cloud entry, represented by T. Thus, we express D_c^a as

$$D_c^a=\begin{cases}0,\ \delta(c)=1\\ T,\ \delta(c)=0\end{cases}=|\delta(c)-1|T. \tag{9}$$

where D_c is the sum of D_c^t, D_c^p and D_c^a, which cannot exceed the QoS delay, D_c^{QoS}. Therefore, we have the following QoS constraint:

$$D_c=D_c^t+D_c^p+D_c^a\leqslant D_c^{\text{QoS}},\quad \forall c\in C. \tag{10}$$

5.2 Objective formula

This model aims to maximize the utilization of edge computing resources. We use the total throughput of all SFCs deployed at the edge to measure the edge resource utilization efficiency. Thus, we can obtain the following objective formula:

$$\max\sum_{c\in C}\delta(c)\lambda_c \quad \text{s.t. } (1)-(10). \tag{11}$$

This objective formula can maximize the utilization efficiency of edge computing resources while satisfying all the constraints. We name this problem the max ET problem. In the next section, we provide the solution.

6. Algorithmic solution

This section presents the algorithmic solution of the max ET problem. First, we prove the complexity of the problem. Since many existing studies have proved that the VNF placement problem in the network is NP-hard, we introduce a new decision-making problem to our model: whether to place SFCs in the cloud or at the edge. Now, we prove that the max ET problem is still an NP-hard problem.

Theorem 1 Max ET problem is an NP-Hard problem.

Proof Reduce (11) as while the number of edge servers, $|S_e|$, and the length of each SFC, $|c|$, are limited to 1, and the constraints (4), (7)−(10) are removed, the problem will become a classic knapsack problem. The backpack capacity is the computing resource capacity of the only edge server; the item volume is the computing resource required by the only VNF in the SFC, and the item value is the flow rate of the SFC.

The max ET problem can be reduced to the knapsack problem, an NP-complete problem, which means that solving the max ET problem is NP-hard. □

The above proof shows that the max ET problem can be reduced to the knapsack problem. Thus, we regard the max ET problem as a much more complicated knapsack problem. The set of edge servers, S_e, is the backpack, and the SFC set, C, corresponds to the object set. The objective is to find a subset of C, that can be placed into S_e and has the largest total throughput. Compared with the

classic knapsack problem, we summarize three challenges of the proposed optimized problem.

Challenge 1 Satisfaction of QoS constraint: The QoS constraint has the highest priority. SFCs with rigorous QoS requirements cannot afford high access delays and must be deployed at the edge. Therefore, we must place SFCs with strict QoS into S_e first.

Challenge 2 Maintenance of remaining computing resources in S_e. When solving the knapsack Problem, the remaining volume of the knapsack must be continuously updated. The edge servers are composed of several independent servers, and their remaining computing resources should be maintained independently and respectively.

Challenge 3 Placement of SFC into S_e. As for the knapsack problem, after one object was put into the knapsack, the state of this object is simple. In the proposed model, putting an SFC into S_e is essentially the process of selecting edge servers for all the VNFs contained by the SFC. The state of this SFC in S_e is still being determined, and we need to decide further which server in S_e fits each VNF best.

Then, we discuss the algorithmic solution in three parts with the above three challenges as clues.

6.1 Prior placement based on QoS constraints

Due to (10), part of SFCs cannot be placed in the cloud because of their strict QoS requirements. This subsection estimates which SFCs must be placed at the edge.

Based on (7)−(9), we can calculate the minimum total delay when an SFC is placed in the cloud. When deployed in the cloud, the internal transmission delay is the same as the delay at the edge, and the access delay is T. Computing resource in the cloud is relatively sufficient. In theory, the maximum computing resource that can be assigned to one VNF instance is the total computing capacity of a single server, k_s. This paper does not consider multiple servers to co-process one VNF instance. Therefore, the theoretical minimum processing delay of an f-type VNF instance is

$$\min D_c^p = \sum_{i=1}^{|c|} \text{Delay}(f_c(i), k_s). \tag{12}$$

Then, we obtain the minimum total delay of SFC c, as

$$\min D_c = (|c|-1)t + \sum_{i=1}^{|c|} \text{Delay}(f_c(i, k_s) + T,$$
$$\delta(c) = 0. \tag{13}$$

If min D_c of SFC in the cloud is higher than the QoS requirement, this SFC has to be placed at the edge preferentially.

6.2 Recursive algorithm for max ET problem

The most used solution to the knapsack problem is dynamic programming. However, due to Challenge 3, the rest space of the knapsack, i.e., the remaining computing resource of edge servers, cannot be simply represented by an integer, and the number of possible states of edge servers is infinite. Therefore, it is impossible to encode the remaining space of the knapsack as index and use data structures such as array and list to store the current state of S_e and the corresponding ET value. Therefore, dynamic programming is not suitable for the max ET problem.

The network environment of this model is inside one network service slice, which only supports one single network service. As a result, the number of SFCs inside the slice is theoretically small, i.e., the size of the object set is small. Hence, adopting recursive methods to solve max ET problem will not cause an intolerable time and space cost. Therefore, we propose the recursive algorithm for max ET, abbreviated as the RMET algorithm.

