Cost-Efficient Dynamic Service Function Chain Embedding in Edge Clouds

Weihan Chen*, Zhiliang Wang*, Han Zhang*, Xia Yin*, Xingang Shi*
*Tsinghua University, China
chenwh18@mails.tsinghua.edu.cn, wzl@cernet.edu.cn

***Abstract*—Edge Computing (EC) provides delay protection for some delay-sensitive network services by deploying cloud infrastructure with limited resources at the edge of the network. In addition, Network Function Virtualization (NFV) implements network functions by replacing traditional dedicated hardware devices with Virtual Network Function (VNF) that can run on general servers. In NFV environment, Service Function Chaining (SFC) is regarded as a promising way to reduce the cost of configuring network services. NFV therefore allows to deploy network functions in a more flexible and cost-efficient manner, and schedule network resources according to the dynamical variation of network traffic in EC. For service providers, seeking an optimal SFC embedding scheme can improve service performance and reduce embedding cost. In this paper, we study the problem of how to dynamically embed SFC in geo-distributed edge clouds network to serve user requests with different delay requirements, and formulate this problem as a Mixed Integer Linear Programming (MILP) which aims to minimize the total embedding cost. Furthermore, a novel SFC Cost-Efficient emBedding (SFC-CEB) algorithm has been proposed to efficiently embed required SFC and optimize the embedding cost. Based on the results of trace-driven simulations, the proposed algorithm can reduce SFC embedding cost by up to 37% compared with state-of-the-art schemes (e.g., RDIP).**

***Index Terms*—Network Function Virtualization, Edge Computing, Service Function Chaining, traffic routing**

I. Introduction

Benefiting from the development of mobile communication (e.g., 5G) and Internet of Things (IoT), the variety of mobile network services is growing rapidly, e.g., augmented reality, driving assistant service, personal health assistant. Such services are usually computing intensive and have different delay sensitivity. For some delay-sensitive services, if they are deployed on public cloud for processing, the propagation delay on backbone network can hardly satisfy their delay requirement. Edge computing (EC) has been proposed to handle the imperative requirements of vast computing resources and low delay at network edge [1], [2]. As a supplement of cloud computing, EC can reduce end-to-end delay effectively by providing resource-limited cloud infrastructures at network edges.

Similar to general datacenters, efficient cost management is also necessary for EC to save operational expenses while serving user requests [3]. Network Function Virtualization (NFV) [4] is proposed to satisfy this requirement. In NFV, network functions are realized in software instances running on general servers, which are called Virtual Network Function (VNF). Compared with dedicated devices, VNFs are easier to manage since the NFV allows to schedule network resources owned by VNF in an elastic and scalable way. Moreover, Service Function Chain (SFC) proposes a flexible and cost-efficient manner for provisioning services [5]. SFC allows a chain-ordered VNFs to perform a network service. For further saving operational expense, several SFCs can share the instances of the common VNFs [6].

Existing efforts for SFC deployment optimization usually jointly optimize VNF operation cost and traffic routing cost. However, these efforts which simply minimize the end-to-end delay for each request without considering the delay sensitivity of the request may cause Unreasonable Resource Preemption (URP) problem. In practice, there are both delay-sensitive services (tolerate hundreds of milliseconds delay, e.g., Internet of Vehicles applications) and delay-tolerant services (tolerate maximum delay from minutes to hours, e.g., personal health analytics applications) [7]. The URP problem is: when the volume of requests exceeds the capacity of the edge clouds, if the VNFs required by delay-tolerant services exhaust the resources of edge clouds with lower transmission delay, other delay-sensitive services have to be deployed on the edge clouds with higher transmission delay or even remote public cloud. This can result in a significant performance decrease for delay-sensitive services. Hence, a crucial problem is raised: *How to embed SFC with minimal cost, and avoid the quality of delay-sensitive services declining due to the URP problem.*

Solving the above problem is challenging. First, it requires making cost-efficient SFC embedding decisions efficiently in an online manner. Second, the offline optimization problem has proved to be NP-hard. To address these challenges, we first introduce Service Level Agreement (SLA) violation cost to replace end-to-end delay. In this way, even if the delay-tolerant services are deployed to the edge clouds or public cloud with higher transmission delay, there is no significant cost burden. And delay-sensitive services still need to be deployed on the edge clouds with lower transmission delays to ensure lower SLA violation cost. This helps to allocate resources in a more reasonable way based on the delay requirements of requests. Then we formulate the cost minimization for SFC embedding in geo-distributed cloud network as Mixed Integer Linear Programming (MILP).

Considering the complexity of solving this MILP problem, we propose a novel SFC Cost-Efficient emBedding (SFC-CEB) algorithm to achieve efficient SFC embedding in online status. This algorithm transforms the process of finding the

978-3-903176-36-2 ©2021 IFIP

SFC embedding scheme into the shortest path search in topology. By constructing the multi-layer graph based on required SFC and use VNF operation cost, bandwidth cost and SLA violation cost as the weight of path, SFC-CEB can leverage modified shortest path algorithm to find the SFC embedding scheme with minimal cost. The main contributions of this paper are summarized as follows:

- We find that existing works which only focus on end-to-end delay in joint cost optimization may incur more SLA violation cost due to inappropriate resource allocation (since they ignore the delay sensitivity of requests). Hence, we propose to replace end-to-end delay with SLA violation cost and optimize it along with VNF operation cost and bandwidth cost.
- We formulate cross geo-distributed clouds SFC embedding problem as a MILP model. In order to solve the problem efficiently, we propose a novel SFC-CEB algorithm which applies the modified shortest path algorithm in multi-layer graph. The algorithm will use related cost as path weight and find the minimal cost embedding scheme for requests based on path searching. We also analyze the complexity of SFC-CEB, which is about $O(N^2 logN)$.
- We conduct extensive experiments with real-trace driven simulations and compare SFC-CEB with state-of-the-art works (e.g., RDIP [6]). The evaluation results demonstrate that SFC-CEB can achieve lower total embedding cost (the cost can be reduced by up to 37%). We further analyze the performance of SFC-CEB in other aspects.

II. Related Work

For most of the existing studies which aim to solve the SFC embedding problem in geo-distributed cloud network, cost optimization and load balancing are the primary objectives. For cost optimization, some existing works only consider optimizing VNF operation cost. Yang *et al*. [8] proposed a heuristic algorithm to allocate resources incrementally and a set cover partition approximation algorithm to reoptimize global resource allocation periodically. There are also some studies that only optimize traffic routing costs. Gouareb *et al*. [9] developed a heuristic algorithm to minimize the queuing delay within edge clouds and network links. Chemodanov *et al*. [10] proposed a metapath-based composite variable approach to satisfy the QoS needs of latency-sensitive SFCs. However, optimizing VNF operation cost or traffic routing cost independently cannot guarantee global cost-efficient, which means the above schemes cannot realize better optimization effect than joint optimization schemes.

