

Delay-Aware Service Function Chain Embedding and Scheduling in Edge Clouds

Weihan Chen , Zhiliang Wang , *Member, IEEE*, Han Zhang , *Member, IEEE*, Xia Yin , *Senior Member, IEEE*, and Xingang Shi , *Member, IEEE*

Abstract—Edge Computing (EC) provides low-latency environment for delay-sensitive network services by deploying limited cloud infrastructure at the edge of the network. Network Function Virtualization (NFV) further enhances flexibility by replacing hardware with Virtual Network Functions (VNFs) that can run on general servers. In NFV, Service Function Chaining (SFC) can provide network services by combining several VNFs in a chain or graph format. 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. Optimizing SFC embedding and performance is crucial for service providers to reduce costs and maximize benefits. This paper investigates dynamic SFC embedding in geo-distributed edge clouds for requests with diverse delay requirements. We propose SFCES, a framework integrating a cost-efficient embedding (SFC-CEB) algorithm with a request rerouting (RR) algorithm. SFC-CEB handles initial embedding based on delay needs, while RR periodically re-optimizes routing for active requests. Experimental results show that SFCES reduces embedding cost by 46% ~ 64% compared to state-of-the-art schemes like RDIP.

Index Terms—Network function virtualization, edge computing, service function chaining, traffic routing.

I. INTRODUCTION

EXISTING mobile communication (e.g., 5G) and Internet of Things (IoT) technologies have significantly expanded the range of mobile network services, most of which are computationally intensive with heterogeneous latency requirements. Deploying delay-sensitive services on remote public clouds often violates latency constraints due to backbone propagation delay. Consequently, Edge Computing (EC) offers localized, resource-constrained infrastructure to meet the dual demands of high-performance computing and ultra-low latency [1], [2].

Similar to data centers, EC requires efficient cost management to minimize operational expenses. [3]. Network Function Virtualization (NFV) addresses the high costs and management

complexity of dedicated hardware by implementing functions as software-based Virtual Network Functions (VNFs) on general servers, enabling flexible resource scheduling [4]. A Service Function Chain (SFC) [5] comprises an ordered sequence of VNFs providing customized network services. To further optimize costs, multiple SFCs can share common VNFs by running a single instance of them [6]. SFC embedding involves two core tasks: (I) VNF placement and (II) traffic routing.

Existing SFC optimization typically jointly minimizes VNF operational (launching and running VNF instances) and traffic routing (bandwidth and transmission latency) costs. However, minimizing end-to-end (E2E) delay without considering request sensitivity often triggers Unreasonable Resource Preemption (URP). Actually, 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]. URP occurs when delay-tolerant services exhaust low-latency edge resources during high request volumes, forcing delay-sensitive services to higher-latency nodes or remote public clouds, thereby degrading performance. Consequently, a critical challenge arises: *minimizing SFC embedding costs while preventing service degradation caused by URP*.

To align resource allocation with request delay requirements, we introduce Service Level Agreement (SLA) violation cost to replace the conventional end-to-end delay metric. Defined as the penalty incurred when actual E2E delay exceeds a request's expected completion time, this metric mitigates cost burdens when deploying delay-tolerant services on higher-latency clouds. Consequently, low-latency edge resources are reserved for delay-sensitive services, reducing resource competition. We formulate the SFC embedding cost minimization problem (comprising VNF operation, bandwidth, and SLA violation costs) as Mixed Integer Linear Programming (MILP). Due to its complexity, we propose the SFC Cost-Efficient emBedding (SFC-CEB) algorithm, which employs a modified shortest-path approach to determine near-optimal embedding schemes for arriving requests on-the-fly.

Cloud resources remain occupied throughout a request's lifecycle. If initial SFC embeddings are static, new arrivals may still suffer from URP. For example, earlier delay-sensitive requests (tolerance ~ 100 ms) might monopolize edge resources, forcing later requests with more stringent requirements (tolerance ~ 50 ms) to public clouds. As the SFC-CEB algorithm does not adjust active embeddings, dynamic reallocation becomes

Received 21 January 2022; revised 10 May 2026; accepted 7 September 2026.
Date of current version 2 October 2026. Recommended for acceptance by A. Anjum. (Corresponding authors: Zhiliang Wang; Xia Yin.)

Weihan Chen is with Zhongguancun Laboratory, Beijing 100084, China (e-mail: chwh@zgclab.edu.cn).

Zhiliang Wang, Han Zhang, and Xingang Shi are with the Institute for Network Sciences and Cyberspace, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China (e-mail: wzl@cernet.edu.cn; zhhan@tsinghua.edu.cn; shixg@cernet.edu.cn).

Xia Yin is with the Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China (e-mail: yxia@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TPDS.2026.3735294

essential. While VNF migration is often too time-consuming for latency-sensitive tasks, routing path re-selection offers a more practical alternative [8]. Consequently, we formulate a routing reallocation optimization problem to jointly minimize routing re-allocation, SLA violation, and bandwidth costs, solved by our proposed Routing Reallocation (RR) algorithm.

These algorithms constitute the SFCES (SFC Embedding and Scheduling) framework. SFCES leverages SFC violation cost to handle initial embedding,¹ while periodically re-optimizing routing for active requests to minimize traffic costs. The main contributions of this study are:

- We demonstrate that traditional E2E-delay-centric optimization overlooks delay sensitivity, leading to high SLA penalties. We propose a joint optimization model incorporating SLA violation, VNF operation, and bandwidth costs.
- Building upon cost-efficient embedding, we introduce a framework that synergizes online embedding for new requests with periodic rerouting for existing flows. This more effectively mitigates URP under unpredictable request arrivals.
- Through simulations and a physical testbed with real-world traces, we show that SFCES reduces total embedding costs by 46% ~ 64% compared to state-of-the-art schemes like RDIP [6].

II. RELATED WORK

Resource scheduling and computation offloading in Edge Computing (EC) have been extensively surveyed in [8], [10]. From a communication perspective, [1] investigates the joint management of radio and computational resources, while [11] demonstrates the role of Software-Defined Networking (SDN) in addressing EC implementation challenges. Furthermore, SFC embedding is recognized as a critical resource allocation challenge within the NFV paradigm [12].

A. Cost Optimization in SFC Embedding

Most SFC embedding studies in geo-distributed clouds optimize either VNF deployment or routing costs independently. For VNF deployment optimization, Luizelli et al. [13] proposed the OCM algorithm to optimize VNF deployment and improve resource utilization. Ceselli et al. [14] applied column generation and iterative rounding to obtain VNF deployment schemes that minimize deployment costs. Yang et al. [15] introduced a heuristic algorithm for incremental resource allocation and an approximation algorithm for periodic re-optimization of global resource allocation. Lin et al. [16] proposed three heuristic algorithms based on an auxiliary graph to minimize the energy consumption of nodes and links during SFC embedding. For routing cost optimization, Gouareb et al. [17] and Chemodanov et al. [18] focused on minimizing end-to-end delay or meeting the Quality of Service (QoS) requirements of SFCs by proposing heuristic algorithms. Zheng et al. [19] proposed the PSFD algorithm to optimize end-to-end service delay, while considering the parallel

execution of VNFs within an SFC. However, independent optimization often compromises global efficiency: isolating deployment may neglect critical routing constraints, degrading service quality; conversely, routing-only focus necessitates excessive VNF instances, inflating operational overhead. Consequently, such disjointed approaches fail to achieve the efficacy of joint optimization schemes.

To address joint optimization of VNF deployment and routing costs, several works have proposed heuristic algorithms. Bari et al. [20] developed a dynamic programming-based algorithm to find near-optimal SFC placement schemes. Ghaznavi et al. [21] proposed the KARIZ algorithm, which uses local search to optimize both VNF deployment costs and link bandwidth allocation. Fischer et al. [22] introduced a simulated annealing-based heuristic to optimize deployment for delay-sensitive SFCs. Khoshkholghi et al. [23] designed two heuristic algorithms utilizing genetic and bee colony algorithms, to jointly optimize VNF deployment costs and routing delay. Jin et al. [24] presented a two-phase algorithm that pre-routes paths and selects the lowest-cost path for each request's deployment. Some studies also offer reliable performance bounds. Guo et al. [25] proposed the MWU algorithm, based on a primal-dual approach, to maximize utility for traffic receiving SFC services. Qu et al. [26] developed the REACH algorithm to minimize bandwidth costs and ensure VNF reliability standards. Jia et al. [27] and Zhou et al. [6] proposed online algorithms using regularization and rounding techniques to derive solutions with provable performance guarantees. Guo et al. [28] proposed an approximation algorithm for the embedding of aggregated service function trees to optimize VNF deployment and link bandwidth costs.

Beyond optimization models, learning-based approaches [29], [30], [31] leverage deep reinforcement learning for SFC embedding. While these model-free methods bypass complex formulations, they often lack interpretability and formal convergence guarantees. Crucially, most neglect request-specific delay sensitivity, potentially exacerbating SLA violations through URP. Conversely, our framework integrates SLA violation penalties into the objective. By jointly optimizing VNF operational and routing costs, SFCES prioritizes resource allocation based on delay requirements, thereby enhancing global network performance.

B. Cost Efficient Flow Migration in NFV

In NFV, reallocating traffic flows across VNF instances is essential to sustain service performance. This flow migration involves transferring active requests between VNF instances, often necessitating routing path updates if instances reside on distinct cloud nodes. However, migration incurs significant overheads, including bandwidth and latency penalties from stateful parameter transfers, service downtime, and path reconfiguration. These costs are particularly critical for delay-sensitive SFC requests.

Most existing efforts focus on selecting flows to migrate to reduce migration costs. For instance, Sun et al. [32] proposed heuristic algorithms to address flow migration in scenarios of scale-out, scale-in, and load balancing, while Sun et al. [33]

¹SFC-CEB was introduced in our preliminary work [9].