The key to recursion is the derivation of the recursive formula. First, we sort all the SFCs in descending order according to λ_c. We use ET(n, S_e) to represent the recursive function where ET refers to total throughput of service chains at the Edge, and n is the recursive index. Here, S_e refers to the set of edge servers and includes the VNF instances placement state in S_e. Analogous to the knapsack problem, S_e is the current state of the knapsack, including the SFCs already placed at the edge and the remaining computing resources of all the edge servers. ET(n, S_e) represents the max total throughput by placing the first n SFCs into the current S_e. The objective of the max ET problem is to obtain ET$(|c|, S_e(\text{the initial} S_e))$ and the corresponding placement scheme.

We utilize a top-down approach to figure out this issue. If we have reached the last step, we need to determine whether to place the last SFC at the edge ($n = |C|$, S_e is empty). For the last SFC $c(n)$, there are two operations:

(i) Not place $c(n)$ at the edge. The total throughput is ET(n, S_e) = ET$(n-1, S_e)$.

(ii) Place $c(n)$ at the edge. The total throughput is ET(n, S_e) = ET$(n-1, S_e^*) + \lambda_{c(n)}$, S_e^* is the state of edge servers after placing $c(n)$ into S_e.

We choose the operation with higher throughput, and obtain the recursion formula

$$\text{ET}(n, S_e) = \max(\text{ET}(n-1, S_e), \lambda_{c(n)} + \text{ET}(n-1, S_e^*)). \tag{14}$$

According to the recursive formula, the recursive function ET is shown in Algorithm 1. When index n is 0, recursion has reached the bottom, and there is no SFC in

S_e. Thus, 0 is returned. Then, we calculate the total throughput without $c(n)$ in S_e, denoted by res_1. To figure out whether S_e has enough resources to contain $c(n)$, we define the SFC placement function, $\text{SFCP}(c,\lambda_c,S_e)$, which returns 1 if $c(n)$ can be placed into S_e, and 0 otherwise. When there is not enough resource for $c(n)$, return res_1. If the placement of $c(n)$ succeeds, we use res_2 to represent the total throughput with $c(n)$ in S_e. We take the larger one between res_1 and res_2 as the return value. In Step 8 and Step 11, $\text{record}(n,i,\text{res})$ is the result recording function, recording the result of every recursive round. When the recursion is completed, the record function can record the entire recursion tree and we can get the optimal placement scheme.

Algorithm 1 Recursive function: $\text{ET}(C,n,S_e)$

Input: C, n, S_e
Output: res
1 if n<0 then
2 return 0;
3 end
4 $S_e^*=S_e$;
5 $\text{res}_1 = \text{ET}(C,n-1,S_e)$;
6 if $\text{SFCP}(c(n),\lambda_{c(n)},S_e^*)=1$ then
7 $\text{res}_2=\lambda_{c(n)}+\text{ET}(C,n-1,S_e^*)$;
8 else
9 $\text{record}(n,0,\text{res}_1)$;
10 return res_1;
11 end
12 $\text{res} = \text{res}_1 > \text{res}_2 ? \text{res}_1 : \text{res}_2$;
13 $\text{record}(n,0,\text{res}_1)$;
14 $\text{record}(n,1,\text{res}_2)$;
15 return res

6.3 SFC placement and resource allocation at the edge

Placing an SFC into edge servers is selecting the most suitable edge server for every VNF type within this SFC. We propose a heuristic SFC placement algorithm to improve the resource utilization efficiency in edge servers while satisfying the throughput and delay QoS requirements of SFCs.

According to the service requirements $<\lambda_c, D_c^{\text{QoS}}>$, we must ensure that the throughput and delay of c satisfy the constraints (5) and (10) simultaneously. The flow rate is a serial flux parameter, and every VNF instance on the deployment path p_c must guarantee that λ_c can pass smoothly. With λ_c and P-R function, the required computing resource can be calculated directly. However, the overall delay of SFC deployed in S_e is the sum of the delays of all links and VNF instances on p_c. It cannot be directly utilized to calculate computing resources.

Therefore, we perform the placement of SFC c in two steps.

Step 1 Calculate the required computing resources according to λ_c and select the appropriate server for each VNF instance.

Step 2 Verify the scheme in Step 1. If the delay QoS is satisfied, the plan is completed. Otherwise, revise the plan until the delay QoS is satisfied.

Now, we move to Step 1. For each f-type VNF in SFC, there are two states in S_e: (i) f-type VNF instance has already existed; (ii) f-type VNF instance does not exist.

In the case of State 1, we choose to expand the existing instance. Since existing instances already contain startup computing resources, expansion can save startup resources compared to redeployment. Meanwhile, expansion can improve the performance and reduce the processing delay of VNF instances. According to (1), we can obtain the expansion resource required by $f_c(i)$-type VNF instance for processing λ_c, which is denoted by ψ_1 as follows:

$$\varphi_1 = \lambda_c/\alpha_{f_c(i)} + \beta_{f_c(i)}. \tag{15}$$

We use S^* to represent the set of servers that contain $f_c(i)$-type VNF instance and have remaining resources greater than ψ_1. Then, we expand the $f_c(i)$-type VNF instance inside the server, which has the least remaining resource in S^*. Now we explain why we select the server with the least resource in S_e. We arrange the SFC set in descending order of λ_c before performing the recursion. The recursion is carried out from top to bottom, which means the flow rate of each subsequent SFC is higher than the current one. Compared with VNF instances of the same type in the subsequent SFCs, the instance in the current SFC requires the least computing resource. Therefore, selecting the server with the least remaining resources in S_e can improve the success rate of finding expansion targets for VNF instances of the same type in the subsequent SFCs.

