Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه
puyamoshaver@gmail.com | شماره تماس: ۰۹۳۵۴۹۴۱۴۳۴

Throughput Optimization and Delay Guarantee VNF Placement for Mapping SFC Requests in NFV-Enabled Networks

Yi Yue, *Student Member, IEEE*, Bo Cheng, *Member, IEEE*, Meng Wang, *Student Member, IEEE*, Biyi Li, Xuan Liu, *Student Member, IEEE*, and Junliang Chen

***Abstract*—Nowadays, network softwareization is an emerging techno-economic transformation trend that significantly impacts how enterprises deploy their network services. As an essential technology in this trend, Network Function Virtualization (NFV) enables scalable and inexpensive network services by flexibly instantiating Virtualized Network Functions (VNFs) on commercial-off-the-shelf devices. In this paper, we focus on the VNF placement problem in NFV-enabled networks, aiming to maximize the number of accepted Service Function Chain Requests (SFCRs) while guaranteeing their delay requirements. To improve resource utilization efficiency, we take account of Fundamental Resource Overheads (FROs) and the shareability of VNF instances. We mathematically formulate the VNF placement problem and propose the Throughput Optimization and Delay Guarantee (TO-DG) heuristic solution, consisting of an affinity-based SFCR mapping algorithm and a VNF request adjustment algorithm. The evaluation results show that the performance of TO-DG is near to results derived by ILP solver for small scale problems. Moreover, TO-DG obtains higher network throughput than contrasting schemes in different scenarios and significantly improves network resource utilization.**

***Index Terms*—Network function virtualization, VNF placement, SFC request, throughput optimization, delay guarantee.**

I. Introduction

TRADITIONALLY, middleboxes or Network Functions (NFs) are attached to dedicated hardware appliances, leading to high management and infrastructure costs. Fortunately, Network Function Virtualization (NFV) emerges as a viable approach to implement traditional NFs in software and deploy them on commercial-off-the-shelf devices, which significantly improves the cost-efficiency, conveniences, and flexibilities of cloud services [1], [2].

Nowadays, the majority of Cloud Service Providers (CSPs) leverage NFV technology for outsourcing NFs to the cloud. These NFs are composed in the form of Virtual Network Functions (VNF) which make them easily deployable and thus greatly reducing long-term Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) [3], [4]. Also, the technology enables the customization of services by chaining several NFs together through the realization of Service Function Chains (SFCs). Therefore, users can submit service requests to CSPs by submitting Service Function Chain Requests (SFCRs). In this paper, the elements in an SFCR are referred to VNF Requests (VNFRs).

To fulfill several of these service requests, it is important that CSPs are able to do so whilst maximizing their profits by efficiently utilizing the resources used for the fulfillment of the requests. However, it can be challenging as the positioning of VNFs within SFCs result in a combinatorial problem where the decision on which resources can be utilized to ensure an efficient global placement falls within the class of NP-complete problems hence making it strongly NP-hard. The challenging nature of the problem is compounded when requirements such as throughput and delay between VNFs in the SFCRs must be fulfilled. To address this challenge, we propose novel schemes based on the following factors:

1) *Fundamental Resource Overheads:* Generally, the resource consumption of a VNF is composed of two parts: one part is the resource consumption used to serve SFCRs when an VNF is in operation. The other part refers to the Fundamental Resource Overheads (FROs) that are resource consumption to support a VNF instance's basic functionality [5], [6], such as the resource consumption of maintaining VNF images and related libraries. To reduce FROs, we can exploit shareable VNF instances to accomplish SFCRs.
2) *Shareable VNF Instances:* Shareable VNF instances are implemented by multi-tenancy technology [7], allowing one type-x VNF instance to serve multiple type-x VNFRs. For a sharable VNF instance, the resource consumption of supporting its shared application platform and separate databases can be regarded as FRO. The FRO of a shareable VNF instance is fixed and can be partaken equally by the VNFRs hosted on it. We can exploit the same VNF instance to serve multiple VNFRs of the same type; even these VNFRs are from different SFCRs. In this way, we need fewer VNF instances to accomplish users' SFCRs, saving more server resources to serve more SFCRs.

Manuscript received October 6, 2020; revised March 18, 2021 and June 1, 2021; accepted June 6, 2021. Date of publication June 9, 2021; date of current version December 9, 2021. This work is supported by the National Key Research and Development Program of China under grant 2018YFB1003804, in part by the National Natural Science Foundation of China under grant 61921003, 61972043, 61772479. The associate editor coordinating the review of this article and approving it for publication was D. M. Hermann. *(Corresponding author: Bo Cheng.)*

The authors are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: yueyi@bupt.edu.cn; chengbo@bupt.edu.cn).