For joint optimization schemes, parts of works proposed heuristic algorithms to solve the problem. Bari *et al*. [11] proposed dynamic programming-based algorithm to find near-optimal SFC placement scheme, and Jin *et al*. [12] proposed two-phase algorithm to obtain pre-routing paths first and then select the path with minimal cost as deployment scheme for each request. Another part of works also tried to offer reliable performance bound. Jia *et al*. [13] and Zhou *et al*. [6] both proposed an online algorithm based on regularization approach and rounding scheme to derive feasible solutions with provable performance guarantee. In addition to these algorithms designed based on linear programming models, some studies [14], [15] also use deep learning or reinforcement learning technology to embed SFC. One advantage of this kind of solution is model-free, it can find the proper SFC embedding schemes without establishing a complex mathematical model, but the reliability of the result cannot be guaranteed. However, all these works ignore the delay sensitivity of different requests. This may cause more SLA violation penalty due to the URP problem. In contrast, our algorithm can achieve better optimization effect by adopting the SLA violation penalty in optimization objective. It can avoid requests for delay-tolerant services to preempt limited edge cloud resources and leave these resources for delay-sensitive services, and the network performance can be further improved.

And in the studies of load balancing, they mainly focus on balancing the traffic load on cloud nodes and links. For example, Pei *et al*. [16] proposed a heuristic approach based on multi-layer graph and traditional shortest path algorithm, then convert resource utilization to path weights for realizing minimizing resource load. Although we both use multi-layer graph to find paths as embedding schemes, our algorithm modified the traditional shortest path algorithm to realize the cost-efficient scheme for SFC requests. And Yang *et al* [17] proposed an approximation algorithm based on a randomized rounding scheme to achieve minimizing link load. Although the schemes which focus on load balancing can avoid the URP problem to some extent, its optimization effect is stochastic because it also does not optimize for the delay sensitivity of requests. This means that it cannot achieve the same optimization effect as our algorithm.

III. Motivation

We construct a toy testbed which includes three directly connected servers to simulate the edge cloud network and evaluate the impact of different schemes on the total embedding cost. The topology includes *Edge Cloud 1* (EC1), *Edge Cloud 2* (EC2) and one *Public Cloud*. The maximum number of VNF instances that EC1 and EC2 can hold are 1 and 4. Supposing the deployment cost is 3 if the cloud does not hold the required VNF image before, and 2 if the cloud holds [1]. The running cost is 1 on both edge and public clouds [2]. Two requests require SFC1 serving ($req1$ arrives before $req2$). Their source and destination are EC1 and EC2 respectively. The VNFs are some dockers running on the devices to process requests. Each request can be handled by one instance of VNF1 and VNF2. The traffic rate and expected completion time of requests are also shown in Fig. 1. We limit the port forwarding rate of servers to simulate different transmission delay on links. According to our test, the average delay on the two paths between EC1 and EC2 are 0.16 ms (link 1 - link 2)

[1]It requires time to download the VNF image from the remote server.

[2]The running cost is mainly determined by the resource cost of cloud location. We regard the resource cost of all clouds as the same.

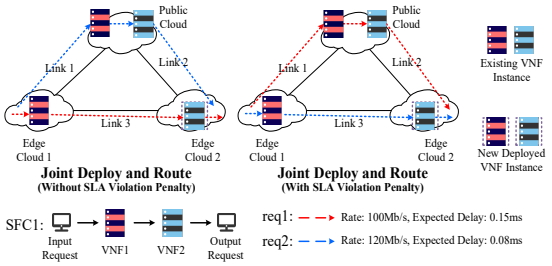


Fig. 1. Different Deployment and Routing Scheme Example

and 0.04 ms (link 3). The following schemes are considered: a) **Joint**: optimize VNF operation cost and traffic routing cost (VNF Cost + Total E2E Delay + Bandwidth Cost); b) **Joint SLA**: optimize VNF operation cost and traffic routing cost with SLA violation penalty (VNF Cost + Bandwidth Cost + SLA Violation Cost). The results are shown in Table I, we normalized the cost in different dimensions for comparison.

TABLE I
SIMULATION RESULT OF DIFFERENT SCHEMES

Scheme Type	VNF Cost [1]	Total E2E Delay (ms)	BW Cost [2]	SLA Violation Cost [3]	Total Cost [4]
Joint	0.35	0.2	0.34	0.08	0.77
Joint SLA	0.35	0.2	0.32	0.01	0.68

[1] VNF Cost = (Deployment Cost + RunningCost) / $\Omega, \Omega = 20$
[2] BW Cost = (Traffic Rate / $\Theta \times$ Link Number, $\Theta = 1000$
[3] SLA Violation Cost $= \alpha + \beta \times max(0,$ Actual E2E Delay $-$ Expected Delay), we set $\alpha = 0, \beta = 1$ here [18]
[4] Total Cost = VNF Cost + BW Cost + SLA Violation Cost

Before $req1$ and $req2$ arrive at the network, some clouds already have deployed VNF instances (e.g., one instance of VNF1 is deployed on EC1 as shown in Fig. 1). However, these deployed instances are not enough to handle the two requests, and new instances need to be deployed. In the **Joint** scheme, the objective is minimizing the VNF operation cost and traffic routing cost (bandwidth cost + E2E delay) for each arriving request. Since $req1$ arrives before $req2$, in order to save VNF deployment cost, $req1$ will use the existing VNF1 instance on EC1. On the other hand, $req1$ will select link 3 as the routing path for saving traffic routing cost. This requires to deploy a new VNF2 instance on EC2, and the routing cost savings are greater than the VNF deployment cost (compared with the scheme which selects link 1 - link 2 as the routing path and uses deployed VNF2 instance in public cloud). As for $req2$, due to existing VNF1 instance on EC1 has been occupied by $req1$ and the maximum number of VNF instances of EC1 can hold is 1, the minimal total cost embedding scheme is using existing VNF1 and VNF2 instance in public cloud and selecting link 1 - link 2 as the routing path. This results in high end-to-end delay for $req2$, which has lower delay tolerance than $req1$. By contrast, end-to-end delay is replaced with SLA violation cost in **Joint SLA** scheme. When serving $req1$, the **Joint SLA** scheme will use the existing VNF1 and VNF2 instances in public cloud and select link 1 - link 2 as routing path, which is the embedding scheme with minimal total cost (VNF operation cost + bandwidth cost + SLA violation cost). And for $req2$, it can use the embedding scheme which $req1$ uses in the **Joint** scheme. As a result, $req2$ no longer has a high SLA cost. This example indicates that combining SLA violation penalty in optimization objective is helpful for avoiding URP problem in edge clouds. Furthermore, minimizing SLA violation costs is more practical than simply minimizing end-to-end delays, since the latter may degrade the service quality of delay-sensitive services due to the URP problem under the premise of uncertain request arrival order.