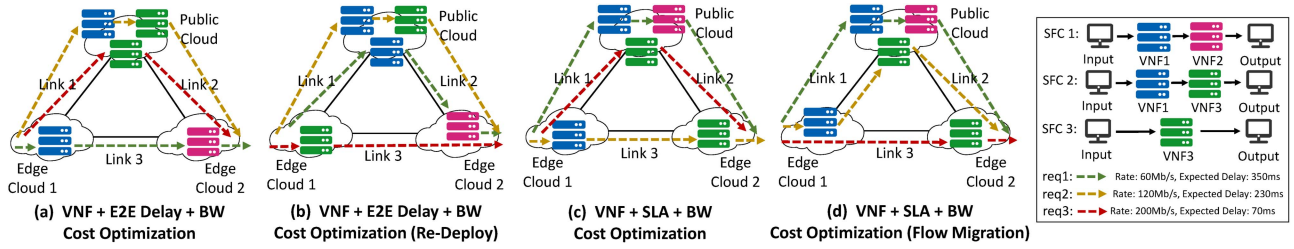


Fig. 1. Different SFC Embedding Scheme Example.

employed graph-aware deep reinforcement learning for flow migration optimization. Both works focus on single-VNF scenarios, without addressing the SFC case that requires routing between VNFs. Qu et al. [34] developed a heuristic algorithm using hop delay bounds to manage dynamic flow migration in 5G core networks, but it only guarantees average delay within acceptable downtime for delay-sensitive services. Without considering variations in delay sensitivity among requests, these approaches fall short in optimizing SLA violation costs related to the URP issue during migration.

III. MOTIVATION

We construct a toy testbed with three interconnected servers: Edge Cloud 1 (EC1), Edge Cloud 2 (EC2), and Public Cloud (PC), to evaluate SFC embedding costs. Due to resource constraints, each edge cloud hosts only one VNF instance (VNFI), implemented via Docker containers. VNF operational costs are modeled based on instantiation time and execution energy. Specifically, launching a VNFI costs 4 units if the image requires remote downloading and 2 units otherwise.² We assume the PC maintains a complete image cache. To reflect realistic operational variances, we assign execution energy costs of 2, 4, and 3 to EC1, EC2, and PC, respectively.³

Three requests with varying delay sensitivities enter the network in the sequence *req1*, *req2*, and *req3*, and remain active concurrently. Specifically, *req1* is delay-tolerant, while *req2* and *req3* are delay-sensitive, with *req3* having the most stringent latency constraints. All requests traverse from EC1 to EC2, where each VNF instance serves only one request at a time. Detailed traffic rates and expected completion times are provided in Fig. 1. To simulate heterogeneous transmission environments, forwarding delays are fixed using the Linux traffic control model,⁴ yielding average delays of 400 ms (Path: link 1–link 2) and 40 ms (link 3), respectively.

We evaluate four solution types: (a) static joint optimization of VNF, E2E delay, and bandwidth costs (e.g., RDIP [6]); (b) the same as (a) with redeployment; (c) static optimization replacing E2E delay with SLA violation cost (e.g., SFC-CEB [9]); and (d) our proposed SFCES, which combines (c) with periodic routing re-optimization. Under (a) and (b), *req1* arrives first and

selects scheme **s1** (deploying VNFI on edge nodes via link 3) due to its lower normalized cost (sum of normalized VNF Cost, E2E Delay Cost, and BW Cost). However, this exhausts edge resources, forcing subsequent *req2* and *req3* to adopt scheme **s2**, which deploys VNFI on the Public Cloud (PC) via the high-latency Path (link 1 – link 2). This mismatch results in severe SLA violations for these delay-sensitive requests compared to the tolerant *req1*.

In solution (c), replacing E2E delay with SLA violation cost alters the initial selection: *req1* opts for the aforementioned scheme **s2** due to its lower normalized cost under the new metric. For *req2*, scheme **s3** (deploying VNFI on ECs via link 3) is selected over **s4** (deploying VNFI on PC via link 1 – link 2), thereby exhausting limited edge resources. Consequently, the late-arriving *req3* is forced onto the PC via the high-latency Path (link 1–link 2), incurring significant SLA penalties due to its tighter latency constraints relative to *req2*. This demonstrates that static SLA-aware optimization (Solution c) remains susceptible to the URP problem and fails to yield the global optimum.

Our proposed SFCES addresses these limitations via periodic routing re-optimization. As illustrated in Fig. 1(d), *req3* is rerouted to the VNFI3 instance on EC2 (formerly serving *req2*), while *req2* is redirected to the PC. Although this rerouting incurs a 20 ms latency (measured on our testbed and encompassing state transitions and routing updates), SFCES significantly reduces the aggregate SLA violation cost by prioritizing the more delay-sensitive *req3*.

Similarly, Solution (b) can reconfigure the SFC embedding scheme after all requests arrive to optimize aggregate costs. Building on Solution (a), it instantiates VNFI3 on EC1 for *req3* and VNFI1 on the PC for *req1*, updating the latter's path to link 1 – link 2. Due to *req3*'s dominant traffic rate, the bandwidth-related savings from this adjustment outweigh the increased VNF operational and E2E delay costs. Furthermore, prioritizing *req3*, which has the lowest delay tolerance, enables Solution (b) to achieve an SLA violation cost comparable to SFCES, resulting in a lower total cost than Solution (c). However, the additional VNFI instantiation increases the operational expenses of Solution (b). Consequently, as demonstrated in Table I, SFCES achieves the optimal balance, yielding the lowest total deployment cost without compromising E2E delay performance.

This example demonstrates that integrating SLA violation costs into the SFC embedding objective prevents resource contention between delay-tolerant and delay-sensitive services.

²Additional time is required to pull VNF images from remote registries.

³In practice, these costs fluctuate based on local electricity rates and server efficiency.

⁴Linux Traffic Control: <https://lartc.org/>

TABLE I
SIMULATION RESULT OF DIFFERENT SOLUTIONS

Scheme Type	VNF Cost ¹	E2E Delay Cost ²	BW Cost ³	SLA Violation Cost ⁴	Total Cost ⁵
(a)	0.290	0.420	0.467	0.500	1.257
(b)	0.350	0.440	0.373	0.240	0.963
(c)	0.290	0.420	0.427	0.380	1.090
(d)	0.290	0.440	0.373	0.240	0.903

¹ VNF Cost = (launching cost + energy cost) / ϵ_1 , $\epsilon_1 = 100$

² E2E Delay Cost = Total E2E Delay / ϵ_2 , $\epsilon_2 = 2000$

³ BW Cost = (Traffic Rate / ϵ_3) $\times$ Link Number, $\epsilon_3 = 1500$

⁴ SLA Violation Cost = $\alpha + \beta \times \max(0, \text{Actual E2E Delay} - \text{Expected Delay})$, we set $\alpha = 0$, $\beta = 0.001$ here [35]

⁵ Total Cost = VNF Cost + BW Cost + SLA Violation Cost

Furthermore, re-optimizing only the request routing scheme ensures that priority flows utilize low-latency edge resources while avoiding the substantial instantiation and state-migration overhead associated with full VNF redeployment. Compared to simple E2E delay minimization, this mechanism more practically mitigates URP-induced service degradation under unpredictable request arrival sequences.

IV. SYSTEM MODEL

In this section, we describe the mathematical 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 related cost is shown in Section IV-B, and the specific descriptions⁵ of the optimization problems are shown in Sections IV-C, IV-D, and IV-E.

A. The Edge Cloud System Model

The edge cloud system is modeled as an undirected graph $G = (N, L)$, which supports F types of VNFs and operates over a time horizon 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 , both of which can deploy VNF instances (VNFIs). The processing capacity of an instance of VNF f , denoted by φ_f , represents the maximum flow rate it can handle per time slot, determined by its allocated resources (e.g., CPU, memory). 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 . The maximum number of VNFIs that can be deployed at edge node $n \in E$ is denoted by $\pi(n)$, 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 $\kappa(l)$. Let $P_{n,n'}$ be the set of available paths between nodes n and n' . And $H_{n,n'}^{l,p}$ is used to represent whether link l belongs to a path $p \in P_{n,n'}$. The total delay of path p is denoted by $\theta_p = \sum_{l \in L} H_{n,n'}^{l,p} \cdot \kappa(l)$.

A request k arriving at time t is characterized by the following properties: (I) the source and destination nodes s_k and d_k ; (II) the required SFC J_k , which contains an ordered sequence of VNFs

from set $[F]$.⁶ Let h_k denote the number of VNFs contained in the SFC:

$$J_k : f_1 \prec f_2 \prec \dots \prec f_i \prec \dots \prec f_{h_k} \quad (1)$$

(III) the flow rate $R_k(t)$, which may vary when passing through specific VNFs (e.g., firewall, IDS). We use λ_k^f to denote the flow scaling ratio of request k at VNF f . Let $R_k^f(t) = R_k(t) \prod_{f' \in J_k} \lambda_k^{f'}$ (f' is the VNF before f) denote the flow rate arriving at instance of VNF f of request k at time t ; and (IV) the expected completion time $M_k(t)$. Furthermore, we assume that each request cannot be split on multiple paths for forwarding.

We define three variables to determine the VNF deployment and flow routing at each time slot $t \in [T]$: (I) $q_n^f(t)$: The number of instances of VNF f deployed at node n ; (II) $X_{p,n,n'}^{k,i}(t)$: a binary variable indicating whether request k flows from the i^{th} VNF instance at node n to the $i+1^{th}$ VNF at node n' via path p ; (III) $Y_n^{k,i}(t)$: a binary variable indicating whether k passes through the i^{th} VNF instance at node n .

B. Cost Structure

1) *VNF Operation Cost*: This mainly refers to: a) time cost incurred by launching VNFIs, and completing the state migration for stateful VNF. This process usually takes a couple of seconds or minutes. Let η_n^f denote the cost of installing the instance of VNF f at node n . And we use $\rho_n^f(t) = \max\{0, q_n^f(t) - q_n^f(t-1)\}$ to denote the number of new instances of VNF f deployed at 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 of running an instance of VNF f at node n . The total VNF operation cost at time t is calculated by:

$$C_O(t) = \sum_{n \in [N]} \sum_{f \in [F]} (\eta_n^f \rho_n^f(t) + \mu_n^f q_n^f(t)) \quad (2)$$

2) *Link Bandwidth Cost*: Transferring requests between distributed clouds incurs bandwidth cost of links. Let δ_l denote the unit bandwidth cost for forwarding flow through link l . We define an indicator variable $I_{k,l}^i = \sum_{n,n' \in [N]} \sum_{p \in [P_{n,n'}]} X_{p,n,n'}^{k,i}(t) H_{n,n'}^{l,p}$ to represent whether link l is used by request k between VNF f_i and f_{i+1} . The overall bandwidth cost at time t is:

$$C_B(t) = \sum_{k \in [K]} \sum_{f_i \in [J_k]} \sum_{l \in [L]} \delta_l I_{k,l}^i \lambda_k^{f_i} R_k^{f_i}(t) \quad (3)$$

3) *SLA Violation Cost*: If a request fails to complete its required service within the expected completion time, an additional SLA violation penalty is incurred. The service completion time of requests is denoted by $D_k(t)$, which includes traffic routing time and VNF processing time ($D_k(t) = \sum_{f_i \in [J_k]} (\theta_p X_{p,n,n'}^{k,i}(t) + p_{f_i})$). Then we can calculate the time-out value $V_k(t)$ of flow k by $V_k(t) = \max\{0, D_k(t) - M_k(t)\}$.