Digital Object Identifier 10.1109/TNSM.2021.3087838

1932-4537 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

The delay of an SFCR occurs on the links and servers it uses, respectively. Specifically, the delay on links is positively correlated with the links' resource utilization. The delay on servers is positively correlated with the servers' resource utilization (detailed mathematical formulation in Section IV-B). When processing one SFCR, its traffic must traverse a series of VNFs in a predefined order to get served. One feasible VNF placement scheme is to deploy all VNFs required by an SFCR on the same server (Scheme 1). The traffic load between VNFs is restricted in the server, and we can avoid the corresponding delay and bandwidth consumption on links. However, Scheme 1 requires the placement of numerous duplicated VNF instances of the same type in the network, which generates a large volume of FROs. Another feasible scheme is placing these required VNFs among multiple servers (Scheme 2). Due to the shareable VNFs, Scheme 2 requires fewer VNF instances than Scheme 1. Scheme 2 saves more FROs while leading to increased delay and bandwidth consumptions on links. So we can observe that there are tradeoffs between delay requirements, FROs, and bandwidth consumptions. We need to carefully devise the VNF placement scheme to achieve a balance between them.

In this paper, we focus on the VNF placement problem for mapping SFCRs in NFV-enabled networks. By optimizing resource consumption, we aim to maximize the number of accepted SFCRs while guaranteeing their delay requirements. To enhance network resource utilization, we take account of FROs and the shareability of VNF instances. Then we mathematically formulate our problem as an Integer Linear Programming (ILP) model. Several studies have proved that the VNF placement problem is NP-hard [2], [8]–[11], as is our VNF placement problem. Hence, we employ Gurobi [12] to solve this ILP when the problem scale is small. We also propose the Throughput Optimization and Delay Guarantee (TO-DG) heuristic solution to solve the ILP when the problem scale is large. TO-DG consists of an Affinity-based SFCR Mapping Algorithm (ASMA) and a VNFR Adjusting Algorithm (VAA). According to the communication overheads between servers, ASMA classifies the servers into different levels of affinity groups. Then ASMA map SFCRs based on the affinity groups. VAA utilizes the feature that FRO is partaken by VNFRs on the same sharable VNF to assemble VNFRs that need the same type of VNF to save more server resources.

The contributions of this paper are listed as follows:

- We consider FROs and multi-tenancy-based shareable VNF instances and observe the feature that VNFRs on shareable VNF instances can share FROs. Exploiting this feature, we can save more server resources to accept more SFCRs in the network.
- Taking the server resource consumption, bandwidth consumption, and SFCR delay requirements into account, we formulate the VNF placement problem as an ILP model, aiming to maximize the number of accepted SFCRs while guaranteeing their delay requirements.
- We propose the novel TO-DG heuristic solution to solve the problem. Then TO-DG is evaluated in detail and compared with solutions in existing literature. The evaluation results show that the performance of TO-DG is close to the ILP solver when the problem scale is small. For large-scale problems, TO-DG obtains higher network throughput than the benchmarks in different scenarios and significantly improves network resources' utilization efficiency.

The reminder of this paper is organized as follows. We present the related work in Section II. Then we illuminate the system model and related concepts in Section III. In Section IV, we state and mathematically formulate the problem. We propose the TO-DG heuristic solution in Section V. After that, TO-DG is evaluated in Section VI. Finally, we conclude this paper in Section VII.

II. Related Work

Recently, the study of the VNF placement problem has become a hot issue in academia, and many researchers have proposed various solutions. Laghrissi and Taleb [13] provided a high-level survey on resource allocation and VNF placement problem. Most of the existing researches mathematically formulate the VNF placement problem as Integer Linear Programming (ILP) models [6], [14]–[16], Binary Integer Programming (BIP) models [17]–[19] or Mixed Integer Linear Programming (MILP) models [20]–[22], which are NP-hard in general. Therefore, it is challenging to obtain optimal solutions efficiently, especially for large scale networks. In references [15] and [22], the authors demonstrated that, in the worst condition, the time consumption caused by using the existing optimization toolboxes is about four orders of magnitude higher than that of the heuristics.

Many works are focused on the SFCR related VNF placement. In particular, Pei *et al.* [17] studied the problem of mapping SFCRs over geo-distributed cloud networks. Then they formulated the problem as a BIP model and proposed a multi-layer-graph based algorithm, aiming to achieve load balancing and minimize the placement cost of VNF instances. Li *et al.* [6] investigated the VNF placement problem in the cloud datacenter, then proposed a polynomial-time heuristic to minimize the total used physical machines. They also considered the VNF placement in edge computing enabled networks [15] and devised a priority-based greedy algorithm, aiming to optimize the total resource consumptions. Bari *et al.* [14] employed ILP to determine the required number of VNFs and their placement, to optimize both the network resource utilization and the network operational costs, without violating the SLAs. Qi *et al.* [16] proposed an efficient VNF placement approach named MSGAS based on the accessible scope, which can optimize solution space size. Tang *et al.* [20] combined the VNF placement in operator data centers with traffic forecasting. They proposed a traffic forecasting approach based on slip-window linear regression then designed two heuristics via relaxing integer variables.