IV. SYSTEM MODEL

In this section, we mainly describe the mathematic model of the SFC embedding cost minimization problem, and prove it to be NP-hard. The basic system model is shown in section IV-A, the definition of the cost to be optimized is shown in section IV-B, and the specific description of the optimization problem is shown in section IV-C.

A. The Edge Cloud System Model

The edge cloud system can be denoted by an undirected graph $G = (N, L)$. It includes F types of VNFs and works within a large time period T to handle user requests. The node set $N = E \cup C$ contains a set of edge cloud nodes E and a remote public cloud node C which can deploy virtual machines (VMs) to run VNFs. These VMs are called VNF instances (VNFIs). The processing capacity φ_f of each instance of VNF f represents the maximum flow rate that VNFI can process in each time slot, it depends on the resources capacities of VMs, including CPU, memory and so on. We assume that if the flow rate does not exceed the VNF processing capacity, the processing delay of VNF f is a mean expected value p_f. We also use $\pi(n)$ to denote the capacity (the max number of VNFIs can be hold) of edge cloud, while the public cloud has no capacity limitation. And the set L represents the links to connect the nodes in N. Each link l has its own forwarding capacity $b(l)$ and forwarding delay $\kappa(l)$. We denote the paths set between any two nodes n and n' by $P_{n,n'}$. And we also introduce $H_{n,n'}^{l,p}$ to represent whether link l belongs to a path $p \in P_{n,n'}$. The delay of p is denoted by θ_p, which is the sum of the $\kappa(l)$ of l belongs to p.

The request k arriving at time t also has following properties: (I) the source and destination nodes s_k and d_k; (II) required SFC J_k, which contains an order sequence of VNFs from set $[F]$ [3]. We use a binary indicator $h_k^{f,f'}$ to denote whether VNF f to f' exists a hop in the SFC. (if $f \to f'$ is a hop of J_k, $h_k^{f,f'} = 1$). (III) flow rate $R_k(t)$, it may alter when passing through specific VNF (e.g., Firewall, IDS, tunneling). we use λ_k^f to denote the change ratio of flow rate of request k on VNF f. Meanwhile, we use $R_k^f(t) = R_k(t) \prod_{f' \in J_k} \lambda_k^f$ (f' is the VNF before f) to denote the flow rate arriving at instance of VNF f of request k at time t; and (IV) expected completion time $M_k(t)$. And we assume that any single request cannot be split on multiple paths for forwarding.

[3]We only consider total-ordered linear SFC in this paper, non-linear SFC can be partitioned into several linear SFCs as description in [17].

We introduce two variables to determine the VFNs deployment in clouds and flow routing on physical links at each time slot $t \in [T]$: (I) $q_n^f(t)$ represents the number of instances of VNF f to deploy in cloud node n; (II) $X_{p,n,n'}^{k,f,f'}(t)$ represents whether request k goes from instances of VNF f in node n to VNF f' in node n' through path p.

B. Cost Structure

(1) **VNF Operation Cost:** The operation cost of VNF mainly refers to: a) time cost incurred by deploying VNFIs for booting VM image to edge or public clouds, and completing the state migration for stateful VNF. This process usually takes a number of seconds or minutes. Let η_n^f denote the cost for deploying the instance of VNF f in node n. And we use $\rho_n^f(t) = max\left\{0, q_n^f(t) - q_n^f(t-1)\right\}$ to denote the number of new instances of VNF f deployed in node n at time t. b) economic cost incurred by running VNFIs, which includes bill for power consumption, cloud container lease and so on. Let μ_n^f denote the cost for running an instance of VNF f in node n. The total VNF operation cost at time t can be calculated by:

$$C_O(t) = \sum_{n\in[N]} \sum_{f\in[F]} (\eta_n^f \rho_n^f(t) + \mu_n^f q_n^f(t)) \tag{1}$$

(2) **Link Bandwidth Cost:** Transferring requests between distributed clouds will incur bandwidth cost of links. Let δ_l denote the cost of forwarding a unit of a flow through link l. We use $I_{k,l}^{f,f'} = \sum_{n,n'\in[N]} \sum_{p\in[P_{n,n'}]} X_{p,n,n'}^{k,f,f'}(t) H_{n,n'}^{l,p}$ to present whether link l is used for request k between VNF f and f'. The overall bandwidth cost at time t is:

$$C_B(t) = \sum_{k\in[K]} \sum_{f,f'\in[F]} \sum_{l\in[L]} \delta_l I_{k,l}^{f,f'} h_k^{f,f'} \lambda_k^f R_k^f(t) \tag{2}$$

(3) **SLA Violation Cost:** If request cannot complete the required service before its expected completion time, it will incur additional time penalty. The service completion time of requests is denoted by $D_k(t)$, which includes traffic routing time and VNF processing time ($D_k(t) = \sum_{f,f'\in[J_k]} \theta_p X_{p,n,n'}^{k,f,f'}(t) + \sum_{f\in[J_k]} p_f$). Then we can calculate the timeout value $V_k(t)$ of flow k by $V_k(t) = max\left\{0, D_k(t) - M_k(t)\right\}$. We also introduce τ_k to denote the timeout penalty of request k. The total SLA violation at time t can be calculated by:

$$C_T(t) = \sum_{k\in[K]} \tau_k V_k(t) \tag{3}$$

C. The Offline Cost Minimization Problem

Suppose we can obtain full knowledge of the system in $[T]$, the offline SFC embedding problem can be formulated as a multi-objective optimization problem shown in Eq. (4). Since the problem has multiple objectives, the common method to treat this kind of optimization problem is: weight each objective of original problem, sum them up and form a new single objective optimization problem [19].