If the remaining resources of servers containing f-type VNF instances are all less than ψ_1, then the situation is the same as State 2, where we must create a new f-type VNF instance in S_e. We use ψ_2 to denote the resource required by a new $f_c(i)$-type VNF instance to process throughput λ_c as follows:

$$\varphi_2 = \lambda_c/\alpha_{f_c(i)} + \beta_{f_c(i)}. \tag{16}$$

Then, we select the server with the most remaining resource to create the new instances, which can increase the expansion success rate of VNF instances in subsequent SFCs. If there is no server with remaining

resources greater than ψ_2 in S_e, there is not enough computing resource for SFC c, and 0 is returned. When all the VNF instances in SFC c complete the server selection, Step 1 is accomplished.

Next, we move to Step 2, where we test whether the preliminary scheme meets the delay QoS. If delay QoS is satisfied, the VNF placement and resource allocation for SFC c is completed. Otherwise, we need to adjust the allocated computing resources of VNF instances to reduce the total delay. First, we calculate the delay excess D^* as follows:

$$D^* = D_c - D_c^{\text{QoS}} = (|c|-1)t + \sum_{i=1}^{|c|} \text{Delay}(f_c(i), \varphi(f_c(i), p_c(i))) - D_c^{\text{QoS}}. \tag{17}$$

For the f-type VNF instance, the computing resource required to shorten the delay by D^* is denoted by ψ^*. The calculation is as follows:

$$\begin{cases} \varphi^{\gamma_f} - (\varphi+\varphi^*)^{\gamma_f} = D^* \\ (\varphi+\varphi^*)^{\gamma_f} = \varphi^{\gamma_f} - D^* \\ \varphi+\varphi^* = (\varphi^{\gamma_f} - D^*)^{-\gamma_f} \\ \varphi^* = (\varphi^{\gamma_f} - D^*)^{-\gamma_f} - \varphi \end{cases}. \tag{18}$$

We use $\psi^*(i)$ to represent the computing resources required by the $f_c(i)$ instance to shorten the processing delay by D^* in the Step 1 scheme. Then, we select the instance with the smallest $\psi^*(i)$ for expansion. When the remaining resources of $p_c(i)$, i.e., $k^r_{p_c(i)}$, are less than $\psi^*(i)$, then expand the $f_c(i)$ instance by $k^r_{p_c(i)}$ and update D^* as

$$D^* = D^* - \left(\varphi^{\gamma_f} - (\varphi + k^r_{p_c(i)})^{\gamma_f}\right). \tag{19}$$

Then, we remove this instance from c and repeat the above process until D^* equals 0. For the worst situation, D^* is still greater than 0 when all the servers in S_e are full-loaded, which means the delay QoS of SFC c cannot be satisfied, and 0 is returned.

When Step 2 is completed, we are faced with a problem, there could be multiple instances with throughput redundancy in S_e. We use $\delta(f,s)$ to denote the throughput redundant resource of the f-type instance in server s. To improve resource utilization efficiency, we modify the selection of expansion targets in Step 1. We first look for expansion targets in instances that include redundant throughput. The complete algorithm description is shown in Algorithm 2.

Algorithm 2 SFCP algorithm

Input: C, λ_c, D_c^{QoS}, S_e
Output: 1 if placement completes; 0 otherwise

for $f_c(i) \in c$ do
  $\psi_1 = \dfrac{\lambda_c}{\alpha_{f_{C(i)}}} + \beta_{f_{C(i)}}$;
  $\psi_2 = \dfrac{\lambda_c}{\alpha_{f_{C(i)}}} + \beta_{f_{C(i)}}$;
if $\exists f_c(i)$-Type instance $\in s, s \in S_e$ then
  $S^* = \{s \mid s \ni f_c(i) - \text{Typeinstance}, s \in S_e\}$;
  $S^{**} = \{s \mid \delta(f_c(i), s) > 0, s \in S^*\}$;
  $s_0 = \arg\max_s (\delta(f_c(i), s)), (s \in S^{**})$;
  if $(\delta(f_c(i), s_0) + k^r_{s_0}) \geqslant \varphi_1$ then
  $p_c(i) = s_0$;
  if $\delta(f_c(i), s_0) \geqslant \varphi_1$ then
    $\delta(f_c(i), s_0) -= \varphi_1$;
    else
      $\delta(f_c(i), s_0) = 0$;
      $k^r_{s_0} -= (\varphi_1 - \delta(f_c(i), s_0))$;
      $\varphi(f_c(i), s_0) += (\varphi_1 - \delta(f_c(i), s_0))$;
    end
  end
  else
    $s_0 = \arg\max_s (k^r_s), (s \in S^*)$;
    if $k^r_{s_0} \geqslant (\varphi_1)$ then
      $p_c(i) = s_0; k^r_{s_0} = \varphi_1$;
      $\varphi(f_c(i), s_0) += \varphi_1$;
    else
      if $\exists s \in S_e, k_s \geqslant \varphi_2$ then
        $p_c(i) = s_0; k^r_{s_0} = \varphi_2$;
        $\varphi(f_c(i), s_0) += \varphi_2$;
      else
        return 0;
      end
    end
  end
end
if $D_c \leqslant D_c^{\text{QoS}}$ then
  return 1;
else
  $D^* = D_c - D_c^{\text{QoS}}$;
end
while $c \neq \varnothing$ do
  for $f_c(i) \in c$ do
    $\varphi^*(i) = ((\varphi)^{\gamma_{f_{C(i)}}} - D^*)^{-\gamma_{f_{C(i)}}} - \varphi$;
  end
  $k = \arg\min_i \varphi^*(i); s_0 = p_c(k)$;
  if $k^r_{s_0} \geqslant \varphi^*$ then