Zeng *et al.* [22], Mechtri *et al.* [23], and Hawilo *et al.* [21] investigated the VNF Forwarding Graph (VNFFG) related VNF placement problem, respectively. In [22], the authors provided three heuristics to jointly optimize spectrum utilization on fiber links, resource consumption, and VNF deployment cost in elastic optical networks. In [23], Mechtri *et al.*

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

studied VNF placement and chaining, aiming to obtain the optimal VNFFG matching in the substrate network. Based on Umeyama's eigendecomposition approach [24], they proposed a heuristic, which can match the extended adjacency matrix of a VNFFG with the adjacency matrix of the substrate network. Hawilo *et al.* [21] considered to optimize the communication delay between two dependent VNFs. Based on the inherited dependency between VNF and VNFFG, they classified VNF types into several sub-groups. Next, all the available servers are used to build a weighted graph where servers are connected with logical communication links. Finally, they calculate the betweenness centrality based on the weighted graph and sub-groups to obtain the optimal VNF placement scheme.

Numerous researchers have considered multiple optimization objectives in VNF placement problem. More in detail, Pham *et al.* [10] studied the VNF placement problem, where the objective was to minimize energy and traffic-aware costs jointly. They proposed a sampling-based Markov Approximation (MA) approach to solve the problem, combined with matching theory to reduce the convergence time. Sun *et al.* [25] investigated the VNF placement problem for energy-efficient and traffic-aware SFC orchestration across multiple clouds. They formulated the problem as an ILP model then proposed a low-complexity heuristic algorithm called Energy-Efficient online SFC request Orchestration across Multiple Domains (EE-SFCO-MD), which can generate near-optimal solutions of the above problem. Chiti *et al.* [26] studied the VNF placement in a fog domain and formulated the problem as a matching game with externalities. They devised two approaches, aiming to optimize both the worst application completion time and the number of applications in the outage. To optimize the energy consumption and resource utilization in NFV-enabled networks, Eramo *et al.* [27], [28] proposed a solution to map SFCRs and place VNFs based on Markov decision process theory. They also considered migrating VNF instances during periods of low traffic to reduce energy consumption. Dieye *et al.* [11] optimized the number and locations of VNFs to minimize costs while guaranteeing QoS requirements in content delivery networks. They formulated the problem as an ILP problem and provided a cost-efficient proactive algorithm to solve it. Reference [29] formulated the VNF scheduling problem as an MILP model and devised a genetic algorithm-based method to optimize the accumulative service delay.

In [8] and [15], the authors took account of shareable VNF instances in their VNF placement problem, which can lead to higher utilization of network resources. For FROs, [8] and [30] have the similar view with us. In [8], the authors mentioned that a VNF instance needs at least an image, an OS, some related libraries, and a virtualization layer to support its regular functioning. So VNF instances consume some resources to support their regular functioning even in an idle state. They investigated the availability-aware VNF deployment problem and proposed a joint deployment and backup scheme to solve the problem. According to the research in [5], the more VNFs instantiated on the server, the more FROs will be generated. The author in [30] mentioned that VNFs are usually executed by VMs, which have high resource consumption to run their own operating systems. To alleviate this problem, they consolidated NFs based on types and reconfigure the same type NFs in fewer VNFs. They also proposed a heuristic to find out the NF reconfiguration scheme with the largest decrement of VNFs.

In study [31], the authors focused on provisioning network services in an NFV-enabled network. They aimed to maximize the admitted requests while minimizing the implementation cost of the requests, assuming that limited numbers of instances of each service chain have been stored in data centers. To solve the problem, they proposed two approximation algorithms with probable approximation ratios, depending on whether the packet traffic of each request is splittable. The authors extended their work in [32]. They studied the throughput maximization problem, assuming that the requests dynamically arrive at the network one-by-one. Moreover, they proposed an online algorithm with a provable competitive ratio. Xu *et al.* [33] studied the throughput maximization problem of admitting a single NFV-enabled multicast request, where the implementation of the service chain of each request will be consolidated into a single cloudlet. Then, for the problem with and without resource capacity constraints, they developed two approximate algorithms, respectively. Jia *et al.* [34], [35] investigated a task offloading problem in an MEC network, where each offloading task requests a network function service with a maximum tolerable delay requirement. Their objective is to maximize the number of accepted requests while minimizing their admission cost. They also devised an efficient online algorithm to solve the problem. The authors in [36] studied the throughput maximization in mobile edge-cloud networks for a set of requests that arrive without knowing future arrivals. Then they developed an efficient algorithm for the throughput maximization problem, by reducing the problem to the single NFV-enabled multicast request admission. They also devised an online algorithm with a provable competitive ratio when NFV-enabled multicast requests arrive one by one without the knowledge of future request arrivals.