⁵For a step-by-step annotated example of these models, please refer to the supplemental documentation.

⁶We only consider total-ordered linear SFC in this paper, non-linear SFC can be partitioned into several linear SFCs as described in [36].

TABLE II
KEY NOTATION

N, S	The set of cloud nodes and switch nodes
F, K, L	The set of VNFs, user requests and links
T	# of time slots
$P^{n,n'}$	Path set between node n and n'
φ_f	Processing capacity of each instance of VNF f
p_f	Processing delay of VNF f
ζ_f	State migration delay of VNF f
$\nu_{n,n'}$	Routing rule update delay from node n to n'
$\pi(n), b(l)$	Capacity limitation of node n and link l
η_n^f, μ_n^f	Cost for deploying and running VNF f in node n
$\kappa(l)$	Forwarding delay of the link l
δ_l	Cost for forwarding a unit of a flow through link l
θ_p	Forwarding delay of the path p
$D_k(t)$	Service completion time of request k at t
$H_{l,p}^{n,n'}$	Indicator of whether link l belongs to path p between node n and n'
$I_{k,l}^f$	Indicator of whether link l is used for request k between VNF f_i and f_{i+1}
s_k, d_k	Source/destination of request k
J_k	Required service function chain of request k
$R_k(t)$	Flow rate of request k at t
$R_k^f(t)$	Flow rate arriving at VNF f of request k at t
$M_k(t)$	Expected completion time of request k at t
$V_k(t)$	The value of timeout of request k at t
$\lambda_{k,f}$	Flow rate change ratio of request k on VNF f
τ_k	Timeout penalty of flow k
$q_n^f(t)$	# of instances of VNF f deployed in node n at t
$X_{p,n,n'}^{k,i}(t)$	Indicator of whether request k goes from VNF f_i in node n to f_{i+1} in n' through path p at t
$Y_{n,n'}^{k,i}(t)$	Indicator of whether request k goes from VNF f_i in node n at t
$m_{n,n'}^{k,i}(t)$	Indicator of whether request k changes from VNF f_i in node n to f_i in n' at t

We also introduce τ_k to denote the timeout penalty of request k . The total SLA violation at time t can be calculated as:

$$C_T(t) = \sum_{k \in [K]} \tau_k V_k(t) \quad (4)$$

C. Offline SFC Embedding Cost Minimization Problem

Suppose that 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 (5). Important notation is listed in Table II. Since the problem has multiple objectives, the common method to treat this kind of optimization problem is: weight each objective of the original problem, sum them up and form a new single objective optimization problem [37].

$$\text{minimize} \sum_{t \in [T]} C_O(t) + C_B(t) + C_T(t) \quad (5)$$

subject to:

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

$$\forall t \in [T], f \in [F], n \in [N]$$

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

$$\sum_{k \in [K]} \sum_{i=1}^{h_k-1} \sum_{n,n' \in [N]} \sum_{p \in [P_{n,n'}]} X_{p,n,n'}^{k,i}(t) H_{l,p}^{n,n'} \lambda_k^{f_i} R_k^{f_i}(t) \leq b(l), \forall t \in [T], l \in [L] \quad (5c)$$

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

$$i \in [1, h_k - 1]$$

$$\sum_{n' \in [N]} \sum_{p \in [P_{n',n}]} X_{p,n',n}^{k,i-1}(t) = \sum_{n' \in [N]} \sum_{p \in [P_{n,n'}]} X_{p,n,n'}^{k,i}(t), \quad (5e)$$

$$\forall t \in [T], k \in [K], i \in [1, h_k - 1], n \in [N]$$

Constraint (5a) guarantees that the total incoming flow rate to all instances of VNF f at node n does not exceed the processing capacity of the deployed instances. ψ_k^f denotes the position index of VNF f in the SFC required by k , if f is not contained in the SFC, $X_{p,n,n'}^{k,\psi_k^f}(t)$ is defined as zero. Constraint (5b) restricts the total deployed VNF instances at edge cloud does not exceed the capacity limitation. Constraint (5c) enforces that the total flow traversing link l remains below its available bandwidth $b(l)$. Constraints (5d) and (5e) guarantee that for each request k requiring VNF f_i, f_{i+1} (deployed in node n and n' respectively) in the SFC, exactly one path $p \in P_{n,n'}$ is selected for routing.

The optimization problem is NP-hard since it can be reduced from minimum knapsack problem (MKP) [38]. Due to space constraints, the detailed proof process is provided in the supplemental document. Considering that directly solving the MILP model using commercial solvers (e.g., Gurobi [39]) is computationally expensive for large-scale instances, we propose a novel algorithm which can make SFC embedding decisions efficiently and minimize the cost defined in (5).

D. Online SFC Embedding Cost Minimization Problem

In an online manner, accurately predicting future SFC request arrivals is challenging. This requires determining minimal cost SFC embedding schemes for each arriving request based on the parameters of coming requests and embedding status of previous requests. The online SFC embedding cost minimization problem is shown in (6). It should be noted that this is only a subproblem for a new request arriving at time t . During the entire time period $[T]$, the request set $[K_t]$ at time t will change with the arrival and departure of the requests. When optimizing the embedding scheme for a newly arriving request k at time t , it is necessary to ensure that the SFC embedding schemes of the previous requests which are still alive at time t remain unchanged. To enforce this requirement, we introduce Constraint (6a), where $[K_t']$ denotes the set of active requests at time t .

$$\text{minimize} C_O(t) + C_B(t) + C_T(t) \quad (6)$$

subject to: (5a) to (5e)

$$X_{p,n,n'}^{k,i}(t) = X_{p,n,n'}^{k,i}(t-1), \forall t \in [T], n, n' \in [N], \quad (6a)$$

$$p \in [P_{n,n'}], k \in [K_t'], i \in [1, h_k - 1]$$

E. The Request Rerouting Optimization Problem

Throughout the online algorithm operation period $[T]$, it is beneficial to periodically re-optimize the routing schemes for active requests to further reduce the overall routing costs. When executing request rerouting at time $t \in [T]$, the VNF placement remains immutable; only the routing paths between pre-deployed VNFs are reconfigured. The objective of the request rerouting optimization problem is minimizing SLA violation and bandwidth costs (collectively referred to as traffic routing costs) for all existing requests at t . While the cost definitions are consistent with (5), we must account for the additional impact of rerouting latency on SLA violations. The request rerouting delay mainly includes the sum of state migration delay ζ_f of all migrated VNFs and the sum of routing rule updating time $\nu_{n,n'}$ of related cloud nodes. And the service completion time of requests is changed to $D_k(t) = \sum_{f_i \in [J_k]} (\theta_p X_{p,n,n'}^{k,i}(t) + p_{f_i}) + \max_{f_i \in [J_k]} (\zeta_{f_i} + \nu_{n,n'}) m_{n,n'}^{k,f_i}$. $m_{n,n'}^{k,f_i}$ denotes whether request k changes from the VNF f_i instance on node n to the VNF f_i instance on node n' . Finally, the request rerouting optimization problem is shown as (7).

$$\text{minimize } C_B(t) + C_T(t) \quad (7)$$

subject to: (5a), (5c), (5d)

$$Y_n^{k,i}(t) \cdot Y_n^{k,i+1}(t) = \sum_{p \in [P_{n,n'}]} X_{p,n,n'}^{k,i}(t), \forall t \in [T], \quad (7a)$$

$$\begin{aligned} k \in [K], i \in [1, h_k - 1], n, n' \in [N] \\ \sum_{n \in [N]} Y_n^{k,i}(t) = 1, \forall t \in [T], k \in [K], i \in [1, h_k - 1] \end{aligned} \quad (7b)$$

Introducing request rerouting delay when calculating SLA violation cost is not only based on actual considerations (additional time cost will be generated when performing request rerouting), but also to avoid network turbulence caused by excessive traffic rerouting (which means frequent network updates). As a result, the optimization objective of this problem will make a balance between minimizing the routing cost and avoiding network turbulence (i.e., minimizing rerouting overhead). For constraint (5a) in this optimization problem, $q_n^f(t)$ is a pre-determined parameter when executing requests rerouting at time t , derived from the solution of the online SFC embedding problem. Constraints (5d), (7a) and (7b) collectively ensure that for each request k requiring VNF f_i, f_{i+1} (deployed in node n and n' respectively) in the SFC, only one path between node n and n' can be selected.

V. ALGORITHM DESIGN

A. Overview

Since the optimization objective of our problem is minimizing the total SFC embedding cost for all requests in a time period $[T]$, we propose a novel delay-aware SFC embedding and scheduling framework called SFCES which integrates cost-efficient SFC embedding scheme provisioning for incoming requests with periodic re-optimization of routing schemes for all active requests

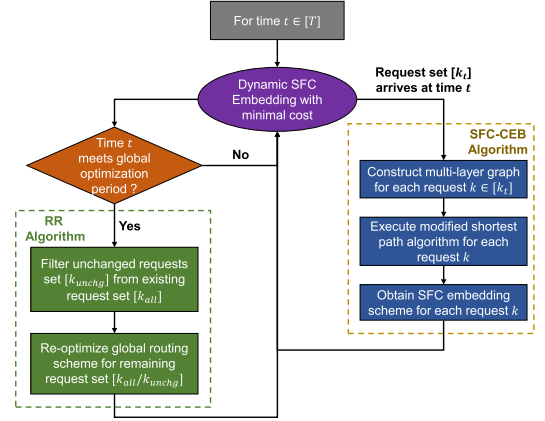


Fig. 2. SFCES Framework Overview.

from a global perspective. An overview of SFCES framework is shown in Fig. 2. The two pivotal components of this framework are:

1) *SFC-CEB Algorithm*: This algorithm is designed to jointly optimize the VNF operational, SLA violation and bandwidth costs for each incoming request. It constructs a multi-layered graph, where edge weights are dynamically determined by the associated operational and routing costs. Then it executes a modified shortest path algorithm to derive a “path” for the SFC requests. Based on 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.

2) *RR Algorithm*: This algorithm periodically re-optimizes the traffic routing cost for active requests. It first prunes requests that do not necessitate re-optimization. To avoid unnecessary computational overhead, a threshold-based triggering mechanism is employed: global re-optimization is only executed if the number of candidate requests exceeds a predefined threshold. Otherwise, it utilizes a linear programming solver (e.g., Gurobi) to solve the request migration optimization problem defined in (7) for remaining requests. Based on the solver’s output, the routing paths for the targeted requests are reconfigured.