$k^r_{s_0} = \varphi^*$;
$\varphi(f_c(k), s_0) += \varphi^*$; $\delta(f_c(k), s_0) += \varphi^*$;
return 1;
else
$\varphi(f_c(k), s_0) += k^r_{s_0}$; $\delta(f_c(k), s_0) += k^r_{s_0}$;
$k^r_{s_0} = 0$;
$D^* = D^* - (\varphi^{\gamma_{f_c(k)}} - (\varphi + k^r_{pc(k)})^{\gamma_{f_c(k)}})$;
remove $f_c(k)$ from c;
end
end
return 0

7. Simulation and numerical results

In this section, we test and demonstrate the performance of the RMET algorithm through simulations in specific scenarios. We first present three comparison algorithms. Then, we build two network scenarios including a randomly generated scenario and an autonomous driving network scenario. Based on two scenarios, we test four algorithms and analyze the numerical results.

7.1 Comparing algorithm

We design three comparing algorithms:

(i) Greedy algorithm: For each VNF instance in each SFC, select the optimal server in the current state, that is, the server with the largest remaining space, to create a new instance.

(ii) Aggregation algorithm: For each VNF instance in each SFC, select an existing instance of the same type for expansion. If there is no deployed instance of the same type, select the server with the most sufficient remaining resources to create a new instance.

(iii) Deep reinforcement learning algorithm: According to [31], we implement the SFC placement algorithm based on deep reinforcement learning (DRL) to maximize the utilization efficiency of edge computing resources.

7.2 Scenario 1: random generated scenario

The network parameters in Scenario 1 are randomly generated, and the purpose is to test the performance of the RMET algorithm from a mathematical point of view. In Scenario 1, the numbers of edge servers, VNF Types, and SFCs are 6, 10, and 10, respectively. The flow rates of SFCs and P-R function parameters of each VNF type are randomly generated within a specific range. The sequence of the VNF types in each SFC is also randomly generated. The specific network service parameters are demonstrated in Fig. 4, Table 2, and Table 3.

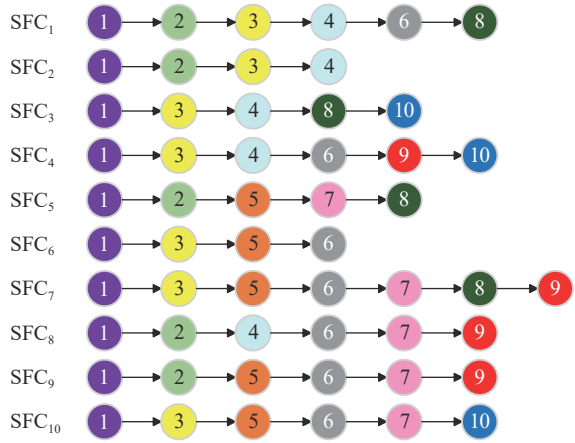


Fig. 4 Network service parameters of Scenario 1

Table 2 Key parameters in Scenario 1

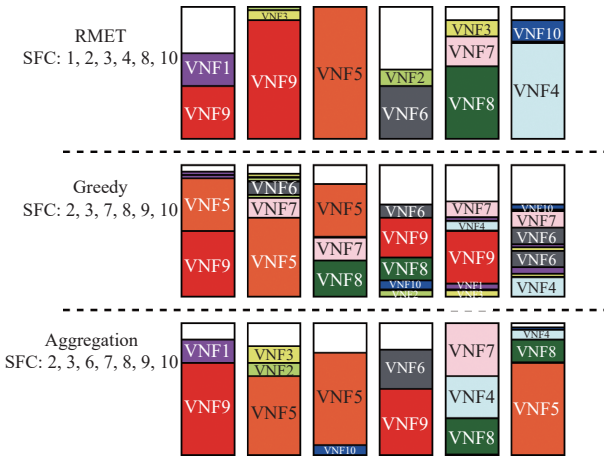
f-Type	α_f	β_f	γ_f
1	10	0	−2
2	8	0	−4
3	12	0	−8
4	3	1	−3
5	0.5	10	−1
6	5	8	−6
7	2	2	−9
8	2	5	−7
9	0.5	0.5	−3
10	6	0	−5