The essential differences of the study in this paper from these mentioned studies [31]–[36] are:

1) We consider the virtualization overheads of VNFs called Fundamental Resource Overheads (FROs) in the paper. Involving FROs in the problem formulation makes the calculation of resource consumption more accurate.
2) We consider the shareable VNF instances implemented by multi-tenancy technology. For VNFRs that are hosted on the same shareable VNF, they share the FRO. By gathering the VNFRs that require the same type of VNF together, we can save more FROs and use these saved resources to accept more SFCRs.
3) We consider the impact of server/link resource utilization on SFCR delay, making the problem more challenging.

Next, we introduce the related models and concepts of the problem.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

III. Preliminaries

A. System Model

We use an undirected graph $G = (N, E)$ to represent the physical network topology, where N and E indicate the sets of physical servers and links, respectively. For network resources, we define C_u^{cpu}, C_u^{mem}, and C_{uw}^{bw} as the available CPU, memory and bandwidth resources on server u and link uw. Moreover, we use $r_{\gamma,u}^{cpu}$, $r_{\gamma,u}^{mem}$, and $r_{\gamma,uw}^{bw}$ to indicate the resource utilization rates of server u and link uw, in the condition that SFCR γ has been mapped.

In this paper, we let Γ denote the set of total SFCRs, and define a 3-tuple $(V_\gamma, E_\gamma, D_{max}^\gamma)$ to describe an SFCR γ, where V_γ represents the VNFRs in SFCR γ; E_γ represents the set of virtual links among the VNFRs of SFCR γ; and D_{max}^γ indicates the maximum tolerable delay of SFCR γ. For an SFCR γ, the parameters $\bar{u}, \bar{w} \in V_\gamma$ indicate two VNFRs, and $\bar{u}\bar{w} \in E_\gamma$ is the logical link between VNFRs $\bar{u}$ and $\bar{v}$.

B. Delay of SFC Requests

To achieve satisfactory SLA performance, we need to guarantee the real-time behavior of each SFCR. The delay of each SFCR is the sum of the following delays:

- the propagation delay is a fixed value that is proportional to the length of the physical link;
- the transmission delay is calculated by dividing the size of a transmitted packet by the bandwidth capacity of the link;
- the queuing delay depends on the link resource utilization (the ratio of allocated bandwidth to the total link bandwidth capacity) and the transmission delay on the link;
- the processing delay depends on the server resource utilization (the ratio of the allocated resource to the server's total resource capacity) and the per-packet processing delay on the server.

All the delays mentioned above are also defined in literature [17] and [37]. In practice, it is necessary to balance the load and avoid the VNF placement that utilizes servers or links with their maximum capacity. Therefore, we use an M/M/1 queuing model to calculate the queuing delay and processing delay, capturing the nearly linear growth in delay for low loads and associated high costs near system capacity. The detailed mathematical formulations are introduced in Section IV-B.

C. FROs and Shareable VNF Instances

A VNF is generally instantiated on a Virtual Machine (VM), and each needs an independent hypervisor, libraries, and guest operation system [38]. Therefore, some FROs are required to support running the basic functionality of a VNF. For isolation, we assume that different VNF instances cannot share FROs. Moreover, FROs are considered to be fixed when instantiating VNFs, and they do not affect the processing capacity of VNFs.

A shareable VNF instance is implemented by multi-tenancy technology, which is a technology in the domain of Software as a Service (SaaS) business paradigm [7]. Multi-tenancy technology enables one software instance to serve multiple tenants; hence one type-x VNF instance can serve multiple type-x VNFRs. For a sharable VNF instance, the resource consumption of supporting its shared application platform and separate databases can be regarded as FRO. As shown in Fig. 1, we can exploit the same VNF instance to serve multiple VNFRs of the same type; even these VNFRs are from different SFCRs. In this way, we need fewer VNF instances to accomplish users' SFCRs, saving more server resources to serve more SFCRs. For the VNF instance of the same type, sometimes, we only need to place one on a server. It is worth noting that, for one specific type of VNF, there exist multiple instances of it within the whole network.

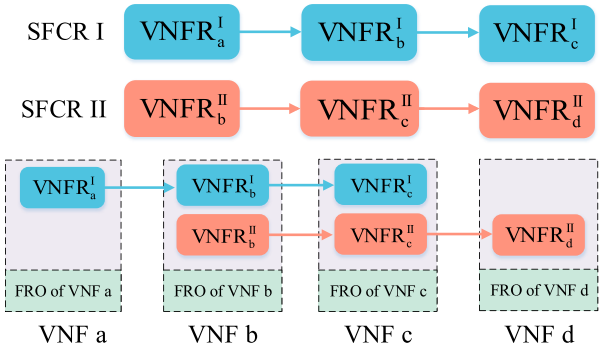


Fig. 1. Two SFCRs served by shareable VNFs.