$$minimize \sum_{t\in[T]} C_O(t) + C_B(t) + C_T(t) \tag{4}$$

subject to:

$$\sum_{k\in[K]} \sum_{f'\in[F]} \sum_{n'\in[N]} \sum_{p\in[P_{n,n'}]} X_{p,n,n'}^{k,f,f'}(t) R_k^f(t) \le \varphi_f q_n^f(t), \quad \forall t \in [T], f \in [F], n \in [N] \tag{5a}$$

$$\sum_{f\in[F]} q_n^f(t) \le \pi(n), \forall t \in [T], n \in [N]\backslash C \tag{5b}$$

$$\sum_{k\in[K]} \sum_{f,f'\in[F]} \sum_{n,n'\in[N]} \sum_{p\in[P^{n,n'}]} X_{p,n,n'}^{k,f,f'}(t) H_{l,p}^{n,n'} \lambda_{k,f} R_k^f(t) \le b(l), \forall t \in [T], l \in [L] \tag{5c}$$

$$\sum_{n,n'\in[N]} \sum_{p\in[P_{n,n'}]} X_{p,n,n'}^{k,f,f'}(t) = 1, \forall t \in [T], k \in [K], \quad f, f' \in [F] \tag{5d}$$

$$\sum_{f'\in[F]} \sum_{n'\in[N]} \sum_{p\in[P_{n',n}]} X_{p,n',n}^{k,f',f}(t) = \sum_{f'\in[F]} \sum_{n'\in[N]} \sum_{p\in[P_{n,n'}]} X_{p,n,n'}^{k,f,f'}(t), \forall t \in [T], k \in [K], f \in [F], n \in [N] \tag{5e}$$

$$q_n^f(t) \in \{0, 1, 2, ...\}, \forall t \in [T], f \in [F], n \in [N] \tag{5f}$$

$$X_{p,n,n'}^{k,f,f'}(t) \in \{0, 1\}, \forall t \in [T], k \in [K], p \in [P_{n,n'}], \quad f, f' \in [F], n, n' \in [N] \tag{5g}$$

Constraint (5a) guarantees that the total incoming flow rate to instances of VNF f in node n does not exceed the processing capacity of the deployed instances. Constraint (5b) guarantees that the total deployed VNF instances in edge cloud does not exceed the capacity limitation. Constraint (5c) guarantees that the total flow rate through link l does not exceed the link capacity. Constraints (5d) and (5e) guarantee that for each request k which requires VNF f, f' (deployed in node n and n' respectively) in the SFC, only one path between node n and n' can be selected.

The optimization problem is NP-hard since it can be reduced from minimum knapsack problem (MKP) [20]. Due to space limitation, the detailed proof process is included in our online technical document [21]. Considering that directly solving the MILP problem with the existing solving tools (e.g., Gurobi [22]) will consume too much time, we propose a novel algorithm which can make SFC embedding decisions efficiently and minimize the cost defined in Eq. (4).

V. Algorithm Design

Since the objective of the optimization problem (4) is minimizing the total SFC embedding cost for all requests in a time period T, we are greedy to minimize the embedding cost for each arrival request in an online manner. In order to realize this objective, we need to determine the number of VNFIs deployed in cloud nodes and the traffic forwarding paths. Considering the above requirement, we propose to use the shortest path algorithm to find a minimal embedding cost scheme for requests. First, it needs to construct a multi-layered graph, the path weights in the graph are determined by the related operation and routing cost. Then it executes the

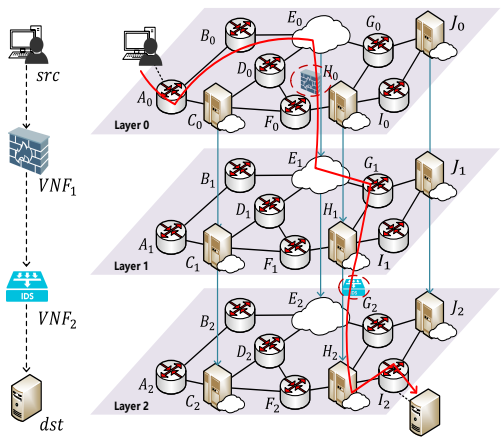


Fig. 2. Constructing multi-layer graph for an SFC request

modified shortest path algorithm to obtain a "path" for the SFC requests. According to the characteristics of the shortest path algorithm and the setting of path weights, the "path" found will be a feasible SFC embedding scheme with minimal embedding cost. The framework of SFC-CEB is shown in Algorithm 1.

Algorithm 1: SFC-CEB Algorithm

Input : $G = (V, L)$, N, K, T, η_n^f, μ_n^f, $\pi(n)$
Input : J_k, α_n, α_l, Φ, J_k, $R_k(t)$, $M_k(t)$, src_k, dst_k
Output: Embedding Scheme: sh_k

for *each* $t \in T$, *and each* $k \in K$ *arriving at* t **do**
  $G' \leftarrow$ `LayeredGraph(`G, J_k, η_n^f, μ_n^f`)`
  for *each* $v \in N, l \in L$ **do**
    $\alpha_n, \alpha_l \leftarrow$ `CalWeight(`v, l`)`
  end
  Initialize three lists $path$, dis and vis
  Initialize a heap $pq = \varnothing$ and push src_k into pq
  while $pq \neq \varnothing$ **do**
    $v \leftarrow pq.pop()$
    if $vis[v]$ **then** continue;
    $vis[v] \leftarrow True$
    for $u \in G'(v)$ **do**
      if `CheckPath(`$path[v]$`)` **then**
        `DFSNode(`u, $path$, G', $\pi(n)$, Φ`)`
        $d \leftarrow$ `CalDis(`v, u, α_n, α_l, $M_k(t)$`)`
        if $d < dis[u]$ **then**
          `Update(`u, $dis[u]$, $path[u]$, d`)`
          Push the node u into heap pq
        end
      end
    end
  end
  $sh_k \leftarrow path[dst_k]$
end

A. Multi-Layer Graph Construction

The layered network graph contains $h+1$ ($h = length(J_k)$) layers, and each layer is a copy of the original network graph. v_i is introduced to denote the corresponding node of vertex v in the original graph in the i^{th} layer. Any two adjacent layers can only be connected by vertical link between nodes v_{i-1} and v_i, where $v \in N$. The weights of these vertical links are related with three kinds of costs: (I) VNF deployment cost $\eta_{f,n}$, (II) VNF running cost $\mu_{f,n}$ and (III) VNF processing delay p_f. As for the weights of horizontal links in each layer, bandwidth cost δ_l and forwarding delay $\kappa(l)$ are considered. The specific weight settings are shown in the following subsection.