Table 3 SFC parameters in Scenario 1

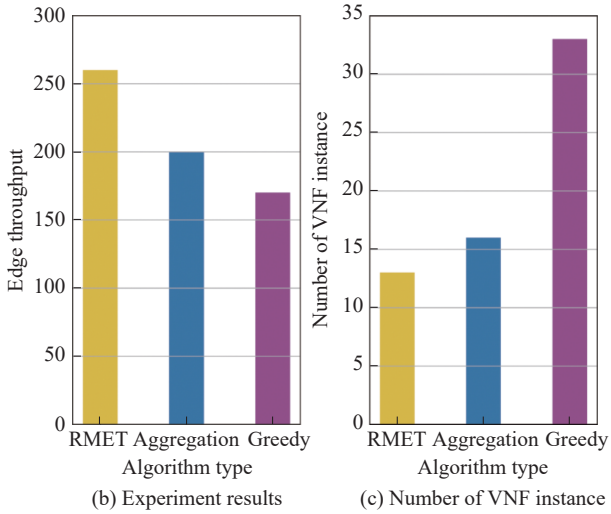
SFC	Flowrate/(request/s)	D^{QoS}/ms
SFC1	55	20
SFC2	50	15
SFC3	45	10
SFC4	45	30
SFC5	35	20
SFC6	30	10
SFC7	25	15
SFC8	20	30
SFC9	15	25
SFC10	15	15

In Scenario 1, the placement schemes of RMET, Greedy, and Aggregation are shown in Fig. 5. Then

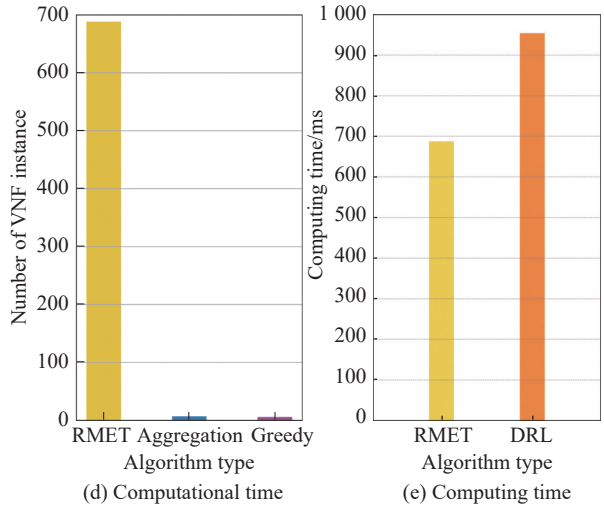
we measure the total throughput at the edge (ET), the number of VNF instances in edge servers, and the remaining computing resources, as shown in Fig. 5. RMET algorithm selects SFC 1, 2, 3, 4, 6, 8, 10 to deploy at the edge and has the largest ET, 260. Greedy Algorithm chooses SFC 2, 3, 7, 8, 9, 10 with ET being 170. Aggregation selects SFC 2, 3, 6, 7, 8, 9, 10 with ET being 200. The results indicate that compared with Greedy and Aggregation, the RMET algorithm maximizes the total throughput at the edge using the same computing resource. Moreover, we compare the computing time of RMET algorithm with the latest reinforcement learning algorithm, as shown in Fig. 5. The results indicate that the SFC placement algorithm based on DRL can approach the optimal solution but takes a long time to converge. Meanwhile, the RMET algorithm can obtain the optimal solution within less computing time.



(a) VNF placement and resource allocation scheme of three algorithms



(b) Experiment results (c) Number of VNF instance



(d) Computational time (e) Computing time

Fig. 5 Simulation results of Scenario 1

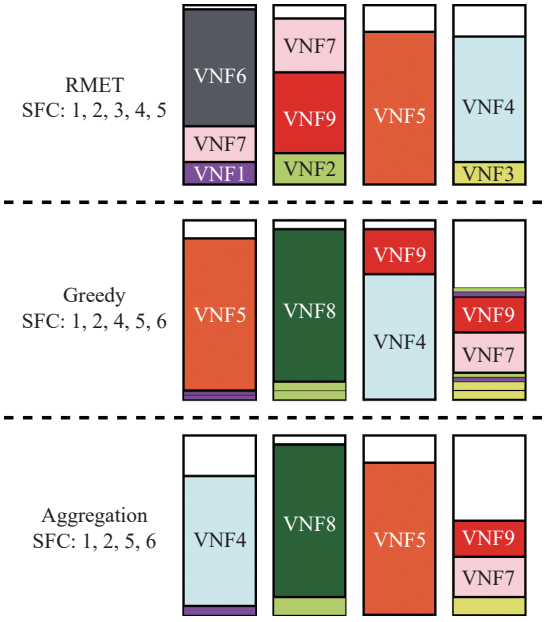
Then, we focus on the number of VNF instances in the edge servers. NFV MANO system at the edge needs to monitor and control all the VNF instances. The more VNF instances, the more computing and bandwidth resources are cost. We can find that the total number of VNF instances in the edge servers of RMET, Aggregation, and Greedy schemes are 13, 16, and 33, respectively. This result verifies that RMET and Aggregation algorithm reduce the MANO cost by scaling deployed instances instead of creating new ones.

We also record the computing times of the three algorithms. All algorithm simulations in this paper are conducted using Intel(r) Core(TM) i5-7500 CPU. In Scenario 1, the computing times of RMET, Aggregation, and Greedy algorithms are 688 ms, 7 ms, and 6 ms, respectively. Although the computing time of RMET is the largest, it still does not exceed 1 s. VNF placement and computing resource allocation are pre-decisions based on existing information and do not require high real-time performance. Therefore, the calculation time of RMET is completely acceptable.

