

Cost-efficient flow migration for SFC dynamical scheduling in geo-distributed clouds

Weihan Chen^a, Zhiliang Wang^{b,a,c,*}, Han Zhang^{b,a,c}, Xia Yin^{d,a,c}, Xingang Shi^{b,a,c}

^a Zhongguancun Laboratory, Beijing, China

^b Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China

^c Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing, China

^d Department of Computer Science and Technology, Tsinghua University, Beijing, China

ARTICLE INFO

Keywords:

Network function virtualization
Service function chain
Flow migration
Software defined network

ABSTRACT

Combining Network Function Virtualization (NFV) and Software Defined Network (SDN) enables the delivery of network services in a flexible and cost-efficient manner. NFV allows elasticity scheduling of Virtual Network Function (VNF) instances to adopt the dynamic variation of network flows, such as VNF instance scale out or in and load balancing. To implement the elasticity scheduling, existing flows need to be re-routed through the VNF instances. One major problem is how to select appropriate flows for migration to reduce migration cost and improve network performance after migration. In this paper, we propose a Cost-efficient Flow Migration framework, CFM, to jointly optimize flow migration cost and network performance after migration (including flow routing cost and maximum VNF instance load) for flow migration in Service Function Chain manner. Experimental results demonstrate that CFM can reduce flow routing cost after migration by 43% ~ 62% compared with previous works, while achieving similar flow migration cost in scale out/in scenario. In load balancing scenario, CFM can also reduce flow migration cost by 65% ~ 79% and flow routing cost after migration by 43% ~ 79% compared with previous works, while ensuring that the maximum VNF instance load does not significantly increase.

1. Introduction

Today's network service operators prefer to implement Network Function Virtualization (NFV) [1] technology in enterprise and edge networks as a replacement for dedicated hardware middleboxes. In contrast to hardware devices, NFV enables the implementation of network functions in a software-based manner on commercial servers. These software-based network functions are called as Virtual Network Functions (VNFs), typically running in Virtual Machines (VMs) or other containers such as Docker. These containers that host VNFs are known as VNF instances (VNFIs). NFV enhances network services by offering flexibility and reduces deployment and operational costs. Complemented by Software Defined Network (SDN) technology, operators can manage traffic flows more flexibly and achieve specific objectives through the installation of routing rules [2].

Network flows typically traverse multiple VNFs in a specific order to achieve predefined objectives. These sequenced VNFs are known as Service Function Chains (SFCs) [3]. While SFCs are often assumed to

be deployed within a single server or a Local Area Network (LAN) composed of multiple servers, representing a non-geo-distributed deployment, they can also be deployed in a geo-distributed manner due to resource constraints in physical devices, latency or robustness requirements, or functional considerations [4].

There are two typical network scenarios that may necessitate geo-distributed deployment of SFCs. The first scenario pertains to edge cloud networks. Considering that edge clouds typically have limited resources and may not handle the increasing user flows independently, they require collaboration with other edge clouds or remote public clouds to deliver services. In this case, flows need to traverse multiple geo-distributed clouds [5].

The second scenario involves Software-Defined Wide Area Network (SD-WAN). Companies like Amdocs¹ offer NFV SD-WAN solutions on Microsoft Azure² cloud. These solutions allow service providers to configure and manage VNFs across multiple clouds, connecting them into an SD-WAN infrastructure [6]. This implies that network services

* Corresponding author at: Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China.

E-mail addresses: chwh@zgclab.edu.cn (W. Chen), wzl@cernet.edu.cn (Z. Wang), zhhan@tsinghua.edu.cn (H. Zhang), yxia@tsinghua.edu.cn (X. Yin), shixg@cernet.edu.cn (X. Shi).

¹ Amdocs, <https://www.amdocs.com/>.

² Microsoft Azure, <https://azure.microsoft.com/>.

can also be offered in a geo-distributed SFC manner when necessary, with flow routing among clouds controlled by the SD-WAN controller. In geo-distributed cloud networks, careful consideration of flow routing is crucial, as routing flows between clouds can result in higher costs compared to non-geo-distributed cloud networks.

When providing network services, an NFV-enabled network must adapt to dynamic flow variations. Besides allocating computing resources directly to Virtual Network Function Instances (VNFI), migrating flows between VNFs is also necessary to maintain network service performance [7–9]. For example, if increased flows overload specific VNFs in an SFC, it requires to migrate a portion of the flows from the overloaded VNFs to non-overloaded ones or newly launched VNFs [10]. In [7], flow migration triggers can be categorized as follows: (1) **Scale out**: when certain VNFs within the SFC become overloaded; (2) **Scale in**: when certain VNFs within the SFC are underutilized; (3) **Load balance**: when the load among VNFs within the SFCs is imbalanced; and (4) **VNF Failure Recovery/VNF Upgrading**: when VNF failures occur or require upgrading. The primary challenge in handling flow migration is *how to select the appropriate flows for migration while optimizing both the migration cost and network performance post-migration*. Specific challenges include:

(1) **Optimize migration cost during migration**: Flow migration operations require the temporary suspension of services for the affected flows. These active flows during the service suspension are termed “in-flight” flow. Prolonged migration operations can significantly impact service quality. Therefore, it is crucial to minimize migration costs when selecting flows to migrate. Additionally, maintaining a buffer in the cloud to hold VNFs is essential for preserving in-flight flows during migration. Careless flow selection could lead to buffer space overflow, resulting in packet loss or service degradation. While prior efforts (e.g., [7]) have aimed to reduce flow migration costs and consider buffer overflow, they mainly focus on optimizing migration cost for individual VNF, overlooking routing performance across multiple VNFs within the SFC. This means that their solutions may not directly apply to optimizing flow migration in SFC scenarios.

(2) **Optimize network performance after migration**: Assessing network performance after migration involves several aspects. Maximum VNFI load is a vital metric. Minimizing maximum VNFI load can ensure the processing performance of VNFs and utilization of network resources. Meanwhile, optimizing the routing cost after migration based on the delay requirements of flows is also important. Since flows require both delay-sensitive services (tolerate hundreds of milliseconds delay) and delay-tolerant services (tolerate maximum hours delay) for serving [11,12], if the resources are limited in clouds, it is better to prioritize the flows that require delay-sensitive services using resources on clouds with lower transmission delay. However, existing works like [8,9] only focus on optimizing the VNFI load after migration but ignore the routing performance optimization.

(3) **Trade-off between migration cost and network performance after migration**: When making flow migration decisions, a migration scheme with low migration cost does not guarantee good network performance after migration, and vice versa. This requires comprehensively evaluating the migration cost and the network performance that can be achieved after migration for each flow, and making trade-offs to finally achieve the overall optimal migration scheme.

To address the above challenges, we propose a flow migration framework for geo-distributed clouds network, called as CFM framework. The CFM framework can achieve optimized flow migration cost and network performance after migration (maximal VNFI load and flow routing cost) when flow migration events are triggered. The main contributions of this paper are shown as follows:

- We propose a general flow migration optimization framework in SFC scenario, which covers the following flow migration events: scale out/in, load balance, VNF failure recovery and VNF upgrading. We also analyze the costs to be optimized and the flow selection targets for each event (Section 3).

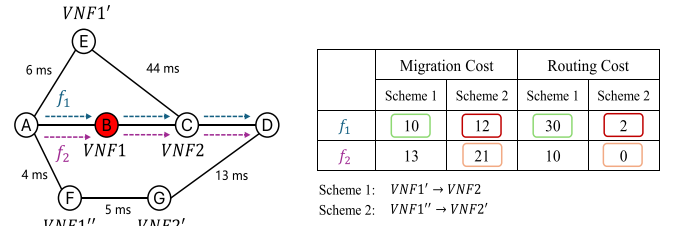


Fig. 1. Example of different flow migration schemes in scale out scenario.

- We formulate the optimization models for each flow migration event. The models can help to achieve optimized flow migration-related costs while ensuring the migration targets being satisfied (Section 4).
- We design proper algorithms for different flow migration events in CFM. For scale out/in events, we propose greedy flow selection algorithms to achieve efficient flow migration. For load balance event, we propose an approximation algorithm to realize near-optimal flow migration schemes (Section 5).
- We conduct extensive simulation experiments to evaluate the optimization effect of CFM. The results demonstrate that CFM can achieve well trade-off between flow migration cost and network performance after migration. For example, in load balance scenario, CFM achieves similar maximum VNFI load compared with state-of-the-art work, but reduces the flow migration cost and routing cost after migration about 65% and 50% respectively (Section 6).

2. Motivation example

In geo-distributed cloud networks, flows requiring SFC services often traverse multiple cloud nodes, which may result in varying routing delays. This discrepancy leads to significant differences in migration and post-migration routing costs across different migration schemes. Therefore, selecting diverse post-migration routing paths during migration can greatly affect the direction of cost optimization.

For example, in the Scale Out scenario, the primary objective of flow migration is to alleviate the load on overloaded VNF instances to below a specified threshold. Our main focus here is to optimize the migration and post-migration routing costs. Migration costs mainly encompass the time spent on VNF state migration and packet redirection, while post-migration routing costs relate to the quality of service violations due to routing delays not meeting flow delay requirements. As shown in Fig. 1, the VNF1 instance on node B is overloaded. The parameters of VNFs and flows include: the state migration delay of VNF1 is 7 ms, the single flow entry installing delay is 1 ms, the redirecting delay for a single packet of flow is 0.5 ms, the flow rates of f_1 and f_2 are 2000 packets per second and 8000 packets per second respectively, and the expected service completion delay of f_1 and f_2 are 20 ms and 40 ms respectively. Additionally, the forwarding delay of part links are also shown in Fig. 1. Suppose that either flow f_1 or f_2 can be migrated to ease the overload. Based on the calculation methods shown in Section 4, we can calculate the costs associated with various migration and post-migration flow routing schemes, as shown in the table of Fig. 1. If the aim is solely to minimize migration costs, f_1 would be migrated along the $VNF1 \rightarrow VNF2$ path (Scheme 1), resulting in a total cost of 10. Conversely, selecting the $VNF1' \rightarrow VNF2'$ path for f_2 migration (Scheme 2) would minimize post-migration flow routing costs, resulting in a total cost of 21. However, neither of these schemes optimizes the combined migration and post-migration routing costs. By striking an appropriate balance between the two costs, migrating f_1 according to Scheme 2 yields the lowest total cost of 14.

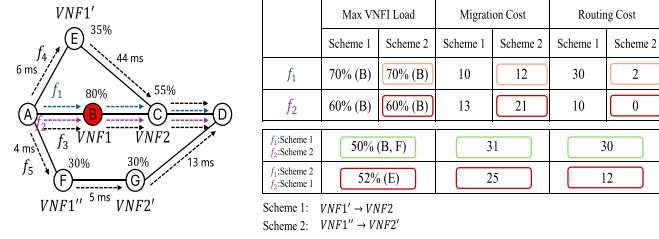


Fig. 2. Example of different flow migration schemes in load balance scenario.

In the Load Balance scenario, besides considering the costs of flow migration and post-migration flow routing, we also need to optimize the maximum load on VNFI after migration. This creates a notable optimization dilemma between VNFI load and flow migration cost, as migrating more flows is likely to alleviate VNFI load. As depicted in Fig. 2, the load on VNFI instance at node B is relatively high (80%). To balance the load between VNFI, we can consider migrating flow f_1 , f_2 , or both (assuming other flows cannot be migrated due to specific reasons). Suppose the parameters are set the same as in the Scale Out example, and the maximum load of VNFI after migration are shown in the table of Fig. 2. If minimizing the maximum VNFI load is the primary optimization goal, f_1 will be migrated along the $VNF1' \rightarrow VNF2$ path (Scheme 1), and f_2 will be migrated along the $VNF1'' \rightarrow VNF2'$ path (Scheme 2). However, this migration scheme incurs high flow migration cost and post-migration flow routing cost, resulting in a maximum VNFI load of 50% and a total cost (migration cost + post-migration routing cost) of 61. If minimizing the cost of flow migration or post-migration routing (or both) is the main objective, only f_1 will be migrated according to Scheme 2. However, this migration approach poorly optimizes the maximum VNFI load, resulting in a load of 70% after migration and a total cost of 14. Yet, by jointly optimizing the three costs, we would migrate f_1 according to Scheme 2 and f_2 according to Scheme 1. This migration scheme prioritizes ensuring VNFI load optimization effects, and achieves a maximum VNFI load of 52% after migration, with a total cost of 37 for flow migration and post-migration routing. If the optimization priority is to ensure migration and post-migration routing costs, we can choose to migrate only f_2 according to Scheme 2, which results in a maximum VNFI load of 60% after migration, with a total cost of 21 for flow migration and post-migration routing. While sacrificing a portion of VNFI load and the sum of flow migration cost and post-migration routing cost, this scheme achieves a more balanced overall optimization effect.