At each time step $t \in [T]$, upon the arrival of new requests, the SFCES framework operates as follows:

- 1) For each incoming request k , construct a multi-layer graph based on its required SFC and the associated cost parameters.
- 2) Execute the modified shortest path algorithm to identify the “path” with the minimum embedding cost for request k .
- 3) Map the identified “path” to a real SFC embedding scheme and generate the corresponding deployment configuration.
- 4) If the current time t coincides with the global routing re-optimization period, identify the set of invariant requests $[k_{unchg}]$ from the total active request set $[k_{all}]$. Then determine whether to trigger the request rerouting process.
- 5) If the re-optimization is triggered, solve the request rerouting optimization problem to obtain updated routing schemes for the residual requests.
- 6) Adjust the network routing policy according to the results obtained in Step (5).

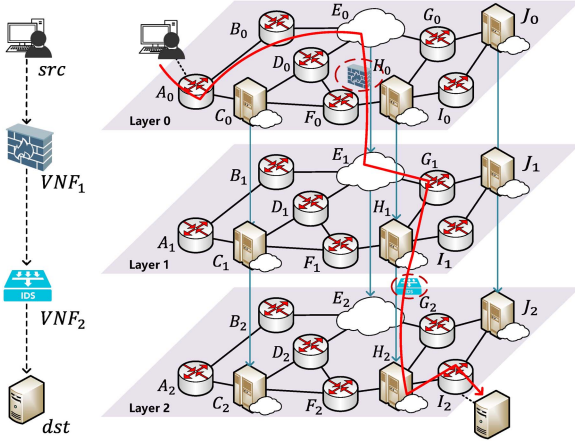


Fig. 3. Constructing multi-layer graph for an SFC request.

B. SFC Cost Efficient Embedding Algorithm

For each arriving request, the SFC-CEB algorithm minimizes its SFC embedding cost in an online manner. Specifically, the algorithm first constructs a multi-layer graph, and subsequently executes a modified shortest path algorithm to derive a “path” for the requests. According to the characteristics of the shortest path algorithm and the setting of path weights, the identified “path” will be a feasible SFC embedding scheme with minimal embedding cost. The details of the SFC-CEB algorithm are shown in Algorithm 1.

1) *Multi-Layer Graph Construction*: The layered network graph contains $h + 1$ ($h = \text{length}(J_k)$) layers, and each layer is a copy of the original network graph. v_i is introduced to denote the corresponding node of the vertex v in the original graph in the i^{th} layer. Any two adjacent layers can only be connected via a 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 the horizontal links in each layer, the bandwidth cost δ_l and the forward delay $\kappa(l)$ are considered. For any request $k \in [k]$ entering the multi-layer graph from the ingress node in the 1^{st} layer and leaving from the egress node in the $(h + 1)^{\text{th}}$ layer, k needs to pass through all h layers to obtain the services defined in J_k . Fig. 3 illustrates the construction of a multi-layer graph and an optimal embedding solution for SFC which contains two VNFs: VNF_1 and VNF_2 . This solution selects nodes E and H to deploy VNF instances, and the routing path in multi-layer graph is the red line. Therefore, the final deployment and routing scheme in the original network topology is: $A \rightarrow B \rightarrow E(VNF_1) \rightarrow G \rightarrow H(VNF_2) \rightarrow I$.

2) *Modified Shortest Path Algorithm*: The SLA violation cost is a piecewise function, which poses a challenge for traditional shortest path algorithms that require additive edge weights. To address this, we define a quintuple weight vector: $[dr_c, sla_c, bw_c, res_w, remn_t]$. In this quintuple, (I) dr_c is the cumulative VNF operation cost, which is updated when a vertical edge between layers is traversed; (II) sla_c is SLA violation cost, calculated via (4), where $V_k(t) = \max(0, remn_t)$; (III) bw_c denotes the bandwidth cost, which accumulates

Algorithm 1: SFC-CEB Algorithm.

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

```

1 For request  $k$  arrives at time  $t$ 
2  $G' \leftarrow \text{LayeredGraph}(G, J_k, \eta_n^f, \mu_n^f)$ 
3 for each  $v \in N, l \in L$  do
4    $\alpha_n, \alpha_l \leftarrow \text{CalWeight}(v, l)$ 
5 end
6 Initialize three lists  $path$ ,  $dis$  and  $vis$ 
7 Initialize a heap  $pq = \emptyset$  and push  $src_k$  into  $pq$ 
8 while  $pq \neq \emptyset$  do
9    $v \leftarrow pq.pop()$ 
10  if  $vis[v]$  then continue;
11   $vis[v] \leftarrow True$ 
12  for  $u \in G'(v)$  do
13    if  $\text{CheckPath}(path[v], \pi(n), b(l))$  then
14       $\text{DFSNode}(u, path, G', \pi(n), b(l), \Phi)$ 
15       $d \leftarrow \text{CalDis}(v, u, \alpha_n, \alpha_l, M_k(t))$ 
16      if  $d < dis[u]$  then
17        Update  $(u, dis[u], path[u], d)$ 
18        Push the node  $u$  into heap  $pq$ 
19      end
20    end
21  end
22 end
23  $sh_k \leftarrow path[dst_k]$ 

```

according to (3) when a horizontal edge within any layer is traversed. (IV) res_w is the cumulative resource weight (for node or link), designed to mitigate resource exhaustion by early-arriving requests; it is updated whenever resources are occupied. (V) $remn_t$ represents the relative latency against the deadline, initialized as $-M_k(t)$. This value accumulates the forwarding delay of links or processing delay of VNF instances. By employing this quintuple, the modified shortest path algorithm can evaluate and compare path lengths, where the total length is defined as the sum of dr_c , sla_c , bw_c and res_w .

Specifically, the exploration process (detailed in lines 8-22) is built upon the standard Dijkstra algorithm. It iteratively explores each node v and its neighbor nodes $u \in G'(v)$ to update the dis (storing path lengths) and $path$ (recording the shortest path from source node to each node in layered graph). The differences between our modified shortest path algorithm and Dijkstra algorithm are: we should check whether the 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 and update dis and $path$. This process is recursive, and the number of recursions is limited by Φ (Max DFS depth). When all nodes in the 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. Further details regarding the underlying logic of the SFC-CEB algorithm are provided in our previous work [9].

In practice, the SFC-CEB algorithm can be implemented as a module within the network control plane. To facilitate real-world deployment, it can be integrated with existing

Algorithm 2: RR Algorithm.

Input : Related parameters used in Eq. 7
Input : Previous routing schemes: $[p_r_sh_k]$, Θ , Δ
Output: Final rerouting scheme: $[f_r_sh_k]$

- 1 Solve linear relaxed optimization problem defined in Eq. 7 for all requests in $[k_{all}]$ to obtain new routing schemes r_sh_k .
- 2 **for** each k in $[k_{all}]$ **do**
- 3 **if** $r_sh_k = p_r_sh_k$ **then** Classify k into $[k_{unchg}]$;
- 4 **else** Classify k into $[k_{chg}]$;
- 5 **end**
- 6 **for** each k in $[k_{unchg}]$ **do**
- 7 Calculate the delay gap between expected completion time and actual E2E delay for k .
- 8 Sort the above calculation results in descending order, and extract the first Θ requests from $[k_{unchg}]$ to $[k_{chg}]$.
- 9 **end**
- 10 **if** $\frac{|k_{chg}|}{|k_{all}|} < \Delta$ **then** return;
- 11 Solve original problem defined in Eq. 7 for requests in $[k_{chg}]$ to obtain new routing schemes $chg_r_sh_k$.
- 12 Update r_sh_k with $chg_r_sh_k$ to obtain final rerouting scheme $[f_r_sh_k]$.

network measurement tools (e.g., NetFlow [40]) to gather traffic statistics, and leveraged alongside cloud orchestration platforms (e.g., Kubernetes [41]) or NFV deployment framework (e.g., OpenNetVM [42]) to instantiate the required SFCs. Section VI-A2, details the implementation of SFC-CEB in a real-world environment. Notably, the algorithm prioritizes long-lived requests (whose lifetimes are longer than the VNFI deployment time). Conversely, short-lived requests are managed through pre-defined provisioning policies, such as utilizing preset embedding schemes to avoid deployment latency. 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 the supplemental document.

C. Request Rerouting Algorithm

At the global routing re-optimization period t , the Request Rerouting (RR) algorithm re-allocates routing schemes for existing requests without altering current VNF deployments to avoid redeployment costs. As common VNFs are frequently shared across SFCs and multiple instances of VNF are often required to meet demand, routing adjustments can effectively optimize routing costs based on the existing VNF deployment. The RR algorithm framework is shown in Algorithm 2.

To enhance computational efficiency, the RR algorithm first prunes the set of candidate requests before executing the full routing re-optimization. This is motivated by the observation that a significant portion of active requests may already possess near-optimal embedding configurations. To identify these, the algorithm solves a linear relaxation of the request rerouting problem using the Gurobi solver (Line 1). Specifically, for any request yielding an integer solution that matches its current configuration (as determined by SFC-CEB), it is tentatively

assigned to the invariant set $[k_{unchg}]$, indicating that no immediate rerouting is required. Otherwise, requests are categorized into the candidate set $[k_{chg}]$ (Lines 2-5). After preliminary filtering, the algorithm will continue to calculate the delay gap between the expected completion time and the actual end-to-end delay for each request from $[k_{unchg}]$ (line 7), and pick out Θ requests with largest delay gap (line 8). Although this type of request is considered to not need rerouting in preliminary filtering, it has higher routing optimization potential compared to other requesters in $[k_{unchg}]$. These selected requests together with the remaining requests after preliminary filtering constitute the request set $[k_{chg}]$ to be optimized.

If the candidate set $[k_{chg}]$ constitutes only a minor fraction of the total active requests $[k_{all}]$ (i.e., the ratio falls below a threshold Δ), the re-optimization process is bypassed to mitigate unnecessary control plane overhead (line 10). Otherwise, the RR algorithm invokes the Gurobi solver to solve the request rerouting problem for $[k_{chg}]$, as defined in (7) (Line 11). To ensure the solver converges within the pre-defined optimization cycle T_{opt} , we can set the maximum solution time of the solver.⁷ Once the solver returns a solution, the algorithm can determine the new routing schemes $chg_r_sh_k$ for requests in $[k_{chg}]$ based on the value of $Y_n^{k,i}(t)$ and $X_{p,n,n'}^{k,i}(t)$, and generate the final request routing schemes $[f_r_sh_k]$ (line 11-12). It should be noted that the values of parameters Θ and Δ in the RR algorithm can be set by the algorithm executor according to the requirements for optimization effects.