In the multi-layer graph, the ingress node of any request k is set in the 1^{st} layer. And k needs to pass through all h layers to get the services defined in J_k. When k reaches the egress node set in the $(h+1)^{th}$ layer, the total embedding cost can be calculated. Fig. 2 illustrates the construction of a multi-layer graph. The SFC contains two VNF: VNF_1 and VNF_2. Consequently, a three layer graph is constructed. $Layer\ 0$ is the original network topology. In the topology, node C, H, J are edge clouds, node E is public cloud and other nodes are switch nodes. The adjacent layers are connected to each other only through node C_i, H_i, J_i, E_i (since only these nodes can deploy VNFIs). The SFC request starts from the node A and ends at the node I. Supposing that the optimal embedding solution selects node E and H to deploy VNF instances, and the routing path in multi-layer graph is the red line shown in Fig. 2. Therefore, the final deployment and routing scheme in the original network topology is: $A \rightarrow B \rightarrow E(VNF_1) \rightarrow G \rightarrow H(VNF_2) \rightarrow I$.

B. Modified Shortest Path Algorithm

The shortest path algorithm can be applied to find an SFC embedding scheme with minimal cost in the multi-layer graph. VNF operation cost and bandwidth cost can be used directly as path weights. However, SLA violation cost is a piecewise function. It needs to obtain its actual end-to-end delay based on the forwarding path, and then calculate the final cost based on Eq. (3). This makes it hard to use SLA violation cost as weight directly in traditional shortest path algorithms.

In order to solve this issue, we construct a quintuple weight: $[dr\_c, sla\_c, bw\_c, res\_w, remn\_t]$. In the quintuple, (I) dr_c is the sum of VNF operation cost, if a vertical edge between layers is traversed (e.g. $E_0 \rightarrow E_1$), this value will be accumulated based on the operation cost of the deployed VNFIs; (II) sla_c is SLA violation cost, it can be updated according to $remn\_t$, if $remn\_t$ is negative then sla_c equals 0, otherwise sla_c can be calculated by Eq. (3); (III) bw_c is bandwidth cost, if a horizontal edge in any layer is traversed (e.g., $A_0 \rightarrow B_0$), this value will be accumulated based on Eq. (2). (IV) res_w is the sum of resource (node or link) weight. The weight is the reciprocal of the resource's remaining capacity multiplied by a constant factor. res_w is introduced to prevent low-cost resources from being preempted by first arrival requests, and it can be accumulated when resources are occupied. (V) $remn\_t$ is the remaining expected completion time, it starts with a negative value (minus expected completion time of SFC request). When a horizontal or vertical edge is traversed, this value will be accumulated based on the forwarding delay of links or processing delay of VNF instances (e.g., suppose a request whose expected completion

time is 500 ms arrives A_0, the $remn\_t$ is started with -500, when the request passes through the link $A_0 \rightarrow B_0$ and the forwarding delay of this link is 80 ms, then $remn\_t$ is updated to -420). By using this quintuple weight, we can calculate and compare the length of path in modified shortest path algorithm (the length of path is the sum of dr_c, sla_c, bw_c, res_w).

Algorithm 2: DFSNode

```
1  for u' ∈ G'(u) do
2      curDepth ← curDepth + 1 ;
3      if vis[u'] then continue;
4      if curDepth ≥ Φ then return;
5      if CheckPath(path[u]) then
6          DFSNode(u', path, G', π(n), Φ) ;
7          d ← CalDis(G', u, u', α_n, α_l, M_k(t)) ;
8          if d < dis[u'] then
9              Update(u', dis[u'], path[u'], d) ;
10             Push the node u' into heap pq ;
11         end
12     end
13 end
```

Before applying the modified shortest path algorithm, the current weight for each node in N and each link in L is calculated based on Eq. (6) (shown in Algorithm 1 line 3-5). ϵ is a constant factor and $remn(n)$ and $remn(l)$ are the remaining capacity of nodes and links. Then it needs to initialize three array variables: $path$, dis and vis. $path$ records the shortest path from source node to each node in layered graph, dis stores the length of path from variable $path$, and vis records whether node in layered graph is visited (avoid nodes being traversed repeatedly). We also maintain a heap pq to speed up traversing the nodes. The source node of request is pushed first in pq. Then we traverse the nodes in multi-layer graph to find the shortest paths to the destination node of request. Similar to the traditional shortest path algorithm (e.g., Dijkstra algorithm), we pop the node v from pq, and further traverse its neighbor nodes $u \in G'(v)$ to update the dis and $path$ during the traversal (line 8-22). The differences between our modified shortest path algorithm and Dijkstra algorithm are: we should check whether the current path is feasible (whether the node and link capacity limit is exceeded) in line 13. If the path is feasible, we will continue to traverse the neighbor nodes of u in $DFSNode$. This process is recursive (shown in Algorithm 2), it repeats to traverse the neighbor nodes (u') of current node (u) and updates the dis and $path$ according to the calculation results. The number of recursions is limited by Φ (Max DFS depth) and the $curDepth$ is set to 0 before executing $DFSNode$. After $DFSNode$, we will calculate the new length of path from u to n based on the quintuple weight in line 15. If the new length is smaller, dis and $path$ will be updated, and node u is pushed in pq in line 16-19. When all nodes in multi-layer graph are traversed, we can obtain a feasible path with optimal (or near-optimal) embedding cost for each request k in line 23.

$$\alpha_n = \frac{\epsilon}{remn(n)}, \alpha_l = \frac{\epsilon}{remn(l)}, \forall n, l \in N, L \tag{6}$$

In practice, in order to apply Algorithm 1 to produce a real embedding scheme, we can implement it as a module in the network control plane, and combine some existing network measurement method (e.g., NetFlow [25]) to obtain the traffic information of requests, then use cloud container orchestration platform (e.g., Kubernetes [26]) or network function deployment framework (e.g., OpenNetVM [27]) to embed required SFCs. For example, we need to capture the data of traffic rate, topology information, resource usage, etc, and use them as input to the SFC-CEB algorithm. After determining the deployment nodes of required VNFs and forwarding path between nodes according to the "path" found by SFC-CEB, we can use the YAML format to declare the VNFIs that are required to be created, and Kubernetes will automatically deploy them to the node we specify. We also need to design the routing table on the nodes which the forwarding path passes through to ensure that the traffic can be forwarded as our definition. After completing the VNFIs deployment and setting routing rules on data plane, the traffic can be served by required SFC with minimal embedding cost. Specifically, considering that the deployment process of new VNF instances often takes tens of seconds or even several minutes, but SFC requests require to get service before their lifetime is over. Therefore, SFC-CEB will only find SFC embedding scheme for long-lived requests (whose lifetimes are longer than the VNFIs deployment time). Requests with shorter lifetime can be handled by setting default processing, such as directly using a preset embedding scheme. We also analyze the reasons that cause SFC-CEB to find a near-optimal embedding scheme, and the competitive ratio of total embedding cost between SFC-CEB and offline optimal solution in our online technical document [21].