The examples above highlight the importance of jointly optimizing migration costs and post-migration flow routing costs. By striking suitable compromises in optimization, we can prevent an escalation in costs favoring one type of optimization over another (e.g., in the Scale Out scenario, migrating flow f_2 according to Scheme 2 led to excessively high migration costs). During joint optimization, network operators can adjust their optimization strategies according to actual requirements (e.g., in the Load Balance scenario, if operators prioritize optimizing migration costs and post-migration routing costs and are willing to tolerate higher VNFI loads, they may opt to migrate only flow f_2 according to Scheme 2). In our designed flow migration optimization framework, CFM, we aim to jointly optimize various costs relevant to different scenarios. The simulation experiment results in Section 6 also demonstrate that CFM achieves a more balanced optimization effect compared to existing approaches that focus solely on optimizing a single cost (such as optimizing VNFI load only in the Load Balance scenario).

3. CFM framework overview

The geo-distributed cloud network in CFM consists of switch nodes (S), cloud nodes (N), and links (L), represented as an undirected graph

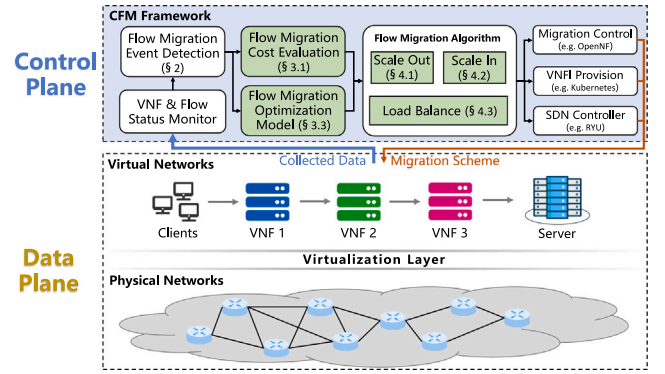


Fig. 3. CFM framework modules and workflow.

$G = (S, L)$. Cloud nodes in N can connect directly to switch nodes in S . Within the cloud nodes, there is a set of VNFI (I), some of which remain active to process flows, while others are inactive and can be launched when the active ones cannot handle all flows. Active VNFI can also be deactivated when the number of flows decreases to save costs.

In this cloud network, the set of user flows is denoted as K , and these flows dynamically change over time, leading to flow migration events. CFM monitors these events and determines how to manage flow migration while aiming to minimize related migration costs. It also strives to optimize network performance after migration. The framework modules and workflow of CFM are depicted in Fig. 3.

The *Flow Migration Event Detection* module is responsible for identifying the occurrence of flow migration events. In this module, we mainly employ the commonly used methods in current work to determine the timing of various types of flow migration operations.³ It uses predefined upper and lower load thresholds (ψ_{up} and ψ_{low}), and calculates the load ψ_i for each VNFI $i \in I$ based on status information (e.g., flow rate) collected by the *VNF and Flow Status Monitor* module.⁴ If the load of a VNFI falls outside the range defined by these two thresholds, the detection module considers executing scale-out or scale-in operations for the corresponding VNFI.

Even when the load of VNFI falls within the normal range, the detection module must also consider the possibility of VNFI load imbalance. VNFI load imbalance can more easily lead to VNFI becoming overloaded or underutilized, or resulting in inefficient resource utilization. Hence, a load balancing operation is required to redistribute flows across the VNFI and balance the load on each VNFI. The load balancing operation can be triggered either when the variance of VNFI loads exceeds a certain threshold, denoted as ψ_{var} (similar to [7]), or it can be triggered periodically.

As for VNF failure recovery and VNF upgrading events, these are triggered when a VNFI failure is detected or when a VNF upgrade request is issued by network operators. In summary, the conditions that trigger flow migration events are as follows:

- **Scale Out:** if some VNFI load $\psi_i \geq \psi_{up}$ in SFC;
- **Scale In:** if some VNFI load $\psi_i \leq \psi_{low}$ in SFC, and VNFI shutdown cost outweighs the benefit;
- **Load Balance:** if the variance of VNFI loads satisfies $var(\psi_1, \dots, \psi_i) \geq \psi_{var}$ or fixed load balance period P_{lb} is met;

³ Determining flow migration timing based on fixed threshold method or current faulty VNF detection method may have limitations. However, finding more suitable flow migration timing and more accurate faulty VNF detection mechanism is not the main contribution of this paper. We mainly optimize flow migration schemes for specific migration events.

⁴ VNF and Flow Status Monitor module can be supported by existing VNFI provisioning platforms and the OpenFlow protocol [13] to collect flow status.

- **VNF Failure Recovery/VNF Upgrading:** if failure events or upgrading requirements are detected.

Once the triggering flow migration event is identified, the following modules are essential for subsequent handling: The *Flow Migration Cost Evaluation* and *Flow Routing Cost Evaluation* modules assess the costs associated with flow migration. The *Flow Migration Algorithm* module then selects specific migration schemes based on the cost assessment and the migration requirements associated with the triggered flow migration events. We outline how our framework handles flow migration based on event types:

(1) **Scale Out:** Scale out operation involves selecting suitable flows from the overloaded VNFI and migrating them to other VNFI. Throughout the migration, inactive VNFI may be activated to handle additional flows that the existing active VNFI cannot accommodate. The number of migrated flows must reduce the load of the overloaded VNFI below ψ_{up} after migration. While minimizing migration costs and post-migration routing costs, it is also essential to prevent overloading of other VNFI, violations of link bandwidth constraints, and buffer overflow for in-flight flows.

(2) **Scale In:** Once the VNFI that need to be shut down are identified, all flows passing through these VNFI must be migrated out. The focus is on optimizing both flow migration costs and post-migration routing costs. Migrated flows will be redistributed to existing active VNFI in the network without activating inactive VNFI. It is crucial to avoid issues like VNFI overloading, violations of link bandwidth constraints, and buffer overflow for in-flight flows.

(3) **Load Balance:** Load balancing operations aim to migrate suitable flows from heavily loaded VNFI to lightly loaded VNFI to address load imbalances among VNFI. This is akin to achieving load balance in traffic engineering. When flow transmission rates remain fixed, enhancing the utility of VNFI (or links) is equivalent to minimizing the maximum VNFI (or link) load ratio [14]. Therefore, one objective of load balancing in CFM is to minimize the maximum load Ψ among all VNFI. However, solely minimizing the maximum load may result in migrating too many flows, potentially impacting the performance of flows after migration. Consequently, we consider both flow migration costs and post-migration flow routing costs, optimizing them jointly with the maximum VNFI load. This balance among costs prevents network performance deterioration due to excessive focus on a single cost aspect.

(4) **VNF Failure Recovery/VNF Upgrading:** VNF failure recovery and VNF upgrading scenarios resemble the *Scale In* event. In both cases, flows passing through VNFI that are faulty or in need of upgrading must be migrated out. The focus is on optimizing the costs associated with flow migration and post-migration flow routing.

All the processes described above are executed by sub-modules within the *Flow Scheduling Algorithm* module, namely, *Scale Out*, *Scale In*, and *Load Balance* (VNF Failure Recovery and VNF Upgrading events can employ *Scale In* sub-module to find flow migration schemes). Following the generation of specific flow migration schemes by these sub-modules, CFM collaborates with the *Migration Control* module, *VNFI Provision* module, and *SDN Controller* module to implement the migration schemes in the data plane. For example, existing VNF migration control frameworks (e.g., OpenNF [15]) can assist in managing the state migration of VNFs, cloud container orchestration platforms (e.g., Kubernetes [16]) can control VNFI provisioning, and SDN controllers (e.g., RYU [17]) can manage flow routing rules in the data plane.

4. Flow migration cost calculation

The calculation of flow migration costs primarily relies on the post-migration path. To facilitate cost computation, we introduce a binary variable, $m_{p',p}^k$, which indicates whether flow k is migrated from path p' to path p . When $m_{p',p}^k = 1$, it signifies that flow k is now routed through

Table 1

Key notation.

Topology related variable	
N, S	The set of cloud nodes, switch nodes
L	The set of links
I, K	The set of deployed VNFI and traffic flows
C_i^n	Processing capacity of VNFI i in node n
A_i^n	Indicator of whether VNFI i is located at node n
B_l	Bandwidth capacity of link l
Q_n	Queue capacity used to buffer packets of node n
I_n	The set of deployed VNFI in node n
G_i	The set of VNFI which should be shut down
θ_p	Routing delay of path p
Φ	Redirecting delay for a single packet of flow
Γ	Single flow entry installing delay
SFC related variable	
ζ_f	State migration delay of VNF f
p_f	Processing delay of VNF f
γ_i^f	Indicator of whether VNFI i belongs to VNF f
Flow related variable	
R_k	Flow rate of flow k
P_k	Feasible candidate path set of flow k
e_k	Expected service completion delay of flow k
DT_k	Maximum service suspend delay of flow k
Decision variable	
$m_{p',p}^k$	Indicator of whether flow k is migrated from path p' to path p
Auxiliary variable	
$x_{p,i}^l$	Indicator of whether link l is the direct successor link of VNFI i on path p
y_p^i	Indicator of whether VNFI i belongs to path p
$m_{i',i}^k$	Indicator of whether flow k is migrated from VNFI i' to VNFI i
π_i	Indicator of whether VNFI i is the first instance happening flow migration in SFC

path p instead of p' . In our model, the routing path includes both switch nodes and the VNFI (along with their corresponding cloud nodes).

As illustrated in Fig. 4, the original routing path for the flow before migration is $S1 \rightarrow S3 \rightarrow A1(N1) \rightarrow S3 \rightarrow S4 \rightarrow B1(N3) \rightarrow S4 \rightarrow S6 \rightarrow C1(N4) \rightarrow S6 \rightarrow S8 \rightarrow D1(N6) \rightarrow S8 \rightarrow S9$, and the path after migration becomes $S1 \rightarrow S2 \rightarrow A2(N2) \rightarrow S2 \rightarrow S4 \rightarrow B1(N3) \rightarrow S4 \rightarrow S5 \rightarrow C2(N5) \rightarrow S5 \rightarrow S7 \rightarrow D2(N7) \rightarrow S7 \rightarrow S9$. Based on the “path”, it is straightforward to determine whether a VNFI is part of a specific path and whether a link is the immediate successor link of a VNFI. We employ two binary auxiliary variables, y_p^i and $x_{p,i}^l$, to represent these affiliation relationships. Specifically, $y_p^i = 1$ indicates that VNFI i belongs to path p , and $x_{p,i}^l = 1$ indicates that link l is the direct successor link of VNFI i on path p (as depicted in Fig. 4, where $S4 \rightarrow S5$ is the direct successor link of VNFI $B1$ but not of VNFI $A2$).

Additionally, we can compare the paths before and after migration to determine whether VNFI undergo flow migration. A binary auxiliary variable, $m_{i',i}^k$, is employed to indicate whether flow k is migrated from VNFI i' to i (both VNFI i' and VNFI i belong to the same type of VNF). If flow migration occurs, then $m_{i',i}^k = 1$ (e.g., as shown in Fig. 4, $m_{A1,A2}^k = 1$).

It is feasible to precompute all (or part) of the viable paths for each SFC when the topology and VNFI deployment details are known. By comparing the variations in paths before and after migration, we can readily compute the costs incurred during the flow migration, as well as the flow routing costs after migration. Key notations are summarized in Table 1.

4.1. Cost analysis for flow migration

(I) Flow Migration Cost

The flow migration cost within CFM primarily encompasses the time cost associated with VNF state transition, buffered packet redirection, and routing rule updates. We employ a flow migration pattern akin

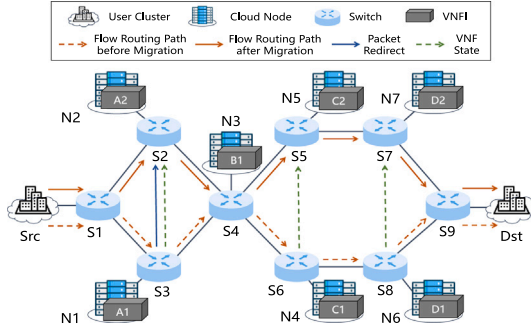


Fig. 4. Example of flow migration in CFM framework.

to [9], which parallelly migrates the state of VNFs within the SFC and synchronously updates routing rules. In contrast to [9], we modify the buffered packet redirection mode on the initially migrated VNF. Instead of routing packets from the original VNF to the target VNF via the controller, we adopt the approach presented in [18]. This approach allows packets to be redirected directly between VNFs by establishing a point-to-point connection between the original and target VNFs, reducing packet redirection delays.