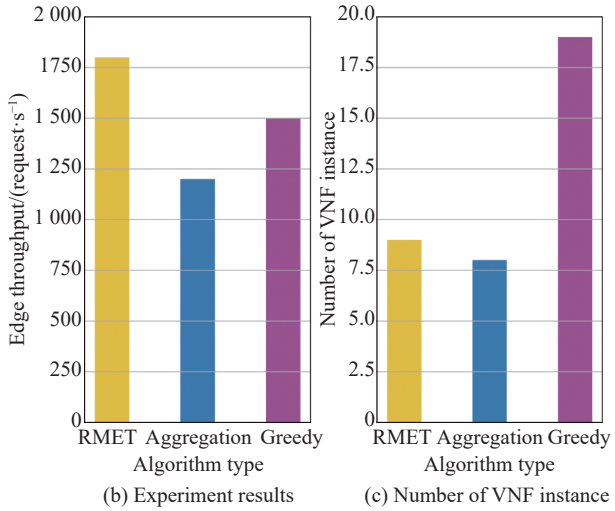
7.3 Scenario 2: automatic driving scenario

We now move to Scenario 2, where we simulate the automatic driving network slice. As shown in Fig. 6, this network slice containing edge and cloud computing resources provides auto-driving support for vehicles at an intersection. The maximum number of vehicles in this intersection is 50. There are four high-performance servers in the edge node, and each server has 100 units of

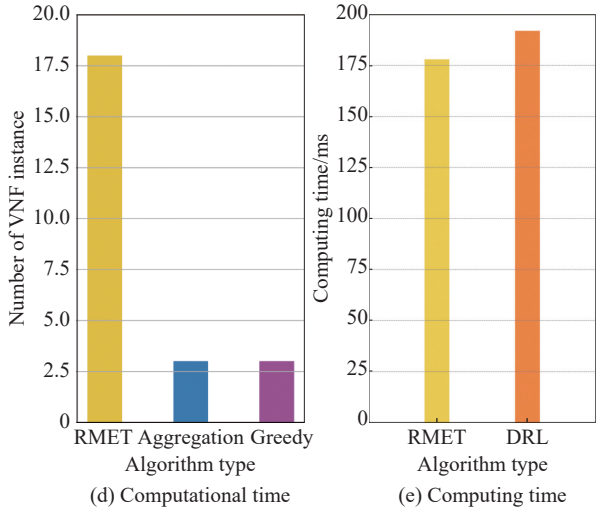
computing resources. We set the level of driving automation as 4 or above, which means fully auto-driving and has extremely strict QoS requirements [28]. The access delay between each car and the cloud is 5 ms. There are nine types of VNF instances in the slice, including NAT, traffic monitoring (TM), FW, and six automated driving network functions (ADNF) 1 to 6. This network slice contains six SFCs, which respectively have their own QoS requirements. Each car will continue to send service requests in the coverage area of the network slice. The specific network service parameters are demonstrated in Fig. 7, Table 4 and Table 5.



(a) VNF placement and resource allocation scheme of three algorithms



(b) Experiment results (c) Number of VNF instance



(d) Computational time (e) Computing time

Fig. 6 Simulation results of Scenario 2

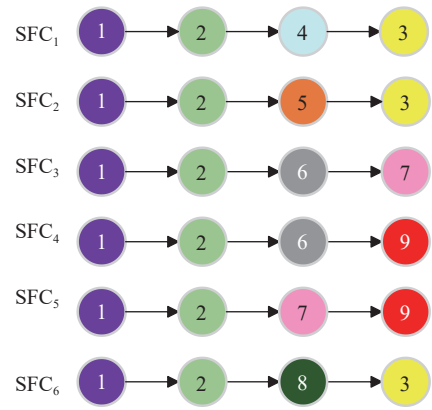


Fig. 7 Network service parameters of Scenario 2

Table 4 Key parameters in Scenario 2

VNF	f-Type	α_f	β_f	γ_f
FW	1	150	0	−10
TM	2	100	0	−8
NAT	3	80	0	−8
ADNF1	4	7	0	−5
ADNF2	5	5	5	−3
ADNF3	6	10	0	−3
ADNF4	7	15	5	−5
ADNF5	8	0.6	0	−2
ADNF6	9	12	0	−5

Table 5 SFC parameters in Scenario 2

SFC	Flowrate/(request/s)	D^{QoS}/ms
SFC1	10	1
SFC2	8	2
SFC3	7	12
SFC4	6	12
SFC5	5	15
SFC6	1	20

The QoS requirements of SFC 1, 2 are all 1 ms, which is less than the cloud access delay. To satisfy the QoS requirements, SFC 1, 2 must be deployed at the edge. Fig. 8 shows the placement schemes and results analysis of RMET, Greedy, and Aggregation algorithm in Scenario 2. When using Greedy and Aggregation algorithms, SFC 1, 2 are also preferentially deployed at the edge. Otherwise, the QoS requirements cannot be guaranteed, and the placement schemes are meaningless. Results indicate that the RMET algorithm has the largest ET of 1 800 requests/s, while the total throughput at the edge of aggregation and greedy are 1 200 requests/s and 1 500 requests/s. Regarding the number of VNF instances, RMET and aggregation are less than greedy and have lower MANO costs. The computing time of RMET for Scenario 2 is 178 ms, which is much lower than the computing time in Scenario 1. Compared with RMET, the DRL algorithm still takes a longer time in Scenario 2. Thus, RMET can maximize the ET and the time consumption is acceptable.