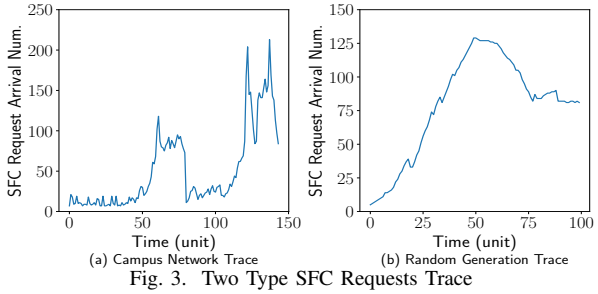
C. Complexity Analysis

In SFC-CEB, the complexity of initializing weight of nodes and links does not exceed $O([N] + [L])$. The construction of multi-layer graph requires to copy original network topology and connect adjacent layers with vertical links, which causes at most $O(h([L] + [N]))$ computations. In the multi-layer graph, there will be no more than $(h + 1)[N]$ nodes and $(h + 1)[L] + h[N]$ links. For given a network topology $G = (N, L)$, the complexity of the shortest path algorithm (e.g., Dijkstra algorithm based on heap optimization) is $O([L]log[N])$. Considering we introduce additional path feasibility checking (complexity: $O(h[N])$) and DFS (complexity: $O((h[L])^{\Phi})$) process, the total complexity of SFC-CEB is $O(((h[L])^{\Phi+1} + h^2[L][N])log(h[N]))$.

VI. Performance Evaluation

A. Simulation Setup

We realize our SFC-CEB algorithm in Python code, and simulate the network scenario with NetworkX [24]. Some details of network parameters are shown as follows:



(a) Campus Network Trace (b) Random Generation Trace

Fig. 3. Two Type SFC Requests Trace

(1) **Topology Dataset**: We use the network topology Abilene (11 nodes, 14 links) and Uunet (49 nodes, 84 links) from TopologyZoo [23]. In each topology, we randomly select several nodes as edge clouds and one node as public cloud. The capacities of edge clouds are set randomly between $[20, 40]$ $units$ (one $unit$ refers to holding one VNFI), and the capacity of public cloud is infinite. The forwarding delays of links are generated randomly between $[3, 70]$ ms and the link bandwidth capacities are set randomly between $[10, 20]$ Gb/s.

(2) **Traffic Dataset**: We use both real and randomly generated SFC request traces shown in Fig. 3: (a) Real Campus network traffic (we treat traffic with same source-destination IP as a single SFC request, and count the number of requests arriving in each time unit) and (b) Random generation. The ingress/egress nodes and required VNFs of requests are set randomly. The flow rates and expected completion times of requests are randomly picked in $[1, 10]$ Mb/s and $[50, 800]$ ms respectively. And the flow size change ratio λ_k^f is picked in $[0.8, 1.2]$. Each request also has its own lifetime which is randomly generated. It requires to provide enough resources for serving SFC requests during the lifetime, and release the resources occupied by expired requests.

(3) **VNF and Cost Data**: There are 10 different types of VNF, whose processing capacity φ_f ranges from 5 to 10 $units$ (one $unit$ can process 10 Mb traffic per second). Each SFC contains 2 to 5 VNFs. In practice, the operation cost of VNFs can be inferred from local energy price of clouds, the bandwidth cost is determined by communication service providers, and the SLA violation cost can be priced by network service providers themselves. For simplifying comparison but without loss of generality, we randomly generate the VNF operation cost parameters ($\eta_n^f \in [1, 6]$ and $\mu_n^f \in [1, 3]$), bandwidth cost of link ($\delta_l \in [0.1, 0.8]$) and timeout penalty factor ($\tau_k \in [0.002, 0.01]$) for different requests to simulate the real situation. We adjust cost parameters with different magnitudes in a similar scale to let them be accumulated directly for calculating the total embedding cost.

B. Compared Algorithm

(1) **RDIP:** RDIP [6] is a state-of-art algorithm for jointly optimizing VNF operation cost, end-to-end delay and bandwidth cost in edge cloud network. It solves the relaxed linear programming problem, and applies the randomized dependent rounding algorithm to obtain the integer results for VNFIs provisioning. Finally, it modifies the traffic routing decision and yields a feasible solution. RDIP is used to explain the drawback of minimizing the total embedding cost without considering the delay sensitivity of requests.

(2) **SFC-MAP:** SFC-MAP [16] constructs a multi-layer graph for required SFC, and assigns weight factors (which depends on residual capacity of resources) to nodes and links. Then it runs the shortest path algorithm to find the forwarding path with minimal sum of weights. If the path does not satisfy the delay requirement or resource capacity limitation, the above process will be repeated, while links or nodes which do not meet the requirements will be imposed with a large penalty factor. We set the maximum number of repetitions of this procedure to 50. SFC-MAP is also a state-of-art algorithm which aims to realize load balancing. It is used to evaluate the effect of load balancing on embedding cost optimization.

(3) **Offline Solving MILP:** We use the MILP solver Gurobi to solve the offline optimization problem defined in Eq. (4) and obtain the offline optimal solution. It is used to evaluate the gap between SFC-CEB and the theoretical optimal solution

C. Simulation Results

(1) **Evaluation of Different Type Cost:** We compare the algorithms in Uunet topology. The cost types used to evaluate algorithm performance include: VNF operation cost, SLA violation cost, bandwidth cost and total SFC embedding cost defined in Eq. (4). According to the request arrival rate shown in Fig. 3, we record the cost consumed to serve requests at each time unit. As shown in Fig. 4 a) and c), SFC-CEB can realize the lowest total embedding cost. Over the entire time period of trace, SFC-CEB can reduce the total embedding cost by up to 37% and 33% (19% and 21% on average) compared with RDIP. And compared with SFC-MAP, SFC-CEB can reduce the total embedding cost by up to 46% and 35% (28% and 21% on average). Especially, when the request volume is heavy (e.g., in campus network trace 60-80 and 120-135 time units), SFC-CEB optimizes the embedding cost more effectively. Meanwhile, according to the distribution of total cost shown in Fig. 4 b) and d), SFC-CEB can always realize lower total embedding cost with higher probability.