A simple example is given to illustrate the flow migration operation logic in CFM. As shown in Fig. 4, if flows traverse the *SFC*: $\{VNF A \rightarrow VNF B \rightarrow VNF C \rightarrow VNF D\}$ and need to migrate from $\{A1 \rightarrow B1 \rightarrow C1 \rightarrow D1\}$ to $\{A2 \rightarrow B1 \rightarrow C2 \rightarrow D2\}$, the migration process is as follows: (a) Simultaneously export states from VNFs $\{A1, C1, D1\}$ and import states into VNFs $\{A2, C2, D2\}$. (b) Before the state transition of $A1$ completes, buffer the packets of newly arriving flows at $A1$. (c) Once the state transition of $A1$ finishes, directly redirect the packets buffered at $A1$ to $A2$. (d) Update the routing rules on $\{S1, S2, S4, S5\}$ so that packets from the source (*Src*) reach $C2$ after processing by $A2$ and $B1$. (e) $C2$ also buffers packets until the state transition from $C1$ is completed. Then, $C2$ processes the packets and forwards them to $D2$ by updating the routing rules on $S5$. (f) Similarly, $D2$ buffers the packets until the state transition from $D1$ is concluded and then processes them. After the routing rules are updated, the packets are forwarded to the destination (*Dst*) through $\{S7, S9\}$.

Gember-Jacobson et al. [18] have demonstrated that the state transition time is primarily influenced by the number of migrated flows rather than the size of individual flows. Additionally, Sun et al. [7] have observed a linear relationship between the state transition time of a single flow and the number of migrated flows through actual measurements. However, since it is not possible to predict the number of migrated flows in advance, [7] assumes that each flow is migrated independently and introduces the concept of Single-Flow Migration Time (SFMT). We adopt the same evaluation approach, considering the state transition time of each flow as a fixed value (represented as ζ_f), which varies depending on the VNF type. We also employ a binary variable γ_i^f to indicate whether VNF i belongs to VNF f .

Following the flow migration logic, it is important to note that only the first migrated VNF in the SFC needs to redirect the buffered packets. To represent whether VNF i is the first migrated VNF, we use a binary variable π_i . The value of π_i can be easily determined if we know the routing path of the flow after migration. For instance, in Fig. 4, $A1$ is the first migrated VNF, which means $\pi_{A1} = 1$.

The number of packets buffered on the first migrated VNF is determined by the difference between the state transition delay of the first migrated VNF and the routing delay of the flow from the source point to this VNF. We represent the routing delay of flow reaching VNF i as d_i , and the delay of redirecting a single packet of flow as Φ . Given the flow rate R_k of flow k passing through the first migrated VNF, we can calculate the redirecting delay of buffered packets for flow k by: $\max(0, \zeta_f \gamma_i^f - d_i) R_k \pi_i \Phi$.

In an SDN environment, the controller can install or modify flow entries to facilitate the updating of routing rules. The number of installed flow entries directly affects the delay of routing rule updates. We denote the number of flow entries that need to be installed or modified on switch node s when flow k is migrated as Z_s^k . For switch nodes that are directly connected to cloud nodes holding VNF instances after migration, represented by g_s^i (where g_s^i equals 1 if switch node s is directly connected to the cloud node holding VNF i , and 0 otherwise), the controller needs to install two flow entries. One entry is for forwarding the flow to the cloud nodes holding VNFs, and the other is for forwarding the flow to the next switch node along the path. For example, in Fig. 4, $S2$ needs to install two flow entries to ensure that flow k from $S1$ can be routed to $N2$, and the flow k processed by $A2$ can be routed to $S4$, resulting in $Z_{S2}^k = 2$. On the other hand, switch nodes like $S1$ and $S9$, which are not directly connected to the cloud nodes, need to install one flow entry each to ensure that flows from the predecessor VNF (or to the successor VNF) that undergo flow migration can be forwarded. For example, $S1$ requires one new flow entry to forward flows from *Src* to $S2$ after migration, which means $Z_{S1}^k = Z_{S9}^k = 1$. In cases where both the predecessor and successor VNFs of the current VNF experience flow migration, like VNF $B1$ in Fig. 4, $S4$ needs to install two flow entries: one for forwarding flows from $S2$ to $B1$ and another for forwarding flows from $B1$ to $S5$, resulting in $Z_{S4}^k = 2$. In all other scenarios, there is no need to update the flow entries on the switch nodes. Finally, Z_s^k is defined as follows⁵:

$$Z_s^k = \begin{cases} 2, & \text{if } g_s^{i'} = 1, m_{i',i}^k = 1 \\ 2, & \text{if } g_s^{i'} = 1, m_{i',i}^k = 0, m_{i',i}^{k_{pr}} + m_{i',i}^{k_{su}} = 2 \\ 2, & \text{if } g_s^{i'} = 0, m_{i',i}^k = 0, m_{i',i}^{k_{pr}} + m_{i',i}^{k_{su}} = 2 \\ 1, & \text{if } g_s^{i'} = 1, m_{i',i}^k = 0, m_{i',i}^{k_{pr}} + m_{i',i}^{k_{su}} = 1 \\ 1, & \text{if } g_s^{i'} = 0, m_{i',i}^k = 0, m_{i',i}^{k_{pr}} + m_{i',i}^{k_{su}} = 1 \\ 0, & \text{otherwise} \end{cases}$$

Subsequently, we can calculate the update delay for routing rules for flow k on switch node s by: $Z_s^k \Gamma$ (where Γ is the delay associated with installing a single flow entry).

Considering that operations such as state transition and routing rules update on different VNFs in the SFC can be performed in parallel, but state transition, buffered packet redirection, and routing rules update on a single VNF need to be performed sequentially [18] shows that the execution order should be: *state transition* $\rightarrow$ *buffered packets redirection* $\rightarrow$ *routing rules update*. Therefore, we only need to consider the maximum delay generated by performing migration-related operations on a single VNF as the flow migration cost. The calculation method is shown in Eq. (1).⁶ We use M_k to represent the migration cost for flow k :

$$M_k = \sum_{p \in P_k} \max_{i' \in p', i \in p} (\zeta_f \gamma_i^f m_{i',i}^k + \max(0, \zeta_f \gamma_i^f - d_i) R_k \pi_i \Phi + g_s^i Z_s^k \Gamma) m_{p',p}^k \quad (1)$$

The routing delay d_i for flows from the source to VNF i is essential for calculating the migration cost M_k and determining the number of buffered packets ($S_i^{k,n}$ represents the count of buffered packets for flow k on VNF i in cloud node n). Due to space limitations, the detailed calculations and formulation for d_i are provided in the supplementary

⁵ In this equation, subscript *pr* (*su*) represents the predecessor (successor) VNF of the current VNF, which belongs to the routing path before migration or the routing path after migration.

⁶ Since the updating of routing rules can be performed in parallel, the update delay of switch nodes not directly connected to the cloud node will not exceed the switch nodes that are directly connected to the cloud node holding VNF. The time cost incurred by these switch nodes ($\sum_{i \in p} g_s^i = 0$) can be ignored.

material (Appendix A). The number of buffered packets for flow k on VNFI i in cloud node n can be calculated by:

$$S_i^{k,n} = \sum_{p \in P_k} \max(0, \zeta_f \gamma_i^f - d_i) R_k y_p^i A_i^n m_{p',p}^k \quad (2)$$

An illustrative example for calculating $S_i^{k,n}$ is also provided in the supplementary material (Appendix A).

(II) Flow Routing Cost

For flow routing cost after migration, most existing works simply optimize the end-to-end delay of all flows but ignore the different delay requirements of flows. Especially, when the computing resources of cloud nodes are limited, it is important to allocate routing paths for each flow based on its delay requirements. Hence, we introduce Service Level Agreement (SLA) violation cost to evaluate the flow routing cost after migration in CFM. SLA violation cost refers to the cost incurred by the actual end-to-end delay exceeding the expected completion delay of network services, and it can be evaluated as SLA Violation Cost = $\alpha + \beta \times \max(0, \text{Actual E2E Delay} - \text{Expected Delay})$, where α and β are cost factors [19].

That means we should calculate the end-to-end delay of a flow before computing the SLA violation cost. The end-to-end delay of a single flow is primarily composed of path routing delay (θ_p) and VNF processing delay (p_f). The path routing delay is the sum of the forwarding delay of the links that constitute the path. Both the link forwarding delay and the VNF processing delay can be measured. We use D_p^k to represent the end-to-end delay of flow k through path p , and it can be calculated as $D_p^k = \theta_p + \sum_{i \in p} p_f \gamma_i^f$. Once we determine the end-to-end delay of the flow, we can compute the SLA violation cost using the formula shown in Eq. (3).

$$T_k = \sum_{p \in P_k} (\alpha + \beta \times \max\{0, D_p^k - \epsilon_k\}) m_{p',p}^k \quad (3)$$

4.2. Optimization model for flow migration

The cost-optimized mathematical model for flow migration in the scale out scenario is shown as follows:

$$\text{minimize } \sum_{k \in K} (M_k + T_k) \quad (4)$$

subject to:

$$M_k \leq DT_k, \forall k \in K \quad (4a)$$

$$\sum_{i \in I_n} \sum_{k \in K} S_i^{k,n} \leq Q_n, \forall n \in N \quad (4b)$$

$$C_i^n \psi_{low} \leq \sum_{k \in K} \sum_{p \in P_k} R_k y_p^i A_i^n m_{p',p}^k \leq C_i^n \psi_{up}, \quad (4c)$$

$$\forall i \in I/G_i, n \in N$$

$$\sum_{k \in K} \sum_{p \in P_k} \sum_{i \in p} x_{p,i}^l R_k \gamma_i^f m_{p',p}^k \leq B_l, \forall l \in L \quad (4d)$$

$$\sum_{p \in P_k} m_{p',p}^k = 1, \forall k \in K \quad (4e)$$

$$m_{p',p}^k \in \{0, 1\}, \forall k \in K, p \in P_k \quad (4f)$$

Constraint (4a) is implemented to guarantee that the suspension time of each flow k does not exceed a predefined threshold. Constraint (4b) sets a cap on the maximum number of packets that each cloud node can buffer. Constraint (4c) and (4d) establish capacity restrictions for VNFs and links, respectively. Constraint (4e) enforces the uniqueness of the routing path for each flow k .

In the scale in scenario, the optimization objective and constraints are similar to the scale out scenario. Additionally, we ensure that no more flows pass through the VNFs that have been marked for

shutdown (as constrained by (5a)). The mathematical model for flow migration in the scale in scenario is as follows:

$$\text{minimize } \sum_{k \in K} (M_k + T_k) \quad (5)$$

subject to: (4a)~(4f)

$$\sum_{k \in K} \sum_{p \in P_k} y_p^i A_i^n m_{p',p}^k = 0, \forall i \in G_i, n \in N \quad (5a)$$

In the load balance scenario, besides the flow migration cost (M_k) and flow routing cost after migration (T_k), our optimization objective also aims to minimize the maximum load ratio of VNFs (Ψ) after migration. The constraints are similar to the scale out scenario, with one difference being the VNFI load limitation (shown in (6a)). The upper bound of the load threshold for each VNFI is no longer a fixed value but depends on the optimization results. The mathematical model for the load balance scenario is as follows:

$$\text{minimize } \delta_1 \Psi + \delta_2 \sum_{k \in K} (M_k + T_k) \quad (6)$$

subject to: (4a), (4b), (4d)~(4f)

$$C_i^n \psi_{low} \leq \sum_{k \in K} \sum_{p \in P_k} R_k y_p^i A_i^n m_{p',p}^k \leq C_i^n \Psi, \quad (6a)$$

$$\forall i \in I/G_i, n \in N$$

In Eq. (6), the maximum load ratio of VNFs can be determined by $\Psi = \max_{i \in I} \psi_i$, where ψ_i represents the load ratio of VNFI i . This load ratio can be calculated by $\psi_i = \frac{\sum_{k \in K} \sum_{p \in P_k} R_k y_p^i A_i^n m_{p',p}^k}{C_i^n}$. Additionally, the setting of parameters δ_1 and δ_2 depends on the user's understanding of the network and their optimization preferences. For example, by leveraging the estimated range of M_k and T_k , δ_2 can effectively scale the combined range of $M_k + T_k$ to 1 ~ 10, while setting δ_1 to 10 ensures equivalence with the range of VNFI load Ψ (i.e., 0 ~ 1). When the values of the relevant costs are controlled within the same magnitude, a higher ratio of δ_1 to δ_2 indicates a greater inclination of the CFM algorithm towards optimizing VNFI load. Conversely, a lower ratio leans more towards optimizing flow migration and routing costs.