In this paper, we instantiate VNF instances using multi-tenancy technology. The FRO on a sharable VNF instance can be partaken by the VNFRs on it, even if these VNFRs belong to different SFCRs. So we can assign multiple same type VNFRs to the same VNF instance they require. In this way, we need fewer VNF instances to accomplish users' SFCRs, effectively reducing server resource consumption.

D. Affinity Model

For a k-ary fat-tree topology [39], its size depends on the number of ports in the switches. Therefore, the k-ary fat-tree contains $(k/2)^2$ k-port core switches and k pods. Each pod consists of $k/2$ aggregation switches and $k/2$ edge switches. Moreover, there are $k^3/4$ servers in total and each pod connects with $(k/2)^2$ servers.

Fig. 2 shows a 4-ary fat-tree topology, and we can obtain its cost matrix A using Eq. (1):

$$A_{pq} = \begin{cases} 0 & \text{if } p = q \\ 2 & \text{if } \left\lfloor \frac{2(p-1)}{k} \right\rfloor = \left\lfloor \frac{2(q-1)}{k} \right\rfloor \\ 4 & \text{if } \left\lfloor \frac{2(p-1)}{k} \right\rfloor \neq \left\lfloor \frac{2(q-1)}{k} \right\rfloor \wedge \left\lfloor \frac{4(p-1)}{k^2} \right\rfloor = \left\lfloor \frac{4(q-1)}{k^2} \right\rfloor \\ 6 & \text{if } \left\lfloor \frac{4(p-1)}{k^2} \right\rfloor \neq = \left\lfloor \frac{4(q-1)}{k^2} \right\rfloor \end{cases} \tag{1}$$

where p and q are IDs of servers, and k is the number of ports in each switch. For instance, we can calculate $A_{12} = 2$ (p = 1 and q = 2), which indicates the number of physical links between server 1 and server 2 is 2.

Based on the cost matrix, we divide servers into several affinity groups at different levels. Table I shows the three affinity groups of server 1. Group A is the first affinity group, consisting of server 2, and its cost value is 2. The second

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

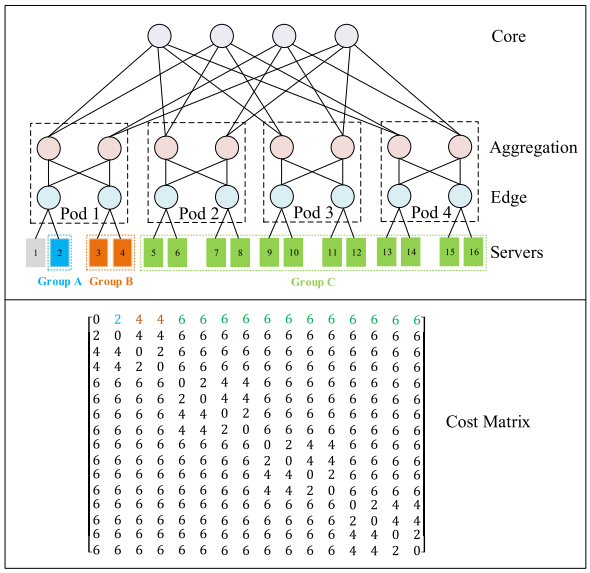


Fig. 2. Affinity groups and cost matrix in fat-tree topology.

TABLE I
AFFINITY GROUPS OF SERVER 1

Group	Cost	Servers
A	2	server 2
B	4	server 3, server 4
C	6	servers 5-16

affinity group is Group B, whose cost value is 4. Group B consists of server 3 and server 4. The remaining servers compose group C, the third affinity group, and its cost value is 6. When mapping SFCRs, we select servers based on the affinity group. We can preferentially map two connected VNFRs in the same low-cost affinity group during the SFCR mapping process, effectively reducing bandwidth consumption.

IV. Problem Statement

A. Problem Description

In a cloud network with the NFV environment, accepting more SFCRs can bring more profits to CSP. However, network resources are finite, and CSPs need to consider how to place VNFs reasonably to maximize the number of accepted SFCRs while guaranteeing their delay requirements. Fig. 3 shows a typical example of the VNF placement problem.