Next, we evaluate the the optimization effect of these algorithms on some specific costs. The first one is VNF operation cost. As shown in Fig. 5, the RDIP algorithm can achieve better optimization effect when the request volume is heavy. The reason is that SFC-CEB and SFC-MAP will not allow the low-cost resources being exhausted (they reverse some low-cost resources to serve subsequent requests), this may lead higher VNF operation cost in sometime. However, RDIP does not consider this situation, it always tries to minimize the VNF operation cost, end-to-end delay and bandwidth cost jointly. Hence, in some case, RDIP can achieve lower VNF operation cost compared with SFC-CEB and SFC-MAP.

According to the results shown in Fig. 6 and Fig. 7, SFC-CEB can achieve the best optimization effect for SLA violation cost and bandwidth cost. Although RDIP also tries to optimize end-to-end delay and bandwidth cost, the result is not well. The reason is that RDIP allocates resources with low routing costs to any incoming requests. When the resources with lower

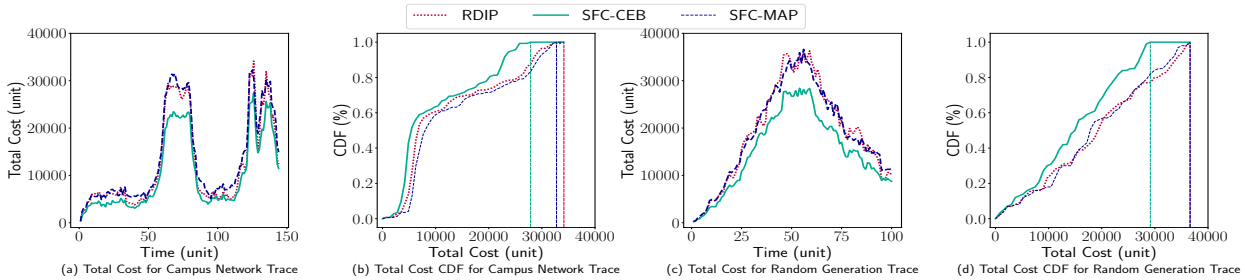


Fig. 4. The Comparison of Total Embedding Cost and the Corresponding CDF

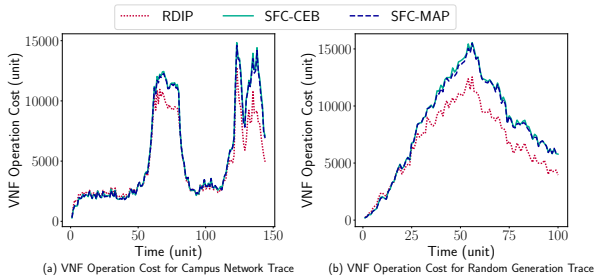


Fig. 5. The Comparison of VNF Operation Cost

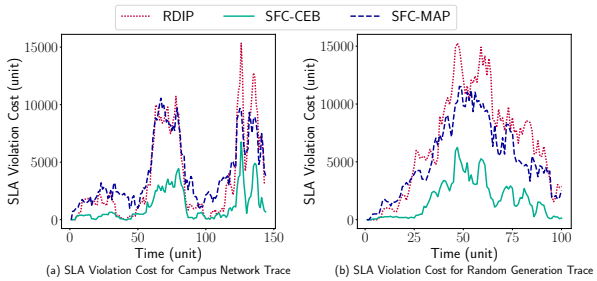


Fig. 6. The Comparison of SLA Violation Cost

routing cost are exhausted by previously arrived requests, and subsequent requests can only be allocated to resources with higher routing costs. This may incurs a large amount of SLA violation cost. However, SFC-CEB allocates resources according to the request's tolerance for delay. It can maintain relatively low routing costs even when the number of requests is large. SFC-MAP can achieve better results than RDIP, since it also save some resources with lower routing cost due to the objective of load balancing. However, it is still less effective than SFC-CEB in routing cost optimization, because it does not allocate resources appropriately based on the delay sensitivity of the requests. In terms of the above results, SFC-CEB can make a trade-off between VNF operation cost and traffic routing cost, and then achieves the optimal total cost.

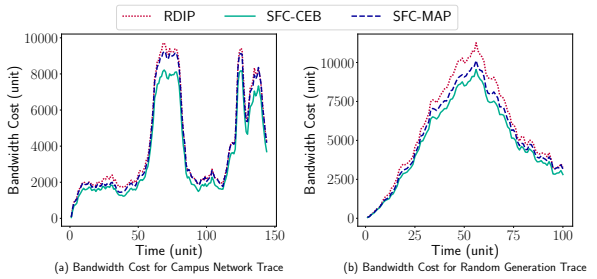


Fig. 7. The Comparison of Bandwidth Cost

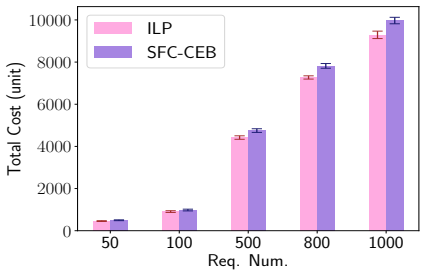
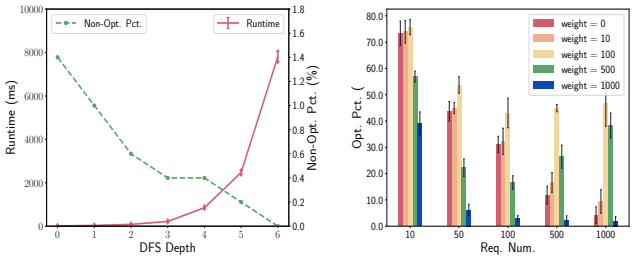


Fig. 8. The Comparison between SFC-CEB and Offline Optimal Solution

(2) **Cost Gap Compared with Optimal Solution:** Solving MILP problem defined in Eq. (4) will incur huge time overhead when the topology size and number of requests is large. In order to shorten the computation time of solving MILP in Gurobi, we replace Uunet topology with Abilene topology. Meanwhile, we also randomly generate several sets $[K]$ which includes 50, 100, 500, 800, 1000 SFC requests to replace the previous two sets of request traces. These sets of requests will be offered together to Gurobi for offline calculation, and they will be offered in sequence to the SFC-CEB algorithm to simulate the online status. The simulation result is shown in Fig. 8. The total embedding cost achieved by SFC-CEB is more than $7.3\% \sim 8.8\%$ of the optimal solution on average.