4.3. Optimization problem complexity analysis

In the **Scale out** and **Scale in** scenarios, if we only consider optimizing the post-migration routing cost, the optimization objective becomes: $\text{minimize } \sum_{k \in K} \sum_{p \in P_k} (\alpha + \beta \times \max\{0, D_p^k - \epsilon_k\}) m_{p',p}^k$. This is equivalent to the Integer Multi-Commodity Flow Problem (IMCFP) [20], where $m_{p',p}^k$ decides the routing path of each flow, and $\alpha + \beta \times \max\{0, D_p^k - \epsilon_k\}$ is the cost factor for the path. Hence, IMCFP is a special case of the optimization problems defined in Eqs. (4) and (5). As proved in [21], the two-commodity integral flow problem is NP-complete, which means the optimization problems defined in Eqs. (4) and (5) are also NP-complete, at least in terms of computational complexity.

In the **Load balance** scenario, as described in [9], an Extended Multi-Commodity Flow Problem (EMCF) is introduced to minimize the maximum load factor among all links by choosing a path for each flow. It has been proven that the EMCF problem is a general Parallel Machine Scheduling (PMS) problem [22]. If we simplify the problem defined in Eq. (6) by focusing only on optimizing the VNFI load ratios, it becomes equivalent to the EMCF problem. Therefore, the PMS problem is a special case of our optimization problem. Since the PMS problem is known to be NP-hard, it follows that our optimization problem defined in Eq. (6) is also NP-hard in terms of computational complexity.

4.4. General constraints for flow migration

During the flow migration process, minimizing service suspend time for flows to prevent service quality degradation is essential. In our

model, we define the service suspend time limit for flow k as DT_k . We also need to buffer packets for in-flight flows during the service suspend time to prevent packet loss. Some prior works [15,23] opt to buffer packets in a centralized controller. However, in a distributed cloud network, controlling the routing delay from local clouds to a central controller can be challenging and may result in excessive packet redirection time. Therefore, we prefer to have the clouds buffer packets locally to reduce packet redirection delays. To ensure efficient use of resources, we set a maximum buffer capacity limit for cloud nodes. Additionally, it is crucial to prevent the overloading of VNFI processing capacities and the violation of link bandwidth capacities during flow migration. Each VNFI i in node n has a computing capacity of C_i^n , and each link l has a bandwidth capacity of B_l .

5. Flow migration algorithm design

We present following algorithms used in CFM to achieve cost-efficient flow migration:

5.1. Scale out/in solution

We can utilize existing linear programming solvers, such as Gurobi [24], to solve the Integer Linear Programming (ILP) problem defined in Eq. (4) or (5) to obtain optimal flow migration schemes.⁷ However, the solving efficiency may not always be ideal. To enhance efficiency, we propose a greedy heuristic algorithm based on minimizing the cost of a single flow to fulfill flow migration requirements in both scale out and scale in scenarios.

The core concept of this algorithm is to prioritize the migration of flows with high flow rates and low total cost, which includes migration cost and routing cost after migration. This approach helps prevent the migration of numerous low-rate flows to meet VNFIs' load requirements. The algorithm's specifics are detailed in Algorithm 1. VNF failure recovery and VNF upgrading scenarios can follow a similar procedure as the scale-in part in Algorithm 1. In these cases, the process of determining whether a VNFI should be shut down can be skipped in the algorithm (lines 21~27), since the faulty and to-be-upgraded VNFIs can be directly treated as VNFIs needing to be shut down.

In the Scale Out scenario, we begin by traversing all overloaded VNFIs to identify the flows passing through them (line 2). For each flow, we compute the migration and routing costs based on the potential paths within P_k .⁸ We then select the path with the lowest cost, denoted as p_{min} , as the candidate migration scheme for this flow (lines 3~8). Subsequently, we sort the flows in ascending order based on a weighted sum of their migration cost M_k , routing cost T_k , and the reciprocal of their rate R_k (line 10). The parameters ω_1 and ω_2 are real numbers used to adjust the relative influence of migration and routing time costs against the flow rate, striking a balance in their impact on sorting results.

Based on the sorting results, we prioritize flows with the smallest weighted sum when selecting flows for migration. We begin by checking whether migrating a flow to the target path p_{min} might lead to buffer overflow or resource capacity constraint violations (line 12). If no violations are expected, we calculate the load on the overloaded VNFI ψ_i after migrating the current flow (line 13). If the current VNFI's load is less than ψ_{up} , we halt the selection of flows for migration (lines 14~17). This heuristic aims to minimize migration and routing costs while meeting the load requirements of overloaded VNFIs.

⁷ The non-linear optimization problems in problems Eqs. (4) and (5) can be transformed into linear optimization problems using common linearization techniques, enabling Gurobi to solve them.

⁸ P_k can be obtained through various methods, such as using shortest path algorithms like Dijkstra [25] or Bellman-Ford [26] to find the k-shortest paths (with path transmission delay as the link weight).

Algorithm 1 Heuristic Algorithm for Scale Out/In

Require: $K, P_k, R_k, DT_k, I_{over}$ or $I_{under}, \psi_{up}, \psi_{low}$
Ensure: Migrated Flow: K_{mig}

```

1: for each VNFI  $i \in I_{over}$  or  $I_{under}$  do
2:   Get flow set  $K_i$  pass through  $i$ ;
3:   for each flow  $k \in K_i$  do
4:     for each path  $p \in P_k$  do
5:       Calculate  $M_k$  and  $T_k$ ;
6:     end for
7:     Determine the  $p_{min}$  with minimal cost for  $k$ ;
8:   end for
9:   if Current scenario is Scale Out then
10:    Sort  $K_i$  by  $\omega_1(M_k + T_k) + \omega_2 \frac{1}{R_k}$  ascending;
11:    for each flow  $k \in K_i$  do
12:      if Path  $p_{min}$  is feasible then
13:        Calculate  $\psi_i$  supposing flow  $k$  migrated;
14:        if  $\psi_i < \psi_{up}$  then
15:          Append flow  $k$  in  $K_{mig}$ ;
16:          break;
17:        end if
18:      end if
19:    end for
20:  end if
21:  if Current scenario is Scale In then
22:    Calculate the benefit  $benf_i$  of shutting down  $i$ ;
23:    if  $\sum_{k \in K_i} M_k < benf_i$  then
24:      Append  $i$  in final shutdown VNFI set  $G_i$ ;
25:    end if
26:  end if
27: end for
28: for each VNFI  $i$  in  $G_i$  do
29:   Sort  $K_i$  by  $M_k + T_k$  ascending;
30:   for each flow  $k \in K_i$  do
31:     if Path  $p_{min}$  is infeasible then
32:       Find backup path for flow  $k$ ;
33:     end if
34:     Append flow  $k$  in  $K_{mig}$ ;
35:   end for
36: end for

```

In the Scale In scenario, we also need to compute the benefit $benf_i$ of shutting down the VNFI (line 22). The benefit of shutting down VNFIs can be assessed based on the economic cost savings for renting cloud resources. If $benf_i$ surpasses the total migration cost for flows passing through VNFI $i \in I_{under}$, the current underloaded VNFI i will be added to the shutdown VNFI set G_i (lines 23~25). For each VNFI $i \in G_i$, we employ a similar approach as in the scale out scenario to handle flows awaiting migration (lines 28~36).

Throughout the migration process, if the path with the minimum costs becomes infeasible due to resource capacity or other constraints, we search for an alternative path with minimal costs (provided it adheres to the relevant constraints) for the flow by examining its candidate feasible paths (lines 31~33). This ensures that flows can still be accommodated post-migration. Finally, all flows passing through VNFIs in G_i will be migrated out, and the corresponding VNFIs can be decommissioned.

5.2. Load balance solution

Balancing the optimization objectives outlined in Eq. (6), it becomes challenging to design a heuristic algorithm capable of yielding near-optimal flow migration schemes. While we can achieve an optimal migration scheme by solving the problems defined in Eq. (6), the

computational efficiency may be inadequate. To ensure both the optimality of flow migration schemes and computational efficiency, we initially relax the original ILP problem into a Linear Programming (LP) problem to obtain a floating solution (where one flow may be routed on more than one path). Subsequently, we employ a Two-Phase rounding algorithm, based on the approach outlined in [27], to derive the final flow migration scheme (where one flow can only be routed on one path). The workflow of the two-phase rounding algorithm is elucidated in Algorithm 2.

Algorithm 2 Two-Phase Rounding Algorithm

Require: $K, P_k, R_k, DT_k, B_l, Q_n, C_l^n, \psi_{low}$

Ensure: $\bar{m}_{p',p}^k$

- 1: Solve the relaxed ILP problem defined in Eq. (6) to obtain fractional results $\tilde{m}_{p',p}^k$;
 - 2: Obtain V, K', P' and P_f from $\tilde{m}_{p',p}^k$, set $m = \tilde{m}_{p',p}^k$;
 - 3: **while** $V \neq \emptyset$ **do**
 - 4: **if** $|V| > |K'| + |P'|$ (Phase 1) **then**
 - 5: Obtain nonzero vector r based on Eq. (8) and (9);
 - 6: Calculate α and β based on r ;
 - 7: With probability $\frac{\beta}{\alpha+\beta}$, set $m_{p',p}^{k,(h+1)} = m_{p',p}^{k,(h)} + \alpha \cdot r, \forall (k, p) \in V$;
 - 8: With probability $\frac{\alpha}{\alpha+\beta}$, set $m_{p',p}^{k,(h+1)} = m_{p',p}^{k,(h)} - \beta \cdot r, \forall (k, p) \in V$;
 - 9: Remove (k, p) from V if $m_{p',p}^k$ is integer;
 - 10: **end if**
 - 11: **if** $|V| \leq |K'| + |P'|$ (Phase 2) **then**
 - 12: Construct bipartite graph $G = (K', P_f, E)$;
 - 13: Find even cycle or maximal path in G , and partition the edges into two matchings $\mathcal{M}_1$ and $\mathcal{M}_2$;
 - 14: Define α, β based on Eq. (10) and (11)
 - 15: With probability $\frac{\beta}{\alpha+\beta}$, set $m_{p',p}^{k,(h+1)} = m_{p',p}^{k,(h)} + \alpha, \forall (k, p) \in \mathcal{M}_1$, and $m_{p',p}^{k,(h+1)} = m_{p',p}^{k,(h)} - \alpha, \forall (k, p) \in \mathcal{M}_2$;
 - 16: With probability $\frac{\alpha}{\alpha+\beta}$, set $m_{p',p}^{k,(h+1)} = m_{p',p}^{k,(h)} - \beta, \forall (k, p) \in \mathcal{M}_1$, and $m_{p',p}^{k,(h+1)} = m_{p',p}^{k,(h)} + \beta, \forall (k, p) \in \mathcal{M}_2$;
 - 17: Remove (k, p) from V if $m_{p',p}^k$ is integer;
 - 18: **end if**
 - 19: **end while**
 - 20: Set $\bar{m}_{p',p}^k = m$;
-

Before executing the rounding algorithm, the integer constraint for $m_{p',p}^k$ can be relaxed as follows:

$$m_{p',p}^k \in (0, 1), \forall k \in K, p \in P_k \quad (7)$$

The algorithm can efficiently solve the relaxed problem using existing linear programming solvers (e.g., Gurobi) in polynomial time (lines 1~2). The results may include fractional values (denoted as $\tilde{m}_{p',p}^k$). The Two-Phase rounding algorithm is tasked with rounding $\tilde{m}_{p',p}^k$ to an integral result, represented as $\bar{m}_{p',p}^k$. Both **Phase 1** and **Phase 2** are divided into several iterations. We use $m_{p',p}^{k,(h)}$ to denote the value of $m_{p',p}^k$ after iteration h . During each iteration of **Phase 1** and **Phase 2**, we refer to a flow k as a *floating flow* if it is currently routed on more than one path. A path p is also considered a *floating path* if it currently has at least one floating flow routed through it. If a path p has at least one floating flow routed through it, it is classified as a *nonsingleton floating path*. Let K', P_f , and P' represent the current sets of floating flows, floating paths, and nonsingleton floating paths, respectively. The set of currently unrounded “flow-path” pairs is denoted as V , defined as $V = \{(k, p) : m_{p',p}^{k,(h)} \in (0, 1)\}$.