In a three-layer fat-tree topology, it contains four switches responsible and 16 servers. SFCRs I and II enter the data center from switches 17 and 19, respectively. VNFR_a^I, VNFR_b^I, VNFR_c^I, and VNFR_d^I are VNFRs in SFCR I, which are mapped on servers 1, 3, 5, and 9, respectively; and VNFR_a^{II}, VNFR_e^{II}, VNFR_c^{II}, and VNFR_d^{II} are VNFRs in SFCR II, which are mapped on servers 16, 13, 5, and 9, respectively. Considering that all of the VNF instances are shareable, the VNF placement scheme is: two VNF_a instances placed on servers 1 and 16; one VNF_b instance on server 3; one VNF_c instance on server 5; one VNF_d instance on server 9; and one VNF_e instance on server 13. Consequently, the traffic path of

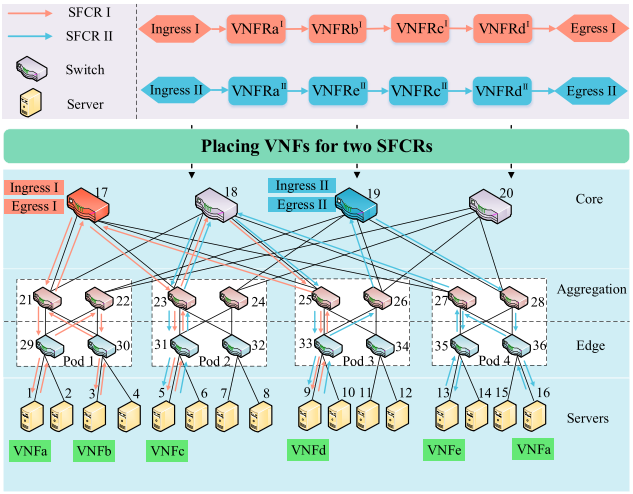


Fig. 3. A VNF placement case in fat-tree topology.

SFCR I is $17 \to 21 \to 29 \to 1 \to 29 \to 22 \to 30 \to 3 \to 30 \to 21 \to 17 \to 23 \to 31 \to 5 \to 31 \to 23 \to 18 \to 25 \to 33 \to 9 \to 33 \to 25 \to 17$, and represented by pink arrows. The traffic path of SFCR II is $19 \to 28 \to 36 \to 16 \to 36 \to 27 \to 35 \to 13 \to 35 \to 27 \to 18 \to 23 \to 31 \to 5 \to 31 \to 23 \to 18 \to 25 \to 33 \to 9 \to 33 \to 26 \to 19$, and represented by blue arrows.

If we map the VNFR_a^{II} on server 1, we can save the FRO of the VNF_a instance on server 16. However, this can change the traffic path of SFCR II to $19 \to 22 \to 29 \to 1 \to 29 \to 21 \to 17 \to 27 \to 35 \to 13 \to 35 \to 27 \to 18 \to 23 \to 31 \to 5 \to 31 \to 23 \to 18 \to 25 \to 33 \to 9 \to 33 \to 26 \to 19$. The number of links used by SFCR II increases from 22 to 24, resulting in more bandwidth consumption. Moreover, the processing delay of an SFCR relies on the resource utilization of the related servers where it is hosted. The higher the resource utilization of a server, the higher the processing delay of the relevant SFCRs deployed on the server. Since there are more VNFRs mapped on server 1, the resource utilization of server 1 increases, leading to higher processing delay of SFCR I and II, which may affect the SLA performance of them.

This simple case demonstrates the conflict between server resource consumption, bandwidth consumption, and delay requirements. Next, we present the mathematical formulation for our VNF placement problem.

B. Problem Formulation

In this subsection, we make a formulation about the throughput optimization and delay guarantee VNF placement problem. All the symbols and variables used in this part are listed in Table II.

We first need to guarantee that one VNFR of an SFCR is mapped on only one server as

$$\sum_{u \in N} x_{\bar{u},u}^{\gamma} = 1, \ \bar{u} \in V_{\gamma}. \tag{2}$$

where the binary variable $x_{\bar{u},u}^{\gamma} = 1$ indicates that a VNFR in SFCR γ is mapped on a server u, and 0 otherwise.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