D. Complexity Analysis

Given a network topology $G = (N, L)$, the complexity of the standard heap-optimized Dijkstra algorithm is $O([L]\log[N])$. Since SFC-CEB incorporates additional path feasibility checks (complexity: $O(h[N])$) and a depth-limited exploration (complexity: $O((h[L])^\Phi)$), the total complexity of SFC-CEB is $O(((h[L])^\Phi + h^2[L][N])\log(h[N]))$. As for the RR algorithm, the computational complexity is primarily governed by the network scale and the number of requests targeted for optimization. By pruning requests that do not necessitate rerouting, we significantly reduce the problem size. Furthermore, operators can balance the efficiency and accuracy of the solving processing, and then determine the proper solving time of the solver (e.g., Gurobi) for optimization problem defined in (7).

VI. PERFORMANCE EVALUATION

We implemented the SFCES framework in Python and evaluated its performance across both simulation and real-world testbed environment. We compare SFCES with the state-of-the-art works and use VNF operation cost, bandwidth cost, SLA violation cost and end-to-end delay as metrics. The experimental results demonstrate the effectiveness of SFCES.

A. Experiment Setup

1) *Simulation Environment Setup:* 1) *Topology Dataset:* We utilized the Abilene (11 nodes, 14 links) and Uninet (49 nodes,

⁷This functionality is natively supported by the Gurobi optimizer.

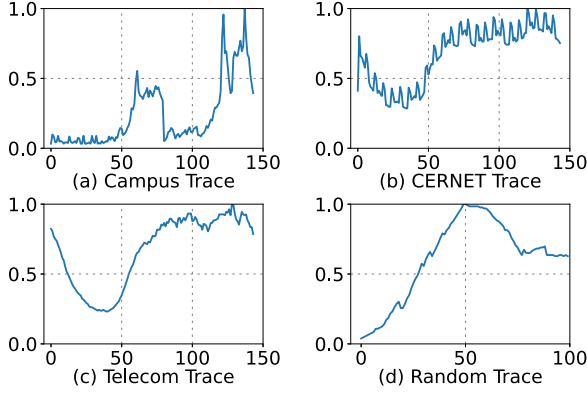


Fig. 4. Four Type SFC Requests Trace.

84 links) topologies from TopologyZoo [43]. For each topology, several nodes were randomly selected as edge clouds, while a single node was designated as the public cloud. The capacities of the edge clouds were uniformly sampled from [20, 40] units (one *unit* refers to holding one VNFI), whereas the public cloud was assumed to have infinite capacity. Link propagation delays were randomly assigned between [3, 70] ms, and bandwidth capacities were set within [10, 20] Gb/s. The network topologies and their associated graph structures were modeled and managed using the NetworkX [44] Python library.

2) *Traffic Dataset*: We utilized the following SFC request traces: (a) real campus network traffic (Campus Trace), (b) CERNET traffic (CERNET Trace), (c) communication operator traffic (Telecom Trace) and (d) random generation (Random Trace). The real-world traces consist of 24-hour NetFlow data collected from multiple nodes within their respective networks. Traffic records sharing the same source-destination IP pair were aggregated into individual SFC requests, and the arrival volume was counted per time unit (10 minutes). The normalized request arrival patterns are shown in Fig. 4, with the x-axis representing time units and the y-axis the normalized number of requests. For simulations, the normalized request numbers were scaled up to 200 times to represent actual arrivals. Ingress/egress nodes and the required VNFs were randomly assigned for each request. Flow rates and expected completion times were randomly sampled from [1, 10] Mb/s and [50, 800] ms, respectively. The flow size change ratio λ_k^f was selected from [0.8, 1.2], and each request was assigned a randomly generated lifetime.

3) *VNF and Cost Data*: We considered 10 distinct VNF types, with processing capacities φ_f ranging from 5 to 10 units (where one unit represents a throughput of 10 Mb/s). Each SFC contains 2 ~ 5 VNFs. In practice, VNF operational costs can be derived from regional energy tariffs and instance instantiation times, while bandwidth and SLA violation costs are stipulated by service providers. To simplify comparisons but without loss of generality, we randomly generated the VNF operation cost parameters ($\eta_n^f \in [1, 6]$ and $\mu_n^f \in [1, 3]$), link bandwidth cost ($\delta_l \in [0.1, 0.8]$) and timeout penalty factor ($\tau_k \in [0.002, 0.01]$) for different requests to simulate the real situation. To facilitate direct summation for the total embedding cost, these cost parameters were normalized to a consistent magnitude.

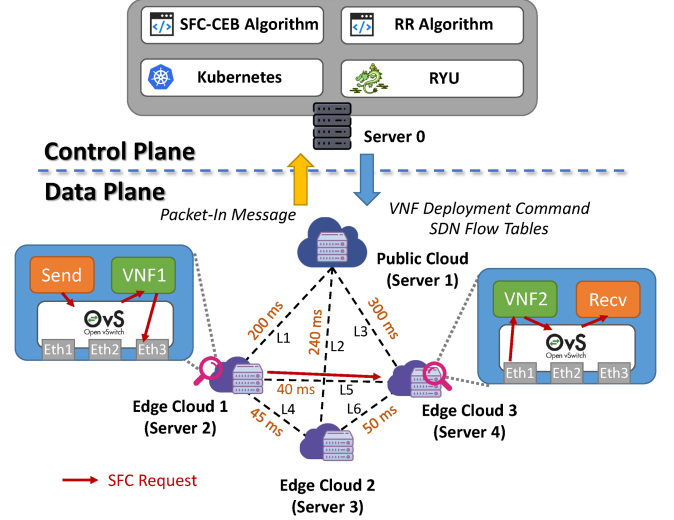


Fig. 5. Testbed Topology Setup.

2) *Testbed Environment Setup*: 1) *Topology Setup*: Our testbed consists of four physical servers (DELL EMC R740) as cloud nodes: one designated as the public cloud and the remaining three as edge clouds. All nodes are orchestrated by a controller (Server 0), which executes SFC embedding algorithms, and issues VNF deployment commands and traffic routing policy. Kubernetes supports VNF deployment, with the controller acting as the master node and the four cloud servers as slave nodes. Traffic routing is implemented via flow tables in a Software-Defined Networking (SDN) environment, using RYU [45] as the SDN controller. Each cloud server runs Open vSwitch, managed by RYU, which also runs on the controller. The testbed topology is shown in Fig. 5, where each edge cloud server accommodates 4 to 10 VNF instances. Link forwarding delays (configured via Linux traffic control) are also indicated in Fig. 5.

2) *Traffic Generation*: To simulate realistic traffic, we instantiated Docker containers on the edge cloud servers to serve as request senders and receivers. Each container runs an Ubuntu OS and utilizes qperf⁸ for message transmission. By controlling the message payload length, we can accurately measure end-to-end delay. The payload lengths were randomly picked in [100 Kb, 500 Kb and 1Mb], and the expected completion times were also randomly generated in [50, 800] ms. We initiated a total of 60 qperf sender-receiver pairs (60 requests) that follow the Poisson distribution to arrive, and we assume that the requests will always exist after it enters the network.

3) *VNF and Request Routing*: We implemented three types of VNF: Network Address Translation (NAT), Firewall and Intrusion Detection System (IDS). Each VNF is instantiated as a Docker container running on Ubuntu-based servers. Specifically, NAT and Firewall are provisioned via iptables,⁹ while IDS is implemented using Suricata.¹⁰ We assume that each docker can

⁸qperf: <https://linux.die.net/man/1/qperf>

⁹iptables, <https://www.netfilter.org/>

¹⁰Suricata, <https://suricata-ids.org/>

process 2 ~ 5 SFC requests. Each SFC contains 1 ~ 3 VNFs. For request routing, all Docker containers are connected to an Open vSwitch [46] bridge, with flow tables issued by RYU controlling request forwarding. Since a single VNF container may serve multiple requests, internal routing tables are configured within the containers to prevent routing conflicts. A representative routing path for an SFC request from edge cloud 1 to edge cloud 3 is shown in the Fig. 5 (dark red line).

B. Compared Algorithm

1) **RDIP** [6] is a state-of-the-art algorithm designed to jointly optimize 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 VNFs provisioning. Finally, it modifies the traffic routing decision and yields a feasible solution. Additionally, RDIP only optimizes embedding schemes for newly arrived requests, without considering adjustments to the embedding schemes of previously handled requests.

2) **RDIP-mig** is an adapted version of RDIP with the same optimization objectives. Unlike RDIP, it enables periodic adjustments to SFC embedding schemes for active requests, including VNF deployment and traffic routing (similar to SFCES). RDIP-mig also considers existing VNF instances on cloud nodes to minimize additional operational costs.

3) **ABSA** [22] utilizes a heuristic algorithm based on Affinity-Based Simulated Annealing to optimize SFC embedding. The algorithm tends to deploy adjacent VNFs within an SFC onto the same cloud node whenever possible, aiming to reduce cross-node traffic transmission. Once the placement of VNF instances is determined, the algorithm subsequently employs a shortest path algorithm to determine the routing paths between VNF instances. For this algorithm, we also enable periodic adjustments to the SFC embedding schemes for active requests.

4) **BCHA** [23] is also a state-of-the-art algorithm. In [23], both a Bee Colony-based heuristic algorithm and a Genetic Algorithm-based heuristic algorithm are proposed to perform multi-objective optimization of VNF deployment costs, SFC service latency, and link bandwidth costs. These algorithms iteratively evolve an initial feasible solution to converge toward a near-optimal solution. Based on the performance benchmarks provided in [23], we selected the Bee Colony-based algorithm (hereafter referred to as BCHA) as our baseline due to its superior efficiency. To ensure a rigorous and fair comparison, we synchronized the re-optimization intervals of RDIP-mig, ABSA, and BCHA with that of our proposed SFCES framework. These algorithms mainly demonstrate that, even with periodic re-optimization of the embedding schemes, they still struggle to effectively address the high SLA violation costs caused by the URP problem.

5) **SFC-CEB** [9] is our previous work. It aims to minimize the VNF operation cost, SLA violation cost and bandwidth cost jointly for each arriving SFC request. This optimization objective is realized by using multi-layer graph and modified shortest path algorithm. However, this work does not consider

adjusting the routing schemes for existing requests. It is used to prove the effectiveness of routing re-optimization.

6) **Offline Solving MILP** is using the MILP solver Gurobi to solve the offline optimization problem defined in (5) and obtaining the offline optimal solution. It is used to evaluate the gap between SFC-CEB and the theoretical optimal solution.