In general, the Two-Phase rounding algorithm starts from **Phase 1** and goes into **Phase 2**. The specific phase can be judged by comparing the number of floating flows $|K'|$, number of nonsingleton paths $|P'|$ and number of unrounded pairs $|V|$. If $|V| > |K'| + |P'|$, the current

iteration is in **Phase 1** (line 4). Once $|V| \leq |K'| + |P'|$, the algorithm moves to **Phase 2** and continues to iterate (line 11).

During the iterations in **Phase 1**, we need to consider the following linear system:

$$\sum_{p \in P_f} m_{p',p}^{k,(h)} = 1, \forall k \in K' \quad (8)$$

$$\sum_{k \in K'} m_{p',p}^k \cdot R_k = \sum_{k \in K'} m_{p',p}^{k,(h)} \cdot R_k, \forall p \in P' \quad (9)$$

We represent the linear system described above as $Am = b$. This linear system is canonical ($m_{p',p}^{k,(h)} \in (0, 1)$) and underdetermined ($|V| > |K'| + |P'|$, meaning the number of variables exceeds the number of constraints in the linear system). It is easy (in polynomial time) to find a nonzero vector r such that $Ar = 0$ and strictly positive values α and β satisfying the following conditions: (1) all entries of $m + \alpha \cdot r$ and $m - \beta \cdot r$ fall within the $[0, 1]$ range, and (2) at least one entry of both $m + \alpha \cdot r$ and $m - \beta \cdot r$ is in the set $\{0, 1\}$. Once r, α , and β are determined, the algorithm updates $m_{p',p}^{k,(h+1)}$ based on the steps outlined in Algorithm 2, lines 5~9.

As the number of iterations increases in **Phase 1**, the rounding process ensures that at least one unrounded pair $(k, p) \in V$ becomes an integer. Consequently, the number of unrounded pairs, floating flows, and nonsingleton paths will decrease. When $|V| \leq |K'| + |P'|$ is reached, the rounding algorithm will transition to **Phase 2**. Otherwise, it will remain in **Phase 1** and round up all fractional $\bar{m}_{p',p}^k$.

In the **Phase 2** iteration process, the bipartite dependent-rounding algorithm [28] can be utilized. This algorithm involves constructing a bipartite graph $G = (K', P_f, E)$, where the edge set is formed by connecting flow $k \in K'$ and path $p \in P_f$ if $(k, p) \in V$. Subsequently, the algorithm selects an even cycle or maximal path within G and partitions the edges into two distinct matchings $\mathcal{M}_1$ and $\mathcal{M}_2$ (lines 12~13). Two positive values, α and β , are defined to satisfy the following conditions:

$$\alpha = \min\{\kappa > 0 : ((\exists (k, p) \in \mathcal{M}_1 : m_{p',p}^{k,(h)} + \kappa = 1) \quad (10)$$

$$\vee (\exists (k, p) \in \mathcal{M}_2 : m_{p',p}^{k,(h)} - \kappa = 0))\}$$

$$\beta = \min\{\kappa > 0 : ((\exists (k, p) \in \mathcal{M}_1 : m_{p',p}^{k,(h)} - \kappa = 0) \quad (11)$$

$$\vee (\exists (k, p) \in \mathcal{M}_2 : m_{p',p}^{k,(h)} + \kappa = 1))\}$$

Subsequently, we proceed with the randomized rounding steps outlined in Algorithm 2 lines 15~16, which ensures that at least one variable is rounded to an integer. Once all fractional results $\tilde{m}_{p',p}^k$ have been rounded to integers, the algorithm can generate the final flow migration schemes based on $\bar{m}_{p',p}^k$ (line 20).

5.3. Algorithm performance analysis

Due to the difficulty in quantifying the proportion of flows that may experience resource contention during migration (the higher the proportion of flows experiencing resource contention, the greater the gap between our heuristic algorithm and the optimal results), analyzing the performance bounds of scale out/in solutions accurately is challenging. Therefore, we primarily assess their performance through simulation experiments. In the case of load balance solutions, we can analyze the competitive ratio concerning optimal results and assess the situation of constraint violations, as outlined below:

Theorem 1. $E[\bar{\Psi}]$ is no larger than $\Psi^* + \frac{|P| \cdot \max_{k \in K} R_k}{\min_{i \in [I]} C_i^n}$, where Ψ^* is the offline optimal VNFI load

The detailed proof is shown in the supplementary material (Appendix F).

Theorem 2. For each flow k , $E[\sum_{k \in [K]} \bar{M}_k + \bar{T}_k]$ is no larger than $\sum_{k \in [K]} M_k^* + T_k^*$, where M_k^* and T_k^* are the offline optimal flow migration cost and flow routing cost after migration respectively

The detailed proof is shown in the supplementary material (Appendix G).

Theorem 3. For each floating flow, we have with probability 1 that:

$$\sum_{p \in P_f} \bar{m}_{p',p}^k = 1$$

The detailed proof is shown in the supplementary material (Appendix H).

Theorem 4. For each packet buffer of cloud node belongs to floating path, we have with probability 1 that:

$$\sum_{i \in I_n} \sum_{k \in [K]} S_i^{k,n} \leq Q_n + |P| \cdot \max(0, \zeta_f \gamma_i^f - d_i) \max_{k \in [K]} R_k$$

The detailed proof is shown in the supplementary material (Appendix I). This means that the capacity constraint of each packet buffer is violated by at most $|P| \cdot \max(0, \zeta_f \gamma_i^f - d_i) \max_{k \in [K]} R_k$.

Theorem 5. For each link belongs to floating path, we have with probability 1 that:

$$\sum_{k \in K} \sum_{p \in P} \sum_{i \in P} x_{p,i}^l R_k \gamma_i^f \bar{m}_{p',p}^k \leq B_l + |P| \cdot \max_{k \in K} R_k$$

The detailed proof is shown in the supplementary material (Appendix J). This means that the capacity constraint of each link is violated by at most $|P| \cdot \max_{k \in K} R_k$.

5.4. Algorithm complexity analysis

In CFM, the computation complexity of heuristic algorithm for **Scale Out** and **Scale In** scenarios mainly depends on the number of overload (or underload) VNFI and user flows. For **Scale Out** scenario, suppose the number of overload VNFIs is $|I_{over}|$ and the number of user flow is $|K|$, and we can also determine the size of feasible routing path set $|P_k|$ for each user flow $k \in K$ by pre-calculating. For each overload VNFI $i \in I_{over}$, if the number of flows passing through VNFI i is $|K_i|$, the complexity of determining the candidate migration path with minimal cost for all flows $k \in K_i$ does not exceed $O(|K_i| |P_k|)$ (where $|K_i| < |K|$). Considering the complexity of determining the final migrated flows does not exceed $O(|K_i|)$, the total complexity of **Scale Out** scenario is $O(|I_{over}|(|K_i| |P_k| + |K_i|))$. For **Scale In** scenario, since the workflow is similar to the **Scale Out** scenario, the complexity is also no more than $O(|I_{under}|(|K_i| |P_k| + |K_i|))$. As for the Two-Phase Rounding Algorithm (for **Load Balance** scenario), the computation complexity mainly depends on both the efficiency for solving relaxed ILP optimization problem defined in Eq. (6), and the size of $\bar{m}_{p',p}^k$ required to be rounded, The computation complexity of rounding part does not exceed $O(|V|)$ ($|V|$ is the size of unrounded “flow-path” pair).

6. Performance evaluation

6.1. Simulation setup

6.1.1. Topology

We select two network topologies, Cernet (41 nodes, 59 links) and Uninet (49 nodes, 84 links), from TopologyZoo [29]. In each topology, we randomly designate 5 to 10 nodes as cloud nodes, while considering the remaining nodes as switch nodes. These cloud nodes are further categorized into two types: *resource-sufficient* nodes and *resource-limited* nodes. *Resource-sufficient* nodes could allocate more computing resources for the VNFIs compared to *resource-limited* nodes. This difference affects the processing capacity of the VNFIs running on these two types of cloud nodes. Additionally, to simulate significant latency differences between different links in a geo-distributed cloud network environment, the forwarding delay of links in the topology is randomly set within [3, 70] ms. Meanwhile, the bandwidth of the links is randomly set within [1, 10] Gb/s.

6.1.2. VNF and SFC

We consider 10 different types of VNFs, consistent with previous works [30,31]. The VNF state migration delay for a single flow is randomly set in [1, 50] ms, following [7,15]. For VNFIs running on *resource-sufficient* cloud nodes, their processing capacity is randomly set in [0.9, 1.2] GB/s. Conversely, for VNFIs running on *resource-limited* cloud nodes, the processing capacity is randomly set in [0.4, 0.8] GB/s. By configuring the processing capacity of cloud nodes in this manner, it aligns closely with the capacity magnitude of the link bandwidth. This helps ensure that flows processed by VNFs can be forwarded promptly without introducing additional delay. The processing delay of VNFs is set in [1, 5] ms, which is similar to [32].

We assign a random number of VNFIs per VNF type, varying from 1 to 5. Some VNFIs remain in an inactive state and could be activated when the active VNFIs could not handle all flows. A SFC consisted of 2 to 5 different VNFs, and we generate 5 different types of SFCs for simulation. We also set different expected service completion delays for SFCs. Delay-sensitive SFCs have expected service completion delays ranging from 80 to 200 ms, while delay-tolerant SFCs have delays extended from 300 to 500 ms. Similar to [9], the average delay for redirecting packets for each flow is $\Phi = 0.5$ ms, and the delay for installing a single flow entry on a switch node is $\Gamma = 5$ ms.

6.1.3. Performance metric

We primarily assess following key metrics:

(a) **VNFI Load Ratio:** We calculate the VNFI load ratio using the formula: $\max_{i \in I} \frac{l_i}{C_i^n}$ where l_i represents the load of VNFI i , and C_i^n is the computing capacity of VNFI i on node n . This metric can help to understand how well the VNFI loads are balanced across the cloud nodes.

(b) **Flow Migration Cost and Flow Routing Cost After Migration:** To ensure a fair comparison between flow migration cost and flow routing cost after migration, we unify the evaluation metric for both as time, measured in milliseconds. Originally, the unit of flow migration cost is at the millisecond level. By setting the coefficients $\alpha = 0$ and $\beta = 1$ in Eq. (3), the unit of flow routing cost after migration is also at the millisecond level. This allows us to directly compare the time-based costs. These metrics can help to assess the overall efficiency and performance of the migration process and post-migration routing.

(c) **Packet Buffer Usage:** Packet buffer usage is evaluated by summing the buffered packets across all cloud nodes: $\sum_{n \in N} \sum_{i \in I_n} \sum_{k \in K} S_i^{k,n}$. This metric can help to assess the efficiency of packet buffering during flow migration.

These metrics collectively provide insights into the load balancing, cost-effectiveness, and resource utilization of our proposed approach.

6.1.4. Flow pattern

(a) **Single-set Flows Pattern:** We maintain flow size and variation settings as described in [9]. Flow sizes follow an 80/20 distribution, with 20% being large flows that potentially account for over 80% of total flows. The average flow size is 20 kB. Small flows fluctuate by up to 10 kB, while large flows vary between 10 kB and 1 MB. Different flow migration events can be triggered by adjusting the flow size. We generate several sets of flows for simulation, each with a fixed number of flows ranging from 10,000 to 50,000. We use the method from our prior work [33], which constructs a layered graph based on the required SFC and applies Dijkstra's algorithm, to find the initial routing path of flow.

(b) **Continuous Flows Pattern:** The flow traces are constructed based on real campus network traffic data. Each record with the same source-destination IP pair is considered a single flow. We calculate the number of flows arriving within each time unit, with one time unit corresponding to ten minutes. Fig. 5 displays the normalized flow arrival rate, where the x-axis shows time units, and the y-axis shows the normalized number of flows. We scale the normalized values by up to 200 times to simulate actual flow arrivals. Each flow has a randomly generated lifetime, and when its lifetime ends, the flow exits the network.

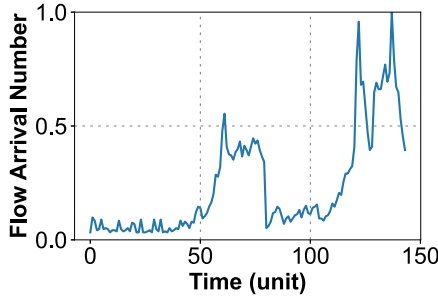


Fig. 5. Campus flow trace.