TABLE II
NOTATIONS

Parameters	Descriptions
Topology related	
N	Set of physical servers in the network.
E	Set of physical links in the network.
u, w	Two servers in the network.
uw	The physical link between servers u and w.
SFCR related	
Γ	Set of SFCRs, $\gamma \in \Gamma$ denotes an SFCR γ.
$V_\gamma, \lvert V_\gamma \rvert$	Set of VNFRs in SFCR γ and number of VNFRs in SFCR γ.
E_γ	Set of virtual links among the VNFRs in SFCR γ.
$\bar{u}, \bar{w}$	Two VNFRs of an SFCR γ, $\bar{u}, \bar{w} \in V_\gamma$.
$\bar{u}\bar{w}$	The virtual link between $\bar{u}$ and $\bar{w}$, $\bar{u}\bar{w} \in E_\gamma$
Θ	The types numbers of VNF instances, $\theta \in \{0, 1, ..., \Theta - 1\}$ is an element.
Input parameters	
$cpu_{\bar{u}}^{\gamma}, mem_{\bar{u}}^{\gamma}$	CPU and memory consumption required by VNFR $\bar{u}$ in SFCR γ to accomplish a service.
$fro_\theta^{cpu}, fro_\theta^{mem}$	CPU and memory FROs when implementing a type-θ VNF instance.
$b_{\bar{u}\bar{w}}^{\gamma}$	Bandwidth consumption of virtual link between VNFRs $\bar{u}$ and $\bar{w}$ in SFCR γ.
$d_{uw}^{prop}, d_{uw}^{trans}$	Propagation delay and transmission delay of link uw.
d_u^{proc}	Processing delay per packet on server u, $u \in N$.
C_u^{cpu}, C_u^{mem}	The capacity of CPU and memory on server u, $u \in N$.
C_{uw}^{bw}	The link capacity of physical link uw.
D_{max}^{γ}	The maximum tolerable delay of SFCR γ.
Binaries	
$x_{\bar{u},u}^{\gamma}$	Binary variable that equals 1 if VNFR $\bar{u}$ in SFCR γ is hosted on server u and 0 otherwise.
$k_{\bar{u},\theta}^{\gamma}$	Binary variable that equals 1 if VNFR $\bar{u}$ in SFCR γ demands a type-θ VNF and 0 otherwise.
y_u^θ	Binary variable that equals 1 if a type-θ VNF instance is placed on server u and 0 otherwise.
$z_{uw}^{\gamma,\bar{u}\bar{w}}$	Binary variable that equals 1 if physical link uw hosts virtual link $\bar{u}\bar{w}$ and 0 otherwise.
h_γ	Binary variable that equals 1 if SFCR γ is completely mapped and 0 otherwise.

We use a binary variable y_u^θ to indicate whether a type-θ VNF instance is placed on server u:

$$y_u^\theta = \begin{cases} 1, \sum_{\gamma\in\Gamma}\sum_{\bar{u}\in V_\gamma} x_{\bar{u},u}^{\gamma} \cdot k_{\gamma,\theta}^{\bar{u}} \geq 1, \\ 0 \quad otherwise, \end{cases} \tag{3}$$

where $u \in N$, and $k_{\gamma,\theta}^{\bar{u}}$ denotes whether VNFR $\bar{u}$ in SFCR γ requires a type-θ VNF instance ($k_{\gamma,\theta}^{\bar{u}} = 1$) or not ($k_{\gamma,\theta}^{\bar{u}} = 0$).

For an SFCR γ, the consumptions of CPU and memory cannot exceed the available resources on servers. So the constraints are:

$$\sum_{\gamma\in\Gamma}\sum_{\bar{u}\in V_\gamma} cpu_{\bar{u}}^{\gamma} \cdot x_{\bar{u},u}^{\gamma} + \sum_{\theta\in\Theta} fro_\theta^{cpu} \cdot y_u^\theta \leq C_u^{cpu}, \quad u \in N. \tag{4}$$

where $u \in N$. The first term in Eq. (4) is the CPU consumption generated by VNFRs to implement services, and the second term denotes the CPU FROs of VNF instances.

Like Eq. (4) for CPU, we can formulate the constraint of memory resource as:

$$\sum_{\gamma\in\Gamma}\sum_{\bar{u}\in V_\gamma} mem_{\bar{u}}^{\gamma} \cdot x_{\bar{u},u}^{\gamma} + \sum_{\theta\in\Theta} fro_\theta^{mem} \cdot y_u^\theta \leq C_u^{mem}, \quad u \in N. \tag{5}$$

Following the server resource constraints, we introduce the bandwidth constraint on links. We first define the link variable:

$$z_{uw}^{\gamma,\bar{u}\bar{w}} = \begin{cases} 1, & \bar{u}\bar{w} \text{ is mapped on } uw \\ 0 & otherwise. \end{cases} \tag{6}$$

Usually, each virtual link in an SFCR corresponds to a physical path. It should be noted that a physical path may span multiple servers. In VNF placement problems, different VNFRs of an SFCR can be placed on the same server; namely, a part of virtual links in an SFCR may be limited in one server without involving any external links.

Then we can formulate the bandwidth constraint on links as

$$\sum_{\gamma\in\Gamma}\sum_{\bar{u}\bar{w}\in E_\gamma} b_{\bar{u}\bar{w}}^{\gamma} \cdot z_{uw}^{\gamma,\bar{u}\bar{w}} \leq C_{uw}^{bw}, \tag{7}$$

We consider the flow conservation of SFCRs as:

$$\sum_{w}^{N} z_{uw}^{\gamma,\bar{u}\bar{w}} \leq 1, \tag{8a}$$

$$\sum_{w}^{N} z_{wu}^{\gamma,\bar{u}\bar{w}} \leq 1, \tag{8b}$$

$$\sum_{w}^{N} z_{uw}^{\gamma,\bar{u}\bar{w}} - \sum_{w}^{N} z_{wu}^{\gamma,\bar{u}\bar{w}} = x_{\bar{u},u}^{\gamma} - x_{\bar{w},u}^{\gamma}, \tag{8c}$$

where $\bar{u}\bar{w} \in E_\gamma$, $uw \in E$, $\gamma \in \Gamma$.