C. Simulation Environment Experiment Results

1) *Evaluation of Cost Types*: We evaluated the performance of the aforementioned solutions with SFCES using the Unnet topology. The evaluation metrics, as defined in (5), include VNF operational, SLA violation, bandwidth, and total embedding costs. For each time unit, we recorded the costs incurred to serve incoming requests following the arrival patterns. As illustrated in Fig. 6(a)–(d), SFCES achieves the lowest total embedding cost across nearly the entire duration of different traces. In particular, during periods of high request volume (e.g., in the campus network trace at time units 60~80 and 120~135), SFCES demonstrates superior cost optimization. Statistical results¹¹ show that SFCES can reduce the total embedding cost by up to 46% ~ 64% (average reduction of 36% ~ 47%) compared to RDIP. Against RDIP-mig, SFCES reduces the cost by up to 40% ~ 52% (average 27% ~ 40%), and against ABSA by up to 43% ~ 55% (average 31% ~ 42%). Compared to BCHA, SFCES reduces costs by up to 45% ~ 51% (average 30% ~ 40%), and against SFC-CEB by up to 20% ~ 24% (average 12% ~ 16%). Furthermore, the Cumulative Distribution Function (CDF) of the total costs, depicted in Fig. 5(e)–(h), confirms that SFCES consistently achieves a lower total embedding cost with a higher probability.

Next, we decomposed the optimization effects into specific cost components to elucidate why SFCES consistently yields the superior total embedding cost. As shown in Fig. 7(a)–(d), the RDIP algorithm demonstrates better optimization under heavy request volumes. For SFC-CEB and SFCES, their strategy involves reserving some low-cost resources for future requests to avoid resource exhaustion, which may sometimes lead to higher VNF operation costs. In contrast, RDIP does not apply such a strategy; instead, it continuously seeks to jointly minimize VNF operation cost, end-to-end delay, and bandwidth cost. Consequently, in certain scenarios, RDIP achieves lower VNF operation costs compared to SFC-CEB and SFCES. For RDIP-mig, ABSA, and BCHA, their higher VNF operation costs result from their periodic optimization of the overall SFC embedding schemes. Adjusting these schemes may require launching new VNF instances to achieve a lower total embedding cost, effectively sacrificing some VNF operation cost to gain improvements in end-to-end latency and bandwidth cost.

As shown in Fig. 7(e)–(h), RDIP, RDIP-mig, and BCHA allocate cloud resources with low routing costs to any incoming request, which leads to high SLA violation costs when these resources are exhausted, as subsequent requests must use cloud resources with higher routing cost. Although RDIP-mig and

¹¹The supplemental document provides details on SFCES cost optimization compared to other solutions

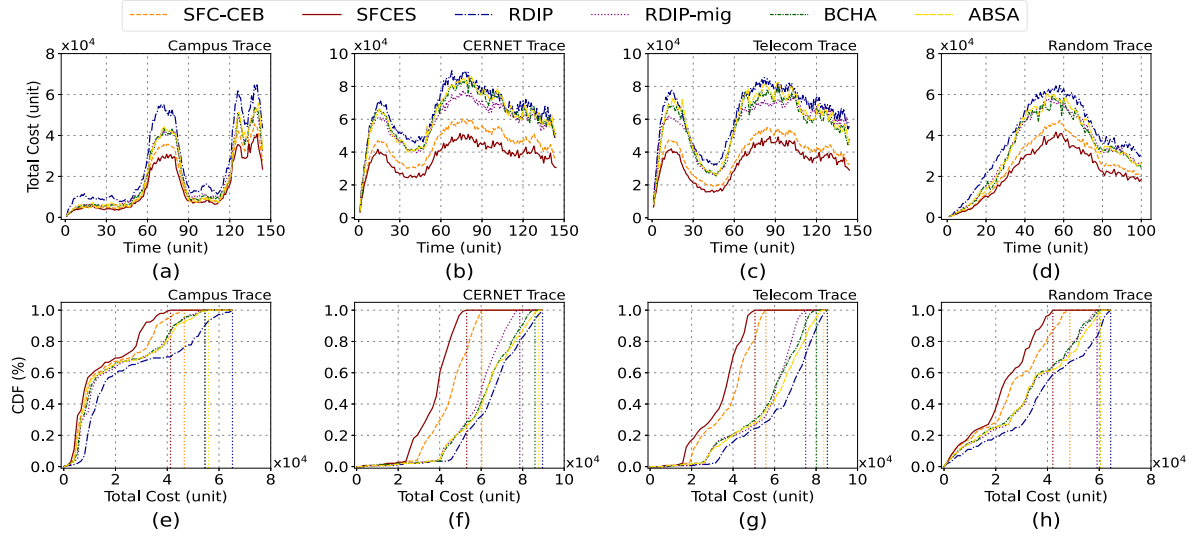


Fig. 6. The Comparison of Total Embedding Cost and the Corresponding CDF.

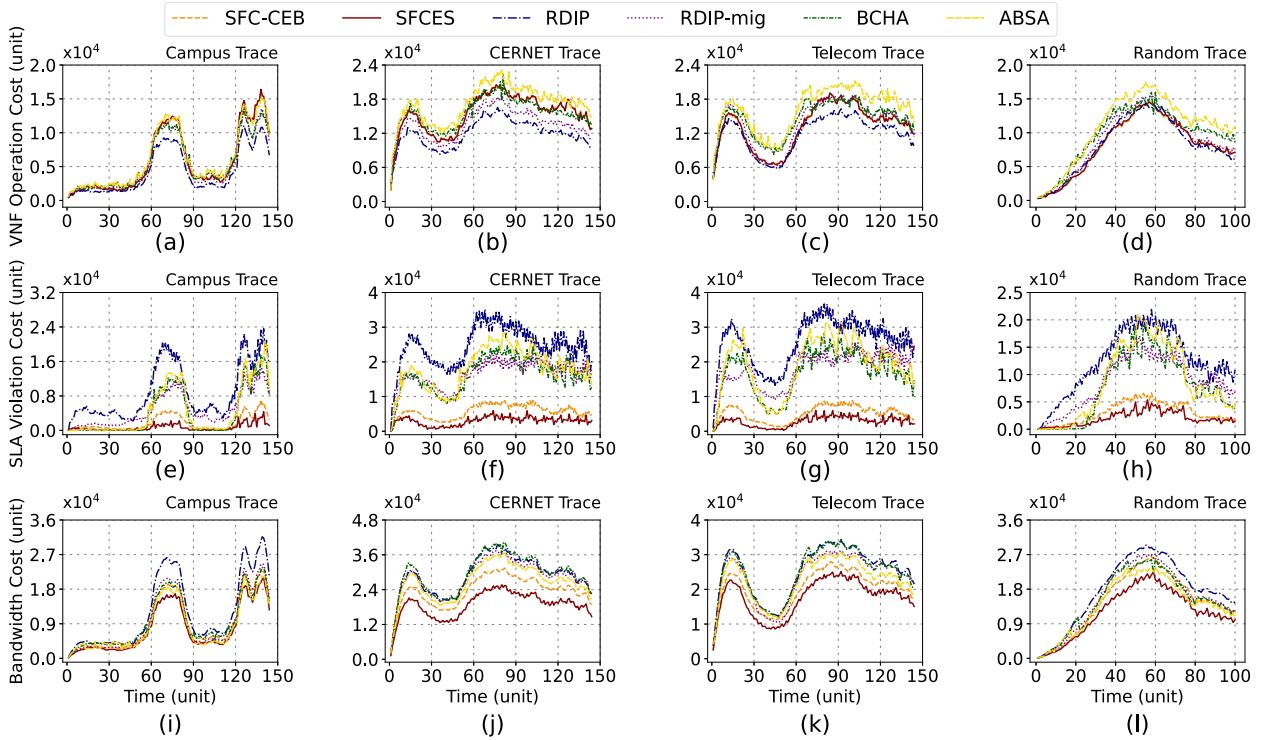


Fig. 7. The Comparison of Different Type Cost.

BCHA periodically re-optimize their SFC embedding schemes, they do not account for request delay tolerance, limiting their effectiveness in reducing SLA violation costs. In contrast, SFC-CEB and SFCES allocate resources based on request delay tolerance, maintaining lower SLA violation costs even under high request volumes. Additionally, SFCES periodically re-optimizes existing routing schemes, achieving lower SLA violation costs than SFC-CEB. ABSA, without specific SLA violation cost or end-to-end delay optimization, performs only slightly better than RDIP due to its periodic adjustments, which allow it to

reconfigure deployments for requests with higher SLA violation costs.

For bandwidth cost optimization performance shown in Fig. 7(i)–(l), SFC-CEB and SFCES achieve superior results by prioritizing SLA violation and bandwidth costs within their joint optimization framework (as defined in (5)). In contrast, RDIP, RDIP-mig, and BCHA pursue slightly different joint optimization objectives (substituting SLA violation cost with end-to-end delay cost), which may lead them to prioritize VNF operation cost savings at the expense of optimizing end-to-end delay

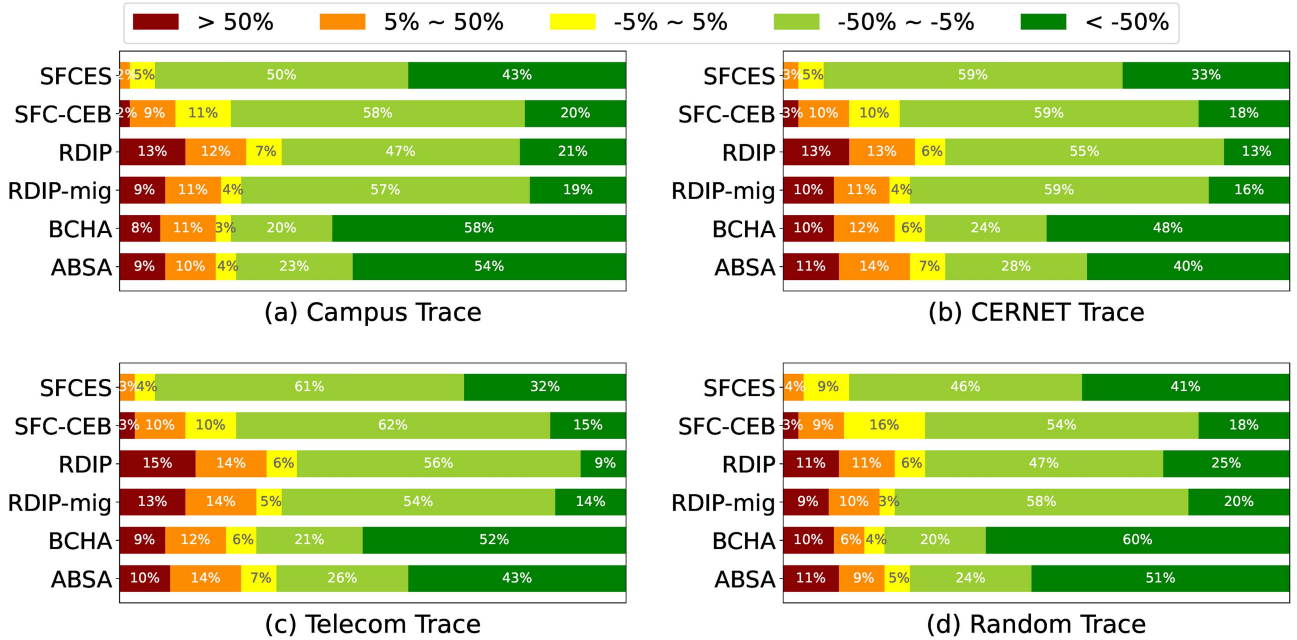


Fig. 8. The Comparison of Percentage of Delay Gap to Expected Completion Time.