6.1.5. Benchmarks

For single-set flow pattern evaluation, we primarily compare CFM with the following state-of-the-art works:

(a) Scale Out: For the Scale Out scenario, we use the algorithm proposed in [8], referred to as *HFM*. In *HFM*, a trade-off is made between reducing the maximum node load and minimizing reconfiguration overhead. To make migration decisions, *HFM* selects one SFC for migration from the overloaded NFV node to a new target NFV node until the load of the overloaded NFV node falls below the expected load. To adapt *HFM* for the Scale Out scenario, we treat the NFV nodes in *HFM* as VNFI and only migrate enough flows that utilize the chosen SFC to ensure that the VNFI load is reduced to below the expected load (the “expected load” in *HFM* is set to ψ_{up}).

(b) Scale In: For the Scale In scenario, we extend the heuristic algorithm proposed in a state-of-the-art work [34], which minimizes the delay of affected flows and the maximum network load during scale in operations. In our extended algorithm, we consider the malfunctioning physical machines in [34] as the VNFIs that need to be shut down during the scale in operation. The selection of new routing paths for migrated flows in our algorithm is based on the same criteria as in [34], which focuses on minimizing the impact of flow routing delay and VNF inherent load after migration. We refer to this extended algorithm as *MSHOR*.

(c) Load Balance: For the Load Balance scenario, we compare our approach with a state-of-the-art work [9], denoted as *GRSS*. The primary objective of *GRSS* is to minimize the maximum VNFI load while ensuring that the update delay does not exceed a specified threshold denoted as T , and that the flow routing delay after migration remains within acceptable bounds. In our simulation, we implement the flow migration optimization based on the original algorithm workflow from [9], and we set the update delay constraint T to be no more than 7 s, which is consistent with the value set in the simulation experiments by [9]. For CFM, according to our testing results, the combined flow migration cost and post-migration routing cost typically fall within the range of 30,000 to 120,000. To sustain a reasonably balanced optimization outcome, we have configured δ_1 to 100,000 and δ_2 to 1 in Eq. (6).

In addition to the comparisons mentioned above, we also replicated the approach outlined in another state-of-the-art work [35] for optimizing SFC migration. The core idea behind the strategy proposed by [35] is to balance resource loads (including cloud nodes and links) after migration while minimizing additional bandwidth costs. We implemented a comparative algorithm based on this strategy, denoted as *AMA*, which can be compared against CFM in all three migration scenarios. In the implemented *AMA* algorithm, we only consider migrating flows from VNFI instead of migrating the entire VNFI. Moreover, since *AMA* iterates through all flows to find feasible migration schemes (which may preserve the original routing path), we configure the algorithm to stop traversing flows for the Scale Out scenario once enough outgoing flows are migrated to reduce the overloaded VNFI to the specified threshold. For the Scale In scenario, we require that all flows passing through the

identified VNFI for closure must be migrated out (i.e., paths containing this VNFI are not considered as candidate paths after migration).

For continuous flow pattern evaluation, we combine *HFM* for handling Scale Out events, *MSHOR* for handling Scale In events, and *GRSS* for handling load balance events into a unified algorithm, denoted as *CMP*, for comparison with CFM. Additionally, after adjusting *AMA*, it can independently handle all flow migration events and thus be compared with CFM.

We also employ the Gurobi solver to tackle the ILP optimization problems defined in Section 4.2. The theoretical optimal results obtained using this method are utilized to assess the performance gap between our algorithms and the optimal solutions in various scenarios. This solving method is referred to as *ILP*. As for the simulation experiments on VNF Failure Recovery and VNF Upgrade scenarios, since their optimization objectives are the same as those of the Scale In scenario, their optimization results can be referred to the subsequent simulation experiments of the Scale In scenario.

6.2. Simulation results

6.2.1. Single-set flows pattern results

(a) Scale Out: In the Scale out scenario, significant improvements in optimization for flow routing cost (SLA Cost) after migration are observed when comparing CFM with *HFM* and *AMA* in different network topologies, as illustrated in Fig. 6. Specifically, CFM can reduce the SLA costs by up to 53.5% (average of 38.6%) in the Uninet topology and by up to 43.2% (average of 29.8%) in the Cernet topology compared to *HFM*. Furthermore, compared to *AMA*, CFM achieves reductions in SLA costs by up to 45.4% (average of 33.8%) in the Uninet topology and by up to 24% (average of 16.2%) in the Cernet topology. This improvement stems from CFM’s ability to optimize flow routing costs based on flow delay requirements, whereas *HFM* and *AMA* do not consider this aspect.⁹

Regarding flow migration cost (Migration Cost), CFM and *HFM* achieve similar optimization effects. Specifically, for scenarios with a low number of flows (e.g., $\leq 3 \times 10^4$ in the left plots of Fig. 6(a) and (b)), *HFM* tends to achieve lower flow migration costs. This is because *HFM* prioritizes minimizing state transfer costs and the number of extra virtual links for flow rerouting. Consequently, *HFM* minimizes the differences between routing paths before and after migration, resulting in lower routing rules update costs. However, this approach also leads to less-than-ideal flow routing costs after migration.

In contrast, as the number of flows increases (e.g., $\geq 3 \times 10^4$ in the left plots of Fig. 6(a) and (b)), more flows need to be migrated out to reduce the load on overloaded VNFIs. *HFM*, not considering flow size in migration decisions, may migrate out a larger number of smaller flows until the load falls below the specified threshold. This results in higher migration costs. Therefore, CFM becomes more effective in optimizing migration costs in scenarios with a higher number of flows.

However, *AMA* incurs the highest migration costs when selecting migration schemes. Compared to CFM, it can increase migration costs by up to 19.8% (average of 16.8%) in the Uninet topology and up to 19.7% (average of 15.6%) in the Cernet topology. This is because *AMA* starts by migrating flows with the least resource consumption (i.e., lowest flow rates), leading to migrating more flows to meet the load requirements of overloaded VNFIs, thus resulting in higher migration costs.

Comparing CFM and the optimal results obtained by ILP, CFM has higher migration and SLA costs. The migration cost gap ranges from

⁹ It is interesting to note that *AMA* achieves lower SLA costs than *HFM*. This is because *HFM* selects post-migration routing paths similar to the original ones (since it aims to optimize migration cost), resulting in minimal reduction in SLA cost. In contrast, *AMA* prioritizes balancing resource loads in post-migration routing, potentially migrating some flows to paths with lower delays, thereby optimizing SLA costs to some extent.

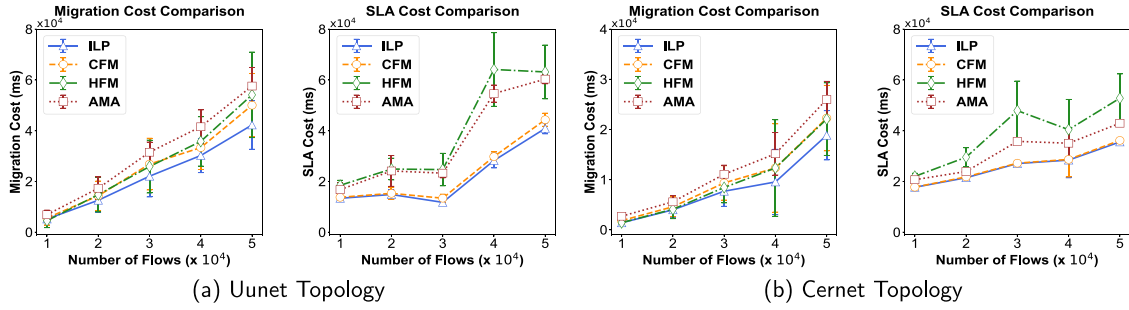


Fig. 6. Migration-related cost vs. number of flows on different topologies in scale out scenario.

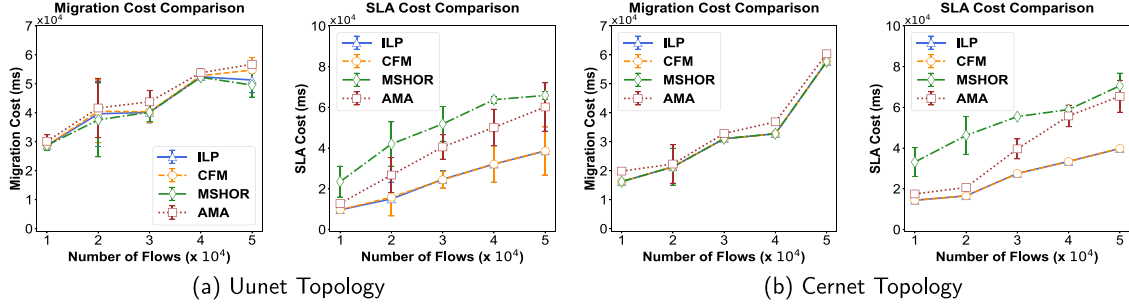


Fig. 7. Migration-related cost vs. number of flows on different topologies in scale in scenario.

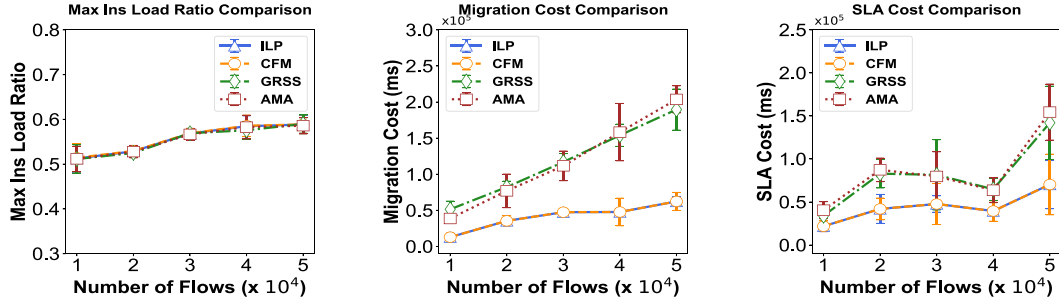


Fig. 8. Migration-related cost vs. number of flows on uunet topology in load balance scenario.

−7.6% to +12.1% (negative values indicate lower migration costs) with a maximum gap of 21.6% (average of 14.3%) for migration costs and a maximum gap of 13.9% (average of 6.8%) for SLA costs in the Uunet topology. In the Cernet topology, the gap for migration and SLA costs can reach a maximum of 32% (average of 23.2%) and 1.7% (average of 1.1%), respectively.

Overall, CFM demonstrates improved optimization for SLA cost after migration, especially in scenarios with a high number of flows, compared to HFM and AMA. However, it may have slightly higher migration costs compared to HFM in low-flow scenarios. Additionally, there is still room for improvement in achieving results closer to the optimal solutions obtained through ILP.

(b) Scale In: In the Scale In scenario, the simulation results are shown in Fig. 7. CFM consistently achieves lower SLA costs compared to MSHOR and AMA in both network topologies. The reason aligns with that observed in the Scale Out scenario: MSHOR focuses on minimizing the influence of flow routing delays before and after migration, while AMA tries to minimize the standard deviation among resource loads. However, neither of them take into account the specific delay requirements of flows. Based on the results, we observe that: In the Uunet topology, CFM reduces SLA costs by up to 62.3% (average of 53.1%) compared to MSHOR, and by up to 40.9% (average of 35.4%) compared to AMA. Similarly, in the Cernet topology, CFM reduces SLA costs by up to 64.2% (average of 51.6%) compared to MSHOR, and by up to 40.1% (average of 29.5%) compared to AMA.

Regarding migration costs, MSHOR, which prioritizes minimizing delay impact, tends to generate routing paths that exhibit minimal differences before and after migration. Consequently, this leads to lower routing rules update costs. Therefore, MSHOR occasionally achieves lower migration costs than CFM, with a maximum reduction of 10.4% in migration cost observed in the Uunet topology. However, in most cases, both CFM and MSHOR produce migration costs that are quite similar. The gap in migration cost between CFM and MSHOR falls within the range of −3.9% to 0% (where negative values indicate lower migration costs). As for AMA, due to similar reasons as in the Scale Out scenario, the migration costs are higher than those of CFM, with a maximum increase of 7.9% (average of 4.2%) in the Uunet topology and 11.2% (average of 8.7%) in the Cernet topology.

Analyzing the gap between CFM and ILP, the results indicate that in the Uunet topology, the maximum gap is 6.7% (with an average of 2%) for migration costs and 5.4% (with an average of 1%) for SLA costs. In the Cernet topology, there is almost no discernible gap for either migration cost or SLA cost between CFM and ILP.