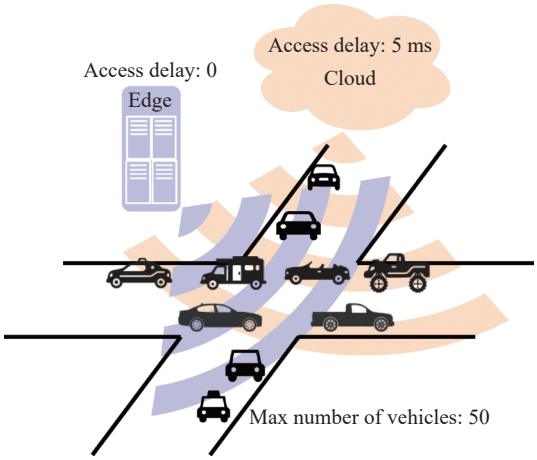


Fig. 8 Simulation results of Scenario 2

Finally, we want to discuss the particularity of VNF8 in SFC 6. The resource consumption rate of VNF8 is extremely high. Although the request frequency of SFC 6 is only one request/s for each vehicle, VNF8 still requires numerous computing resources. When using the Greedy and Aggregation algorithms, SFC 6 is deployed at the edge, and VNF8 almost occupies the computing resource of one whole server. Differing from greedy and aggregation, the RMET algorithm will calculate the largest edge throughputs with and without SFC 6 deployed at the edge and choose the one with a larger ET. Therefore, RMET can avoid the situation that part of SFCs occupies too much computing resource at the edge to maximize the overall utilization efficiency of edge computing resources.

In summary, in both random and autonomous driving scenarios, the placement scheme of RMET algorithm has the largest edge throughput. Simultaneously, the RMET can reduce the number of VNF instances in edge servers and reduce the overhead of maintaining and managing VNF instances. The computational time of RMET does not exceed 1 s, which is completely acceptable for proactive network service placement. Therefore, the RMET can effectively improve the utilization efficiency of edge computing resources.

8. Conclusions

In this paper, we focus on optimizing the placement of network services in cloud-edge environments to maximize the efficiency of edge computing resources. We formulate one new regulation for SFC placement in cloud-edge environments: VNF instances contained in the same SFC should integrally be deployed in the cloud or at the edge. Moreover, based on confirmatory experiments, we propose the concept of P-R function, which describes the relationship between the VNF instance performance and the allocated resource. With P-R function, we can calculate the required computing resource for every type of VNF instance in one network service according to the service requirement. We then present a VNF placement and resource allocation model in cloud-edge environments and configure each VNF type in the model with its particular P-R function. By taking the edge throughput as KPI, we obtain the max ET problem. We propose the RMET algorithm as the solution, which contains two parts: (i) a recursive strategy for selecting SFCs to be deployed at the edge; (ii) a heuristic solution for placing SFC into edge servers and allocating computing resources precisely based on the P-R function and service requirement. To verify the performance of the RMET algorithm, we conduct simulations in random scenario and auto-driving scenario. The simulation results demonstrate that RMET can improve the utilization efficiency of edge computing resources. Furthermore, our future work will aim to adopt the P-R function to other network scenarios, such as the IoT network and the large-scale DC network.

References

[1] COLAKOVIC A, HADZIALIC M. Internet of things (IOT): a review of enabling technologies, challenges, and open research issues. Computer Networks, 2018, 144: 17–39.
[2] PANTELOPOULOS A, BOURBAKIS N G. A survey on wearable sensor-based systems for health monitoring and prognosis. IEEE Trans. on Systems, Man, and Cybernetics, Part C, 2009, 40(1): 1–12.
[3] YURTSEVER E, LAMBERT J, CARBALLO A, et al. A survey of autonomous driving: common practices and emerging technologies. IEEE Access, 2020, 8: 58443–58469.
[4] ALTURKI R, GAY V. Augmented and virtual reality in mobile fitness applications: a survey. Cham: Springer, 2019.
[5] FENG C, HAN P C, ZHANG X, et al. Computation offload-