(3) **Relationship between Accuracy and Running Time:** SFC-CEB algorithm includes a DFS procedure, and the depth of DFS will affect the algorithm accuracy and running time. We use Abilene topology and run SFC-CEB 500 times to count the percentage that fails to find the optimal solution (compared with offline optimal solution) at each different DFS depth. As shown in Fig. 9 (a), although the percentage of failures to find the optimal solution becomes 0 when depth is 6, the running time also increases to about 8 seconds. In general, when the DFS depth equals 1 or 2, the percentage of failures to find the optimal solution has already been tolerable, and the running time is about 30 to 80 ms, which is much less than the request lifetime or launching time of VNF instances.

(4) **Impact of Weight Factor on Optimization Effect:** α_n and α_l are introduced to avoid low-cost resources being exhausted, and save them to serve future coming requests. The optimization effect of SFC-CEB can be affected by the setting of resource weight. We randomly generate several sets $[K]$ which includes 10, 50, 100, 500, 1000 SFC requests, and execute SFC-CEB with different constant factor ϵ (pick in [0, 10, 100, 500, 1000]) 500 times for each given request set. We evaluate the percentage of the number of times that



(a) Impact of Different DFS Depth (b) Impact of Different Resource Weight Factor

Fig. 9. The Impact of Different DFS Depth and Resource Weight Factor

these algorithms realize the minimum total embedding cost. As shown in Fig. 9 (b), if the algorithm has a high value of ϵ (i.e. 1000), the probability of finding the lowest cost embedding scheme decreases when the number of requests increases. Because excessive weight results in reserving more resources for future requests, current requests cannot utilize low-cost resources and cause a high total embedding cost. $\epsilon = 0$ means that the algorithm will not reverse resources, it may cause the low-cost resources are occupied by first coming requests, and incur higher total embedding cost for future requests. Hence, it is better to set relatively low constant factor ϵ (e.g., 100) for resources when the network load is heavy.

VII. Conclusion

In this paper, we formulate the SFC embedding problem in geo-distributed edge cloud network, which minimizes the total SFC embedding cost. Combining with SLA violation cost in optimization problem can effectively avoid inappropriate resource competition on the edge cloud. In a way, this can help to reduce the cost of service degradation and improve overall network performance. We also presented an online SFC requests embedding algorithm called SFC-CEB, which can efficiently find an approximate optimal embedding scheme for each SFC request. The simulation results reveal that our algorithm outperforms other algorithms in terms of total embedding cost. In our future work, we plan to implement our algorithm in real-world testbed and further improve the execution efficiency of our algorithm.

VIII. Acknowledgements

This research is supported in part by the National Key Research and Development Program of China under Grant 2018YFB1800400 and the National Natural Science Foundation of China (No. 61802092 and No. 62002009).

References

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

[2] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657-1681, 2017.

[3] L. Jiao, L. Pu, L. Wang, X. Lin, and J. Li, "Multiple granularity online control of cloudlet networks for edge computing," in *Proc. IEEE SECON*, pp. 1–9, Jun. 2018.

[4] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng, and et al., "Network functions virtualisation introductory white paper," in *SDN and OpenFlow World Congress*, 2012.

[5] 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 Commun. Magazine*, vol. 55, no. 2, pp. 216–223, 2017.

[6] Z. Zhou, Q. Wu, and X. Chen, "Online Orchestration of Cross-Edge Service Function Chaining for Cost-Efficient Edge Computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1866-1880, Aug. 2019.

[7] R. Li, Z. Zhou, X. Chen, and Q. Ling, "Resource Price-Aware Offloading for Edge-Cloud Collaboration: A Two-Timescale Online Control Approach," *IEEE Trans. Cloud Comput.*, 2019.

[8] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient NFV-enabled mobile edge-cloud for low latency mobile applications," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 1, pp. 475-488, 2018.

[9] R. Gouareb, V. Friderikos, and A. Aghvami, "Virtual Network Functions Routing and Placement for Edge Cloud Latency Minimization," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2346-2357, Oct. 2018.

[10] D. Chemodanov, P. Calyam, and F. Esposito, "A Near Optimal Reliable Composition Approach for Geo-Distributed Latency-Sensitive Service Chains," in *Proc. IEEE INFOCOM*, Paris, France, 2019, pp. 1792-1800.

[11] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 4, pp. 725-739, Dec. 2016.

[12] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF Chain Deployment with Efficient Resource Reuse at Network Edge," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, 2020, pp. 267-276.

[13] Y. Jia, C. Wu, Z. Li, F. Le, and A. Liu, "Online Scaling of NFV Service Chains Across Geo-Distributed Datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 699-710, April 2018.

[14] Z. Luo, C. Wu, Z. Li, and W. Zhou, "Scaling Geo-Distributed Network Function Chains: A Prediction and Learning Framework," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1838-1850, Aug. 2019.

[15] L. Gu, D. Zeng, W. Li, S. Guo, A. Zomaya, and H. Jin, "Deep Reinforcement Learning Based VNF Management in Geo-distributed Edge Computing," in *Proc. IEEE ICDCS*, Dallas, TX, USA, 2019, pp. 934-943.

[16] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-Distributed Cloud System," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2179-2192, 1 Oct. 2019.

[17] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, "Delay-aware virtual network function placement and routing in edge clouds," *IEEE Trans. Mobile Comput.*, to be published.

[18] L. Wu, S. K. Garg, and R. Buyya, "SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments," in *Proc. IEEE/ACM CCGRID*, Newport Beach, CA, 2011, pp. 195-204.

[19] S. Boyd, and L. Vandenberghe, *Convex Optimization. Cambridge*, U.K.: Cambridge Univ. Press, 2004.

[20] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer, 2013.

[21] *Technical Document for Cost-Efficient Dynamic Service Function Chain Embedding in Edge Clouds*. [Online]. Available: https://github.com/130x00/online-document.git

[22] Gurobi, https://www.gurobi.com/

[23] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765-1775, 2011.

[24] NetworkX, https://networkx.org/

[25] B. Claise, G. Sadasivan, V. Valluri, "Cisco systems netflow services export version 9", RFC 3954,2004.

[26] Kubernetes, https://www.docker.com/

[27] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K.K. Ramakrishnan, T. Wood, "OpenNetVM: A platform for high performance network service chains," *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, 2016.