In summary, CFM consistently achieves superior reductions in SLA costs compared to MSHOR and AMA by considering flow delay requirements. While MSHOR may occasionally produce slightly lower migration costs, CFM maintains competitive performance. Furthermore, in the Cernet topology, CFM closely aligns with the optimal solutions obtained through ILP for both migration cost and SLA cost.

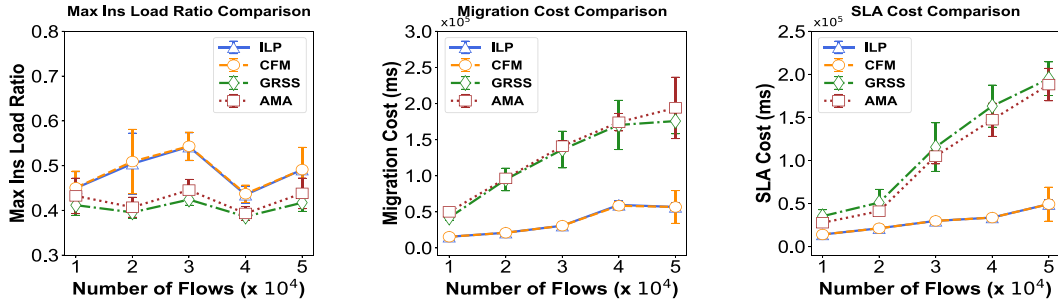


Fig. 9. Migration-related cost vs. number of flows on cernet topology in load balance scenario.

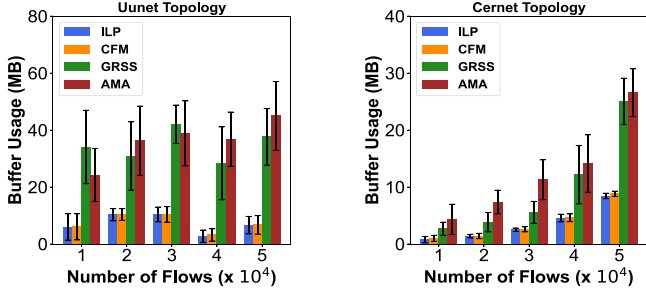


Fig. 10. Packet buffer usage vs. number of flows in load balance scenario.

(c) Load Balance: In the Load Balance scenario, based on the results shown in Figs. 8 and 9, we can see that GRSS and AMA demonstrate better effective optimization of VNFI load ratios. GRSS can reduce the maximum VNFI load ratio by up to 1.7% (average of 0.6%), while AMA can reduce the maximum VNFI load ratio by up to 0.5% (average of 0.45%), when compared to CFM in the Uunet topology. In the Cernet topology, GRSS achieves a maximum reduction of 28.7% (average of 19.4%), while AMA achieves a maximum reduction of 25.1% (average of 14.9%) in VNFI load ratios compared to CFM. For migration cost, CFM can reduce migration costs by up to 75% (average of 65.5%) compared to GRSS, and by up to 69.8% (average of 63.5%) compared to AMA in the Uunet topology. In the Cernet topology, CFM achieves a maximum migration cost reduction of 77.8% (average of 70.3%) compared to GRSS, and a maximum migration cost reduction of 78.2% (average of 72.5%) compared to AMA. As for SLA cost, CFM can reduce SLA costs by up to 50.5% (average of 43.2%) compared to GRSS, and by up to 54.5% (average of 46.1%) compared to AMA in the Uunet topology. In the Cernet topology, CFM achieves a maximum SLA costs reduction of 79.4% (average of 69.4%) compared to GRSS, and a maximum SLA costs reduction of 77.1% (average of 64.1%) compared to AMA.

The divergence in optimization results is attributed to the different objectives pursued by each algorithm. While GRSS and AMA focus primarily on optimizing VNFI load ratios, CFM adopts a more comprehensive approach by jointly optimizing VNFI load, migration cost, and SLA cost. This comprehensive approach entails a tradeoff among these three cost components. Consequently, CFM can reduce migration and SLA costs by sacrificing some of the optimization gains achieved in VNFI load ratios. It is worth noting that GRSS and AMA, while effectively optimizing VNFI load ratios, may result in more flows being migrated, incurring higher migration costs. Specifically, AMA incurs the highest flow migration cost, as it starts migrating from flows with lower rates, leading to the migration of more flows. Furthermore, its greedy approach to selecting flow migration schemes results in inferior optimization of VNFI load ratios compared to GRSS (which uses an approximation algorithm to obtain flow migration schemes). Additionally, GRSS and AMA overlook the delay requirements of flows when optimizing routing delays, which may lead to higher SLA costs. In

contrast, CFM accounts for these delay requirements in its optimization strategy.

Due to the incorporation of the two-phase rounding algorithm, CFM effectively minimizes the gap in optimization performance compared to ILP. Theoretically, based on Theorem 1 and our experimental parameter settings, we can estimate that the maximum gap between CFM and ILP in terms of maximal VNFI load is approximately 12%. This estimation takes into account the scale of the path set, $|P|$, which was set to 2500 in our experiments. Encouragingly, the results displayed in Figs. 8 and 9 reveal that the actual gap is approximately 1%, a value well within the anticipated theoretical range.

Moreover, we can deduce that the expected sum of migration cost and SLA cost obtained by CFM does not exceed the optimal results of the relaxed LP problem, as stipulated by Theorem 2. Considering that the optimal outcomes achieved by ILP are generally not inferior to those of the LP problem, we can reasonably conclude that CFM can achieve migration and SLA costs similar to ILP. Empirically, our experimental results corroborate this inference, as the maximum gap between CFM and ILP in terms of the sum of migration cost and SLA cost is less than 1%.

Beyond these three primary metrics, we also conducted evaluations concerning packet buffer usage, as depicted in Fig. 10. CFM excels in minimizing buffer usage, since it considers buffer capacity constraints during optimization. As GRSS and AMA migrate more flows to achieve lower maximum VNFI load ratios, they inevitably result in higher buffer usage. In practical scenarios, excessive buffer usage can lead to increased packet redirection delays, ultimately affecting migration efficiency and service performance.

6.2.2. Continuous flows pattern results

In this pattern, we focus on assessing the fluctuations in the maximum VNFI instance load ratio, flow migration cost (Migration Cost), and flow routing cost (SLA Cost) over time, along with their probability distributions. Fig. 11 provides specific cost values, while Fig. 12 presents the probability distributions of these costs through CDF plots.

Observing the results, it is evident that CFM outperforms CMP and AMA in optimizing SLA cost, particularly when dealing with heavy flows, such as during time units 50 to 80. CFM also tends to achieve lower flow migration costs than CMP and AMA in most scenarios. Furthermore, for each performance metric, there is no significant gap between CFM and ILP. Over the entire time period, the CDF results demonstrate that CFM and ILP produce nearly identical optimization outcomes.

6.2.3. Scalability of CFM

Considering the absence of real-world settings for the NFW environment, such as the maximum number of concurrent flows or the actual count of VNFI, our evaluation focuses on assessing the potential trend in CFM's computational efficiency as the number of flows or VNFI increases. This assessment was conducted on the Uunet topology. Due to space constraints, we present the detailed results in the supplementary material (Appendix K).

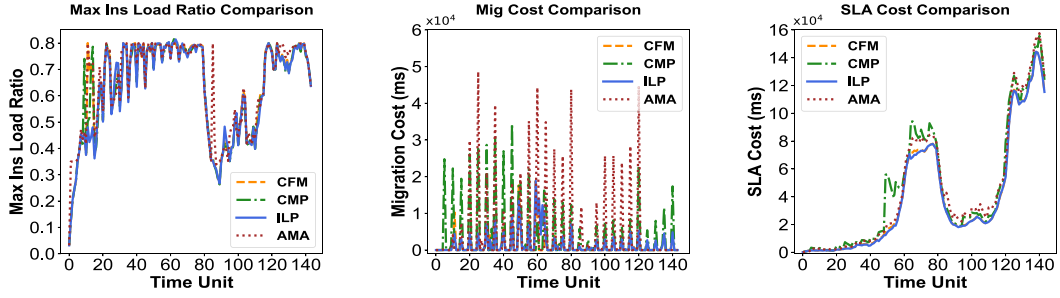


Fig. 11. Migration-related cost vs. time unit on Unnet topology.

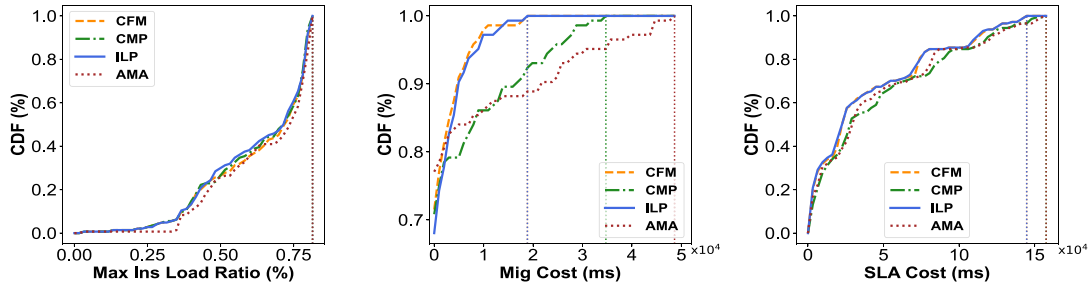


Fig. 12. CDF vs. migration-related cost on Unnet topology.

The results reveal that CFM can deliver improved optimization outcomes while maintaining similar solving efficiency compared to existing schemes. Specifically, in the Scale Out scenario, both CFM, HFM and AMA can generate flow migration schemes in under 60 s, whereas the ILP solution may require up to 150 s to produce optimal migration schemes. In the Scale In scenario, CFM, MSHOR and AMA can provide flow migration schemes in less than 15 s, whereas the ILP solution may take up to 80 s to generate optimal migration schemes. Moreover, in the Load Balance scenario, CFM can reduce computation time by at least half compared to the ILP and achieve a 10% to 27% reduction in computation time compared to GRSS, while increasing computation time by 5% to 32% compared to AMA.

6.2.4. Summary

CFM can achieve similar flow migration costs and maximum VNFI load ratios compared to existing works, while significantly reducing flow routing costs after migration. The key reason for this improvement is that CFM simultaneously optimizes these cost metrics during the migration process. Notably, CFM can further optimize the routing costs of flows based on their delay requirements, leading to enhanced overall performance.

7. Related works

7.1. Flow migration for single VNF

Several previous works have established the fundamental framework for achieving flow and state migration, recognizing its importance in NFV elasticity control. For instance, Split/Merge [23] and OpenNF [15] propose the use of a centralized control plane to buffer VNF states during migration. Additionally, efforts [18,36–38] have been made to enhance the performance of these approaches. For example, Gember-Jacobson et al. [18] improve OpenNF by introducing state and packet transfer directly in the data plane to enhance scalability. However, these works primarily focused on ensuring the safety and efficiency of state migration, often overlooking migration cost optimization.

As a complementary aspect, some research has concentrated on optimizing migration costs for individual VNFI. Sun et al. [7] tackle the challenge of flow selection for NFV elasticity control to minimize migration costs. And Sun et al. [39] propose solving the flow migration optimization problem using graph-aware deep reinforcement learning techniques. Nevertheless, all these prior works centered around migration for single VNFI and did not account for VNFI processing order constraints within SFCs. This limitation means that these solutions are not well-suited for optimizing migration costs in the context of SFCs.

7.2. Flow migration for SFC

Several works have explored SFC reconfiguration methods to address network service quality degradation stemming from flow variations. These methods encompass adjustments such as scaling the computing resources of VNFI, increasing link bandwidth, executing VNFI migration, and updating flow routing paths. Some of the works primarily concentrate on optimizing flow routing paths post-reconfiguration and neglect to optimize migration costs and maintain state consistency within VNFI. Furthermore, they have not delved into optimizing VNFI loads after reconfiguration. For instance, Liu et al. [40] propose a column generation-based algorithm aimed at optimizing the profit derived from served user flows and SFC deployment costs through SFC deployment and readjustment schemes. Toosi et al. [41] introduce a heuristic auto-scaling algorithm designed to minimize scaling costs while meeting end-to-end SFC delay requirements. Xue et al. [42] develop polynomial-time approximation algorithms to optimize schemes involving VNFI redeployment and flow routing path updates, with a focus on maximizing quality of service improvements.