ing in mobile edge computing networks: a survey. Journal of Network and Computer Applications, 2022, 202: 103366.
[6] CAO K, HU S Y, SHI Y, et al. A survey on edge and edge-cloud computing assisted cyber-physical systems. IEEE Trans. on Industrial Informatics, 2021, 17(11): 7806–7819.
[7] ETSI GS NFV 001. Network functions virtualisation (NFV); use cases. Nice: European Telecommunications Standards Institute, 2013.
[8] RFC 7665. Service function chaining (SFC) architecture. Lake Wylie: RFC Editor, 2015.
[9] MCKEOWN N, ANDERSON T, BALAKRISHNAN H, et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69–74.
[10] MCKEOWN N. Software-defined networking. INFOCOM Keynote Talk, 2009, 17(2): 30–32.
[11] ZHANG S L. An overview of network slicing for 5G. IEEE Wireless Communications, 2019, 26(3): 111–117.
[12] DESOUSA N F S, PEREZ D A L, ROSA R V, et al. Network service orchestration: a survey. Computer Communications, 2019, 142/143: 69–94.
[13] SHI W S, CAO J, ZHANG Q, et al. Edge computing: vision and challenges. IEEE Internet of Things Journal, 2016, 3(5): 637–646.
[14] REN J, ZHANG D Y, HE S W, et al. A survey on end-edge-cloud orchestrated network computing paradigms: transparent computing, mobile edge computing, fog computing, and cloudlet. ACM Computing Surveys, 2019, 52(6): 1–36.
[15] WANG K, ZHOU Q H, GUO S, et al. Cluster frameworks for efficient scheduling and resource allocation in data center networks: a survey. IEEE Communications Surveys & Tutorials, 2018, 20(4): 3560–3580.
[16] DU M Z, WANG Y, YE K J, et al. Algorithmics of cost-driven computation offloading in the edge-cloud environment. IEEE Trans. on Computers, 2020, 69(10): 1519–1532.
[17] HUANG M F, LIU W, WANG T, et al. A cloud–MEC collaborative task offloading scheme with service orchestration. IEEE Internet of Things Journal, 2019, 7(7): 5792–5805.
[18] LONG X, WU J G, CHEN L. Energy-efficient offloading in mobile edge computing with edge-cloud collaboration. Proc. of the International Conference on Algorithms and Architectures for Parallel Processing, 2018: 460−475.
[19] SLIM F, GUILLEMIN F, HADJADJ-AOUL Y. CLOSE: a costless service offloading strategy for distributed edge cloud. Proc. of the 15th IEEE Annual Consumer Communications & Networking Conference, 2018. DOI: 10.1109/CCNC.2018.8319276.
[20] PENG G, WU H M, WU H, et al. Constrained multiobjective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing. IEEE Internet of Things Journal, 2021, 8(17): 13723–13736.
[21] WU J Z, CAO Z Y, ZHANG Y J, et al. Edge-cloud collaborative computation offloading model based on improved partical swarm optimization in MEC. Proc. of the IEEE 25th International Conference on Parallel and Distributed Systems, 2019: 959−962.
[22] ELIE E H, NGUYEN T M, ASSI C. Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds. IEEE Trans. on Communications, 2019, 67(5): 3407–3421.
[23] ZHANG Q X, LIU F M, ZENG C B. Adaptive interference-aware VNF placement for service-customized 5G network slices. Proc. of the IEEE Conference on Computer Communications, 2019: 2449−2457.
[24] RANDRIAMASINORO N M, NGUYEN K K, CHERIET M. Optimized resource allocation in edge-cloud environment. Proc. of the Annual IEEE International Systems Conference, 2018. DOI: 10.1109/SYSCON.2018.8369606.
[25] SEFRAOUI O, AISSAOUI M, ELEULDJ M. OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications, 2012, 55(3): 38–42.
[26] AGARWAL S, MALANDRINO F, CHIASSERINI C F, et al. VNF placement and resource allocation for the support of vertical services in 5G networks. IEEE/ACM Transactions on networking, 2019, 27(1): 433–446.
[27] ALAMEDDINE H A, SHARAFEDDINE S, SEBBAH S, et al. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. IEEE Journal on Selected Areas in Communications, 2019, 37(3): 668–682.
[28] LI M, ZHANG Q X, LIU F M. Finedge: a dynamic cost-efficient edge resource management platform for NFV network. Proc. of the IEEE/ACM 28th International Symposium on Quality of Service, 2020. DOI: 10.1109/IWQoS49365.2020.9212908.
[29] VAN ROSSEM S, TAVERNIER W, COLLE D, et al. Profile-based resource allocation for virtualized network functions. IEEE Trans. on Network and Service Management, 2019, 16(4): 1374–1388.
[30] HU X, CAO W, ZHU H Q. Data plane development kit (DPDK). Florida: CRC Press, 2020.
[31] FAN W T, YANG F, WANG P L, et al. DRL-based service function chain edge-to-edge and edge-to-cloud joint offloading in edge-cloud network. IEEE Trans. on Network and Service Management, 2023, 20(4): 4478–4493.

Biographies

HAN Yingchao was born in 1987. He received his M.S. degree from Harbin Institute of Technology, Harbin, China, in 2012. He is pursuing his Ph.D. degree at the School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin, China. His research interests include design of the aircraft and networking communication.
E-mail: 21B90567@stu.hit.edu.cn

MENG Weixiao was born in 1968. He received his B.S. degree in electronic instruments and measurement technology and Ph.D. degree in information and communication engineering from Harbin Institute of Technology, Harbin, China, in 1990 and 2000 respectively. He is now a professor in the School of Electronics and Information Engineering, Harbin Institute of Technology. His research interests include wireless communication and networking, sky and ground information transmission and networking, and integrated communication perception.
E-mail: wxmeng@hit.edu.cn

FAN Wentao was born in 1995. He received his B.S. and Ph.D. degrees of information and communication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2017 and 2023 respectively. His research interests include cloud/edge computing, network orchestration, data center network, and remote direct memory access (RDMA) system.
E-mail: fanwentao@cmss.chinamobile.com