and bandwidth costs. SFCES further outperforms SFC-CEB by periodically re-optimizing routes, which also benefits RDIP-mig and BCHA compared to RDIP. ABSA, by co-locating adjacent VNFs of SFC within the same cloud node, reduces inter-node traffic, achieving better bandwidth cost than RDIP-mig and BCHA. However, since ABSA does not specifically target cost optimization, its bandwidth cost performance is still inferior to that of SFCES (though it may surpass SFC-CEB, as seen in the campus trace results in Fig. 7(i), with a gap of about 5%). In terms of the above results, SFCES effectively balances VNF operation and traffic routing costs to minimize total embedding cost, with the routing re-optimization enhancing this effect.

We further analyzed the delay gap between the actual end-to-end delay and the expected completion time for each request across different solutions, as shown in Fig. 8. The delay gap percentage (actual delay minus expected time, relative to expected time) is divided into five levels: $[> 50\%, 5\% \sim 50\%, -5\% \sim 5\%, -50\% \sim -5\%, < -50\%]$. A higher gap indicates poorer service quality, while a lower or negative gap reflects better performance. The results show that RDIP has the most positive delay gaps, indicating poorer service performance. RDIP-mig, BCHA, and ABSA perform slightly better but remain inferior to SFC-CEB and SFCES. SFC-CEB and SFCES achieve fewer positive gaps, with SFCES performing best due to request rerouting. A scatter plot of delays and expected times is included in the supplemental document. These results demonstrate that SFCES prioritizes delay-sensitive requests by reallocating resources from delay-tolerant ones, ensuring better service quality.

2) *Cost Gap Compared with Optimal Solution*: To evaluate the cost gap relative to the optimal solution, we benchmarked our algorithm against the offline optimal results obtained by solving the MILP formulation (5) directly via Gurobi. Due to the significant computational overhead for large topologies and

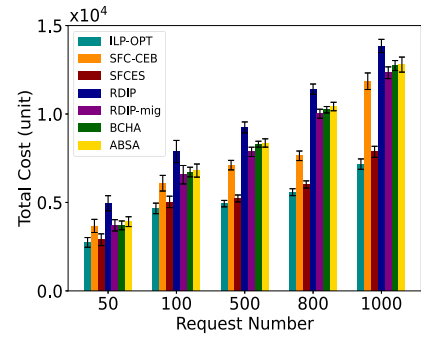


Fig. 9. The Cost Gap Compared with Optimal Solution.

request sets, we limit the topology size and number of requests to reduce solving time. The Abilene topology is used as the experimental setup, with randomly generated request sets of 50, 100, 500, 800, and 1000 SFC requests. Each set is provided to Gurobi for offline calculation to obtain optimal results. Requests are also processed sequentially to simulate online scenarios. As shown in Fig. 9, the total embedding cost achieved by SFCES is consistently lower than other algorithms and averages within 5.1% ~ 7.3% of the optimal solution (ILP-OPT). Given the much higher computational time required for exact solutions, this cost gap is considered acceptable. For larger network scales, directly solving the original optimization problem is impractical for obtaining feasible SFC embedding schemes.

3) *Evaluation of the Number of Rerouting Requests*: Frequent request rerouting will cause routing turbulence in the network and reduce the reliability of the services. And request migration usually causes an additional processing delay that may also affect the SLAs of requests [32]. Hence, it is better to control

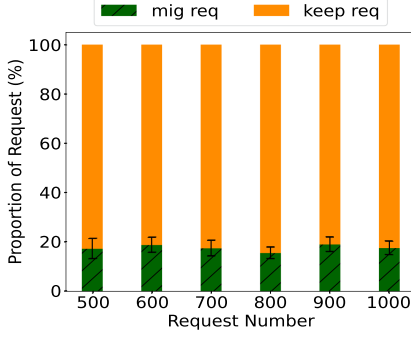


Fig. 10. The Number of migration requests in SFCES.

the number of migration requests when we execute routing re-optimization operation. We incorporated pruning mechanisms within SFCES to minimize the number of requests that need to be rerouted (e.g., filter out the requests for which the rerouting operation cannot bring sufficient benefits). In the simulation, we executed the SFCES algorithm across several request sets $[K]$ (ranging from 500 to 1,000 requests) and quantified the migration ratio. As shown in Fig. 10, the number of migrated requests as a percentage of the total requests is on average less than 20%. This means that there is no large-scale request migration in rerouting operation, and the network can remain in a relatively stable state.

4) *Evaluation of Accuracy and Running Time:* In the SFC-CEB algorithm, the depth of the DFS procedure influences both accuracy and runtime. The results in [9] show that, regarding the search accuracy for individual SFC request embedding, failure to find the optimal embedding scheme for a single request drops below 1% at DFS depths of 1 or 2. In these cases, the runtime ranges between 30 and 80 ms, which is faster than typical request lifetimes. For the RR algorithm, accuracy and runtime represent a manageable trade-off. As solver execution times are strictly capped at 300 seconds, our framework consistently identifies high-quality solutions (with an optimality gap between 0.05% and 0.2% in 70%–80% of cases). Notably, the solver achieves over 95% optimality within 30 seconds. The ratio of solving time to the optimization period represents a flexible trade-off: operators can prioritize near-perfect precision or truncate early to ensure an ample buffer for network adjustments. Detailed analysis of this scalability across varying optimization periods and traffic loads is provided in the supplemental documentation.

D. Testbed Environment Experiment Results

Following the testbed evaluation, the VNF dockers deploying status and link bandwidth usage status of the algorithms is shown in Table III. The results indicate that the number of deployed VNF dockers is similar across all algorithms, which means that there is almost no gap in the optimization of VNF operation cost. Regarding bandwidth usage, SFCES achieves the lowest total bandwidth consumption 61 Mb, reducing bandwidth usage by 13%, 22%, 15%, 17%, and 12% compared to SFC-CEB (70 Mb), RDIP (78 Mb), RDIP-mig (72 Mb), BCHA (73 Mb), and ABSA (69 Mb), respectively. However, the optimization effect

TABLE III
SERVERS AND LINKS USAGE

Alg. ¹	Deployed Docker (#) ²				Bandwidth Usage (Mb) ³					
	S1	S2	S3	S4	L1	L2	L3	L4	L5	L6
(1)	18	5	9	10	11	5	22	5	6	12
(2)	18	5	9	10	12	4	27	6	8	13
(3)	19	5	8	10	6	7	29	5	12	19
(4)	18	5	9	10	9	5	29	4	15	10
(5)	19	6	8	9	9	4	29	9	10	12
(6)	18	6	8	10	10	5	29	6	9	10

¹ Alg. (1) is SFCES, (2) is SFC-CEB, (3) is RDIP, (4) is RDIP-mig, (5) is BCHA, (6) is ABSA

² Number of VNF dockers deployed on the servers

³ Sum of bandwidth occupied by requests passing through the link

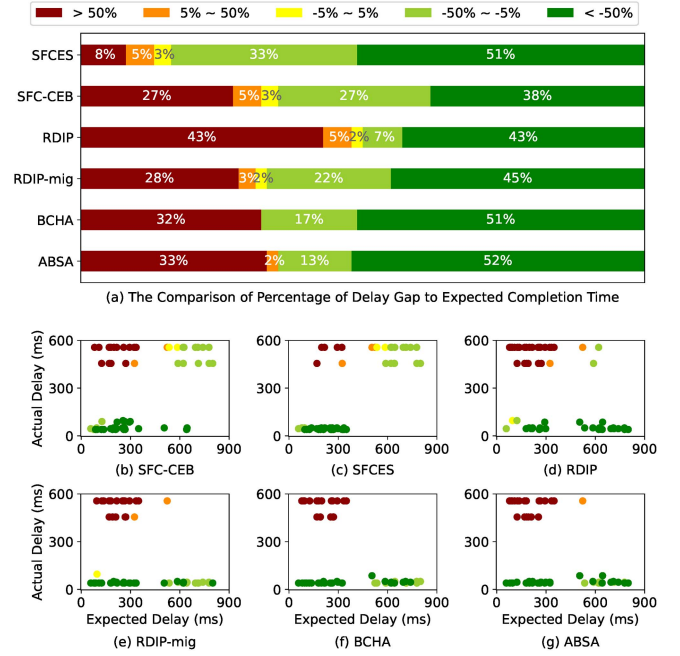


Fig. 11. The Delay optimization results of Testbed.

on bandwidth cost is not significant, mainly due to the simple topology of the testbed, which offers limited routing paths for optimization.

Consequently, we pivoted our analysis to the delay optimization efficacy of these algorithms. As shown in Fig. 11, (a) compares the percentage of delay gap relative to the expected completion time for each solution, while (b)~(g) provide scatter plots of the results from (a). The findings align with the simulation results: SFCES achieves the lowest percentage of high delay gaps (over 50%) and the highest percentage of low delay gaps (under -50%). Notably, for requests with low expected completion times, SFCES reduces end-to-end delay through request rerouting, with dark green dots mainly clustered in the lower left corner of (c). In contrast, other algorithms show more red dots in the upper left corner and some dark green dots in the lower right, indicating that some requests with higher latency tolerance occupy edge cloud resources to achieve lower end-to-end delay. These results demonstrate that SFCES more effectively allocates limited edge cloud resources to requests with varying delay sensitivities, addressing the URP problem.