There are also some works aimed at optimizing the overhead generated by the migration process. For example, to alleviate the load of overloaded VNFI, Zhang et al. [43] propose a heuristic algorithm for selecting flows to migrate, with the goal of minimizing the number of affected flows during migration. Eramo et al. [44] introduce a heuristic algorithm based on the Markov Decision Process to determine VNFI migration policies, aiming to balance energy cost savings and QoS degradation costs during migration. Qin et al. [45] propose a Markov

approximation based heuristic algorithm for selecting VNFI and flow migration schemes, aiming to balance between the SFC latency and the migration cost. Tokuda et al. [46] propose a deep reinforcement learning model for optimizing network slice reconfiguration costs, including the number of migrated VNFs and additional links. However, these works still neglect to optimize network performance after migration, such as VNFI load. Moreover, some works focus on optimizing resource load after migration. For instance, Feng et al. [35] propose two heuristic algorithms to determine VNFI and flow migration schemes. Their objective is to balance the load of cloud nodes and links after migration and minimize additional bandwidth costs incurred by migration. However, this approach overly emphasizes resource load optimization, and when using greedy algorithms to balance the load, the optimization effect may be limited (potentially performing worse than approximate algorithms).

Additionally, there are works that specifically target the optimization of flow migration costs within SFCs. For instance, Qu et al. [8] investigate dynamic flow migration problems for SFCs, aiming to meet the end-to-end delay requirements of flows. Their objective is to strike a balance between load balancing and reconfiguration overhead, leading to the development of a heuristic algorithm based on the redistribution of hop delay bounds. Fan et al. [9] propose a problem related to NFV-enabled network updates that prioritize satisfying delay requirements. They seek to achieve load balancing among VNF instances while optimizing the costs associated with SFC and routing updates. Their work introduces an approximation algorithm to determine optimized update schemes. Tu et al. [47] put forward a rounding-based algorithm for real-time SFC routing updates with the goal of optimizing VNF instance and link loads.

However, all above existing works tend to overlook the optimization of flow routing costs after migration based on flow-specific delay requirements. This oversight can potentially lead to reduced service performance for SFCs, especially in scenarios where computing resources on cloud nodes are limited. Moreover, these works predominantly focus on single-flow migration scenarios, such as load balancing. In contrast, our framework offers joint optimization of flow migration costs and flow routing costs after migration across various migration scenarios, including scale out, scale in, and load balance.

8. Conclusion

We present CFM, a cost-efficient flow migration framework designed to facilitate SFC elasticity scheduling across various scenarios. CFM incorporates a calculation method for assessing flow migration costs, formulates optimization objectives, and integrates constraints to tackle optimization problems. Our framework includes algorithmic solutions that deliver near-optimal flow migration schemes, demonstrated through extensive simulations to generate efficient schemes in a timely manner. Considering that the timing of flow migration operations also influences the optimization effect on network performance, in our future work, we will endeavor to design more rational decision-making schemes for determining the timing of flow migration. These schemes will be integrated with the flow migration optimization algorithms proposed in this paper to further enhance overall network performance. We also consider to implement our framework in open-source NFV and SDN environment to further evaluate its effectiveness.

CRedit authorship contribution statement

Weihan Chen: Formal analysis, Investigation, Methodology, Software, Validation, Writing – original draft. **Zhiliang Wang:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Han Zhang:** Conceptualization, Methodology, Writing – review & editing. **Xia Yin:** Project administration, Supervision, Writing – review & editing. **Xingang Shi:** Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Zhiliang Wang has patent Method, device, electronic device, and storage medium for flow migration based on service function chain pending to None. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at <https://doi.org/10.1016/j.comnet.2024.110496>.

References

- [1] R. Guerzoni, et al., Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper, in: SDN and OpenFlow World Congress, Vol. 1, 2012, pp. 5–7.
- [2] Q. Duan, N. Ansari, M. Toy, Software-defined network virtualization: An architectural framework for integrating SDN and NFV for service provisioning in future networks, *IEEE Netw.* 30 (5) (2016) 10–16.
- [3] P. Quinn, T. Nadeau, Problem statement for service function chaining, 2015, RFC 7498.
- [4] N. Toumi, O. Bernier, D.-E. Meddour, A. Ksentini, On cross-domain service function chain orchestration: An architectural framework, *Comput. Netw.* 187 (2021) 107806.
- [5] S. Yang, F. Li, S. Trajanovski, et al., Delay-aware virtual network function placement and routing in edge clouds, *IEEE Trans. Mob. Comput.* 20 (2) (2019) 445–459.
- [6] Amdocs developers, Amdocs NFV SD-WAN on microsoft azure. URL: query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWBlo5.
- [7] C. Sun, J. Bi, Z. Meng, T. Yang, X. Zhang, H. Hu, Enabling NFV elasticity control with optimized flow migration, *IEEE J. Sel. Areas Commun.* 36 (10) (2018) 2288–2303.
- [8] K. Qu, W. Zhuang, Q. Ye, et al., Dynamic flow migration for embedded services in SDN/NFV-enabled 5G core networks, *IEEE Trans. Commun.* 68 (4) (2020) 2394–2408.
- [9] X. Fan, H. Xu, H. Huang, X. Yang, Real-time update of joint SFC and routing in software defined networks, *IEEE/ACM Trans. Netw.* 29 (6) (2021) 2664–2677.
- [10] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, R. Boutaba, Elastic virtual network function placement, in: 2015 IEEE 4th International Conference on Cloud Networking, CloudNet, IEEE, 2015, pp. 255–260.
- [11] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, M. Zhang, Guaranteeing deadlines for inter-data center transfers, *IEEE/ACM Trans. Netw.* 25 (1) (2016) 579–595.
- [12] R. Li, Z. Zhou, X. Chen, Q. Ling, Resource price-aware offloading for edge-cloud collaboration: A two-timescale online control approach, *IEEE Trans. Cloud Comput.* (2019).
- [13] A. Nygren, B. Pfaff, B. Lantaz, B. Heller, C. Barker, C. Beckmann, et al., OpenFlow Switch Specification 1.5. 0, Tech. Rep. ONF TS-020, Open Netw. Found., 2014.
- [14] S. Sen, D. Shue, S. Ihm, M.J. Freedman, Scalable, optimal flow routing in datacenters via local link balancing, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, 2013, pp. 151–162.
- [15] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, A. Akella, OpenNF: Enabling innovation in network function control, *ACM SIGCOMM Comput. Commun. Rev.* 44 (4) (2014) 163–174.
- [16] Kubernetes developers, Kubernetes. URL: <https://kubernetes.io/>.
- [17] RYU developers, RYU. URL: <https://ryu-sdn.org/>.
- [18] A. Gember-Jacobson, A. Akella, Improving the safety, scalability, and efficiency of network function state transfers, in: Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, 2015, pp. 43–48.
- [19] L. Wu, S.K. Garg, R. Buyya, SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments, in: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2011, pp. 195–204.
- [20] C. Barnhart, C.A. Hane, P.H. Vance, Integer multicommodity flow problems, in: International Conference on Integer Programming and Combinatorial Optimization, Springer, 1996, pp. 58–71.

- [21] S. Even, A. Itai, A. Shamir, On the complexity of time table and multi-commodity flow problems, in: 16th Annual Symposium on Foundations of Computer Science, IEEE, 1975, pp. 184–193.
- [22] E. Mokotoff, Parallel machine scheduling problems: A survey, *Asia-Pac. J. Oper. Res.* 18 (2) (2001) 193.
- [23] S. Rajagopalan, D. Williams, H. Jamjoom, A. Warfield, {Split/merge}: System support for elastic execution in virtual middleboxes, in: 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 13, 2013, pp. 227–240.
- [24] Gurobi developers, Gurobi. URL: <https://www.gurobi.com/>.
- [25] E.W. Dijkstra, et al., A note on two problems in connexion with graphs, *Numer. Math.* 1 (1) (1959) 269–271.
- [26] R. Bellman, On a routing problem, *Q. Appl. Math.* 16 (1) (1958) 87–90.
- [27] V.A. Kumar, M.V. Marathe, S. Parthasarathy, A. Srinivasan, A unified approach to scheduling on unrelated parallel machines, *J. ACM* 56 (5) (2009) 1–31.
- [28] R. Gandhi, S. Khuller, S. Parthasarathy, A. Srinivasan, Dependent rounding and its applications to approximation algorithms, *J. ACM* 53 (3) (2006) 324–360.
- [29] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, *IEEE J. Sel. Areas Commun.* 29 (9) (2011) 1765–1775.
- [30] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, Making middleboxes someone else's problem: Network processing as a cloud service, *ACM SIGCOMM Comput. Commun. Rev.* 42 (4) (2012) 13–24.
- [31] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, S. Shenker, E2: A framework for NFV applications, in: *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 121–136.
- [32] J. Pei, P. Hong, K. Xue, D. Li, Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system, *IEEE Trans. Parallel Distrib. Syst.* 30 (10) (2018) 2179–2192.
- [33] W. Chen, Z. Wang, H. Zhang, X. Yin, X. Shi, Cost-efficient dynamic service function chain embedding in edge clouds, in: *2021 17th International Conference on Network and Service Management, CNSM, IEEE*, 2021, pp. 310–318.
- [34] B. Li, B. Cheng, X. Liu, M. Wang, Y. Yue, J. Chen, Joint resource optimization and delay-aware virtual network function migration in data center networks, *IEEE Trans. Netw. Serv. Manage.* 18 (3) (2021) 2960–2974.
- [35] H. Feng, Z. Shu, T. Taleb, Y. Wang, Z. Liu, An aggressive migration strategy for service function chaining in the core cloud, *IEEE Trans. Netw. Serv. Manage.* (2022).
- [36] B. Kothandaraman, M. Du, P. Sköldström, Centrally controlled distributed vnf state management, in: *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, 2015, pp. 37–42.
- [37] Y. Wang, G. Xie, Z. Li, P. He, K. Salamatian, Transparent flow migration for NFV, in: *2016 IEEE 24th International Conference on Network Protocols, ICNP, IEEE*, 2016, pp. 1–10.
- [38] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, S. Shenker, Elastic scaling of stateful network functions, in: *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 18*, 2018, pp. 299–312.
- [39] P. Sun, J. Lan, J. Li, Z. Guo, Y. Hu, T. Hu, Efficient flow migration for NFV with graph-aware deep reinforcement learning, *Comput. Netw.* 183 (2020) 107575.
- [40] J. Liu, W. Lu, F. Zhou, P. Lu, Z. Zhu, On dynamic service function chain deployment and readjustment, *IEEE Trans. Netw. Serv. Manage.* 14 (3) (2017) 543–553.
- [41] A.N. Toosi, J. Son, Q. Chi, R. Buyya, ElasticSFC: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds, *J. Syst. Softw.* 152 (2019) 108–119.
- [42] Y. Xue, Z. Zhu, On the upgrade of service function chains with heterogeneous NFV platforms, *IEEE Trans. Netw. Serv. Manage.* 18 (4) (2021) 4311–4323.
- [43] B. Zhang, P. Zhang, Y. Zhao, Y. Wang, X. Luo, Y. Jin, Co-scaler: Cooperative scaling of software-defined NFV service function chain, in: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN, IEEE*, 2016, pp. 33–38.
- [44] V. Eramo, E. Miucci, M. Ammar, F.G. Lavacca, An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures, *IEEE/ACM Trans. Netw.* 25 (4) (2017) 2008–2025.

- [45] Y. Qin, D. Guo, L. Luo, J. Zhang, M. Xu, Service function chain migration with the long-term budget in dynamic networks, *Comput. Netw.* 223 (2023) 109563.
- [46] K. Tokuda, T. Sato, E. Oki, Network slice reconfiguration with deep reinforcement learning under variable number of service function chains, *Comput. Netw.* 224 (2023) 109636.
- [47] H. Tu, G. Zhao, H. Xu, Y. Zhao, Y. Zhai, A robustness-aware real-time SFC routing update scheme in multi-tenant clouds, *IEEE/ACM Trans. Netw.* 30 (3) (2022) 1230–1244.



Wei Han Chen received the Ph.D. degree with the department of computer science and technology, Tsinghua University, B.E. degree in telecommunications engineering and management and M.E. degree in software engineering from Beijing University of Posts and Telecommunications, China. He is currently working in the Zhongguancun Laboratory, Beijing, China. His research interests include network function virtualization, software defined networking and traffic engineering.



Zhiliang Wang received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University, China in 2001, 2003 and 2006 respectively. He is an Associate Professor in 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 received the B.E. degree in computer science and technology from Jilin University and the Ph.D. degree from Tsinghua University. He is an Assistant Professor in the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include computer networks, network security, and network system.



Xia Yin received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University in 1995, 1997 and 2000 respectively. She is a Full Professor in 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 received the B.S. degree from Tsinghua University and the Ph.D. degree from The Chinese University of Hong Kong. He is an Associate Professor in 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.