VII. DISCUSSION

Scalability and High-Concurrency Management: The scalability of the SFCES framework is ensured by the synergistic design of its two stages. For embedding, although the execution time of SFC-CEB scales with the topology size, its processing efficiency can be enhanced through parallel multi-threading (utilizing resource sharding to prevent allocation conflicts), request aggregation, and priority queue. Complementarily, the RR algorithm manages computational overhead by filtering high-impact requests and leveraging the ability of modern solvers to limit solving time and accuracy. Since global re-optimization is a periodic strategic task, network operators can balance real-time decision efficiency with long-term cost-effectiveness across diverse network scales.

Robustness to Spikes and Failures: SFCES can ensure reliability via an SLA-aware prioritization and offloading strategy; during workload spikes, edge resources are reserved for delay-sensitive SFCs while delay-tolerant traffic is queued or offloaded. To address network failures, the framework leverages its online nature to bypass outages for new requests, while existing flows are restored through a two-tier recovery: immediate re-mapping via SFC-CEB and periodic re-optimization via RR. This architecture is designed to integrate with infrastructure monitoring (e.g., Kubernetes heartbeats and SDN-based LLDP) as triggers for automated self-healing. Such a design maintains service continuity and high availability amidst hardware instability and traffic fluctuations.

Hybrid and Multi-cloud Adaptation: SFCES is inherently built for hybrid cloud environments. This architecture is naturally extensible to multi-cloud scenarios by abstracting diverse providers as heterogeneous nodes with distinct cost and latency profiles. By incorporating provider-specific constraints (e.g., inter-cloud egress fees and data sovereignty policies) into the optimization objectives, the framework can maintain its decision logic across complex, multi-vendor ecosystems without requiring fundamental algorithmic changes.

Limitations: SFCES still faces several constraints. First, centralized scalability remains a bottleneck as topologies grow toward carrier-grade dimensions. Second, the heuristic nature of SFC-CEB targets real-time responsiveness, which may deviate from the theoretical global optimum in high-dimensional search spaces. Third, in highly dynamic environments, the mismatch between control-plane decision cycles and rapid data-plane churn can lead to “stale decisions”. Finally, the operational overhead of VNF transitions is simplified; in practice, the latency of “cold starts” and the requirement for state consistency during migrations can impact service continuity.

Future Work: To address the above challenges, we aim to explore distributed orchestration architectures to alleviate centralized bottlenecks and leverage deep reinforcement learning to enhance decision optimality. We also plan to develop machine learning-driven adaptive optimization cycles that dynamically tune the optimization interval and the selection of requests for migration. Furthermore, we will integrate multi-dimensional constraints (e.g., storage and energy) and model performance/security interference among co-located VNFs. We will also conduct extensive benchmarking across diverse topologies to ensure the framework’s robustness.

VIII. CONCLUSION

In this paper, we address the SFC embedding and request migration problems in geo-distributed edge cloud networks. By factoring in SLA violation costs and re-optimizing routing, our approach reduces resource competition across edge clouds, helping to lower service degradation costs and improve network performance. We introduce a delay-aware SFC embedding and scheduling framework, SFCES, which efficiently finds near-optimal embedding schemes for each SFC request while optimizing traffic routing costs for existing requests. Experimental results show that our solution achieves lower total embedding costs compared to other solutions. Our future work will focus on transitioning toward a distributed, AI-driven orchestration framework, while further enhancing the robustness and practical utility of SFCES through extensive evaluation on diverse, large-scale topologies.

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. Surv. Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 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. Surv. Tuts.*, vol. 19, no. 3, pp. 1657–1681, Thirdquarter 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. 15th Annu. IEEE Int. Conf. Sensing, Commun., Netw.*, 2018, pp. 1–9.
- [4] M. Chiosi et al., “Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action,” white paper, SDN and OpenFlow World Congress, Darmstadt, Germany, Oct. 22–24, 2012. [Online] Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf
- [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. Mag.*, vol. 55, no. 2, pp. 216–223, Feb. 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.*, vol. 10, no. 1, pp. 648–661, Jan.–Mar. 2022.
- [8] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Thirdquarter 2017.
- [9] W. Chen, Z. Wang, H. Zhang, X. Yin, and X. Shi, “Cost-efficient dynamic service function chain embedding in edge clouds,” in *Proc. 17th Int. Conf. Netw. Serv. Manage.*, 2021, pp. 310–318.
- [10] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, “Resource scheduling in edge computing: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2131–2165, Fourthquarter 2021.
- [11] A. C. Baktir, A. Ozgovde, and C. Ersoy, “How can edge computing benefit from software-defined networking: A survey, use cases, and future directions,” *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2359–2391, Fourthquarter 2017.
- [12] J. G. Herrera and J. F. Botero, “Resource allocation in NFV: A comprehensive survey,” *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.
- [13] M. C. Luizelli, D. Raz, and Y. Sa’ar, “Optimizing NFV chain deployment through minimizing the cost of virtual switching,” in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 2150–2158.
- [14] A. Ceselli, M. Premoli, and S. Secci, “Mobile edge cloud network design optimization,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1818–1831, Jun. 2017.
- [15] 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 Manag.*, vol. 15, no. 1, pp. 475–488, Mar. 2018.
- [16] R. Lin, L. He, S. Luo, and M. Zukerman, “Energy-aware service function chaining embedding in NFV networks,” *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1158–1171, Mar.–Apr. 2023.

- [17] R. Gouareb, V. Friderikos, and A.-H. 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.
- [18] D. Chemodanov, P. Calyam, and F. Esposito, "A near optimal reliable composition approach for Geo-distributed latency-sensitive service chains," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1792–1800.
- [19] D. Zheng, G. Shen, X. Cao, and B. Mukherjee, "Towards optimal parallelism-aware service chaining and embedding," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 2063–2077, Sep. 2022.
- [20] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.
- [21] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2479–2489, Nov. 2017.
- [22] A. Fischer, D. Bhamare, and A. Kassler, "On the construction of optimal embedding problems for delay-sensitive service function chains," in *Proc. 28th Int. Conf. Comput. Commun. Netw.*, 2019, pp. 1–10.
- [23] M. A. Khoshkholghi et al., "Service function chain placement for joint cost and latency optimization," *Mobile Netw. Appl.*, vol. 25, no. 6, pp. 2191–2205, 2020.
- [24] 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 Conf. Comput. Commun.*, 2020, pp. 267–276.
- [25] L. Guo, J. Pang, and A. Walid, "Joint placement and routing of network function chains in data centers," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 612–620.
- [26] L. Qu, M. Khabbaz, and C. Assi, "Reliability-aware service chaining in carrier-grade software networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 558–573, Mar. 2018.
- [27] 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, Apr. 2018.
- [28] D. Guo, B. Ren, G. Tang, L. Luo, T. Chen, and X. Fu, "Optimal embedding of aggregated service function tree," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2584–2596, Oct. 2022.
- [29] 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.
- [30] 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 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 934–943.
- [31] D. Xiao, J. A. Zhang, X. Liu, Y. Qu, W. Ni, and R. P. Liu, "A two-stage GCN-based deep reinforcement learning framework for sfc embedding in multi-datacenter networks," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 4, pp. 4297–4312, Dec. 2023.
- [32] C. Sun, J. Bi, Z. Meng, T. Yang, X. Zhang, and H. Hu, "Enabling NFV elasticity control with optimized flow migration," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2288–2303, Oct. 2018.
- [33] P. Sun, J. Lan, J. Li, Z. Guo, Y. Hu, and T. Hu, "Efficient flow migration for NFV with graph-aware deep reinforcement learning," *Comput. Netw.*, vol. 183, 2020, Art. no. 107575.
- [34] K. Qu, W. Zhuang, Q. Ye, X. Shen, X. Li, and J. Rao, "Dynamic flow migration for embedded services in SDN/NFV-enabled 5G core networks," *IEEE Trans. Commun.*, vol. 68, no. 4, pp. 2394–2408, Apr. 2020.
- [35] 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. 11th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2011, pp. 195–204.
- [36] 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.*, vol. 20, no. 2, pp. 445–459, Feb. 2021.
- [37] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge U.K.: Cambridge Univ. Press, 2004.
- [38] V. V. Vazirani, *Approximation Algorithms*, vol. 1. Berlin, Germany: Springer, 2001.
- [39] "Gurobi," (n.d.). [Online]. Available: <https://www.gurobi.com/>
- [40] B. Claise, G. Sadasivan, and V. Valluri, "Cisco systems netflow services export version 9," RFC 3954, Internet Engineering Task Force, Oct. 2004, doi: [10.17487/RFC3954](https://doi.org/10.17487/RFC3954). [Online]. Available: <https://www.rfc-editor.org/rfc/rfc3954>
- [41] "Kubernetes," (n.d.). [Online]. Available: <https://kubernetes.io/>
- [42] W. Zhang et al., "OpenNetVM: A platform for high performance network service chains," in *Proc. Workshop Hot Topics Middleboxes Netw. Function Virtualization*, 2016, pp. 26–31.
- [43] 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, Oct. 2011.
- [44] "Networkx," (n.d.). [Online]. Available: <https://networkx.org/>
- [45] "Ryu," (n.d.). [Online]. Available: <https://ryu-sdn.org/>
- [46] "Open vswitch," (n.d.). [Online]. Available: <http://www.openvswitch.org/>



Weihan Chen received the PhD degree from the Department of Computer Science and Technology, Tsinghua University, and the BE degree in telecommunications engineering and management and the ME degree in software engineering from the Beijing University of Posts and Telecommunications, China. He is currently with Zhongguancun Laboratory, Beijing, China. His research interests include network function virtualization, software defined networking, and traffic engineering.



Zhiliang Wang (Member, IEEE) received the BE, ME, and PhD degrees in computer science from Tsinghua University, China, in 2001, 2003, and 2006, respectively. He is currently an associate professor with the Institute for Network Sciences and Cyberspace, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University. His research interests include formal methods and protocol testing, next generation internet, and network measurement.



Han Zhang (Member, IEEE) received the BE degree in computer science and technology from Jilin University, and the PhD degree from Tsinghua University. He is currently an assistant professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include computer networks, network security, and network system.



Xia Yin (Senior Member, IEEE) received the BE, ME, and PhD degrees in computer science from Tsinghua University, in 1995, 1997, and 2000, respectively. She is currently a full professor with the Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University. Her research interests include future Internet architecture, formal method, protocol testing and large-scale Internet routing.



Xingang Shi (Member, IEEE) received the BS degree from Tsinghua University, and the PhD degree from The Chinese University of Hong Kong. He is currently an associate professor with the Institute for Network Sciences and Cyberspace, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University. His research interests include network measurement and routing protocols.