Fig. (8)(a) determines whether a virtual link $\bar{u}\bar{w}$ in SFCR γ is mapped ($\sum_w^N z_{uw}^{\gamma,\bar{u}\bar{w}} = 1$) on one of the physical links that leave from server u, or if it is not ($\sum_w^N z_{uw}^{\gamma,\bar{u}\bar{w}} = 0$). Likewise, Fig. (8)(b) determines whether the same virtual link is mapped on one of the physical links that enter server u ($\sum_w^N z_{wu}^{\gamma,\bar{u}\bar{w}} = 1$), or not ($\sum_w^N z_{wu}^{\gamma,\bar{u}\bar{w}} = 0$). Fig. (8)(a) and 8(b) together guarantee that one virtual link cannot be split. We also use Fig. (8)(c) to ensure that the path of each virtual link is consecutive in the substrate network.

Equation (9) defines the possible mapping results of two VNFRs in an SFCR.

$$x_{\bar{u},u}^{\gamma} - x_{\bar{w},u}^{\gamma} = \begin{cases} 1, & \bar{u} \text{ is placed on } u, \\ -1, & \bar{w} \text{ is placed on } u, \\ 0, & otherwise, \end{cases} \tag{9}$$

If only one of $\bar{u}$ and $\bar{w}$ is mapped on server u, then we have $x_{\bar{u},u}^{\gamma} - x_{\bar{w},u}^{\gamma} = 1$ or $x_{\bar{u},u}^{\gamma} - x_{\bar{w},u}^{\gamma} = -1$. If $\bar{u}$ and $\bar{w}$ are mapped on the same server, then there will always be $x_{\bar{u},u}^{\gamma} - x_{\bar{w},u}^{\gamma} = 0$. Besides, if $\bar{u}$ and $\bar{w}$ are neither mapped on server u, then $x_{\bar{u},u}^{\gamma} - x_{\bar{w},u}^{\gamma} = 0$.

Following the link constraints, we introduce the delay constraint for SFCRs. The delay of each SFCR consists of propagation delay, transmission delay, queuing delay and processing delay. Based on the study in literature [17] and [37], the delay of an SFCR γ on link uw can be formulated as

$$d_{uw} = d_{uw}^{prop} + d_{uw}^{trans} + d_{uw}^{queuing}, uw \in E, \ \gamma \in \Gamma. \tag{10}$$

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

The first term in Eq. (10) is the propagation delay on link *uw*. The second is the transmission delay, which is calculated by dividing the size of a transmitted packet by the bandwidth capacity of the link. We assume that requests arrive according to the Poisson process with an average rate of 5 SFCRs per 100 time units. According to the Little's law in the M/M/1 queuing model, the queuing delay is calculated as:

$$d_{uw}^{queuing} = \frac{r_{uw}^{bw}}{1 - r_{uw}^{bw}} \cdot d_{uw}^{trans} \tag{11}$$

r_{uw}^{bw} is the ratio of the occupied bandwidth resource of *uw* to the bandwidth capacity of *uw*. r_{uw}^{bw} dynamically changes with the number of SFCRs mapped on the link *uw*. When new SFCRs are mapped to link *uw*, r_{uw}^{bw} increases; when SFCRs leave link *uw*, r_{uw}^{bw} decreases.

Similarly, the processing delay on server *u* is

$$d_u = \frac{r_u^{cpu}}{1 - r_u^{cpu}} d_u^{proc}, \tag{12}$$

where r_u^{cpu} is the CPU utilization of server *u*. r_u^{cpu} is the ratio of the occupied CPU resource of *u* to the CPU capacity of *u*. r_u^{cpu} dynamically changes with the number of SFCRs mapped on the server *u*. When new SFCRs are mapped to server *u*, r_u^{cpu} increases; when SFCRs leave server *u*, r_u^{cpu} decreases.

Consequently, the delay constraint for each SFCR can be formulated as:

$$\sum_{\substack{\bar{u}\bar{w}\in E_\gamma, \\ uw\in E}} d_{uw} \cdot z_{uw}^{\gamma,\bar{u}\bar{w}} + \sum_{\substack{\bar{u}\in V_\gamma \\ u\in N}} d_u \cdot x_{\bar{u},u}^{\gamma} \leq D_{max}^{\gamma}, \gamma \in \Gamma. \tag{13}$$

We use a binary variable h_γ to indicate whether an SFCR γ is completely mapped in the substrate network

$$h_\gamma = \begin{cases} 1, \sum_{u\in N} \sum_{\bar{u}\in V_\gamma} x_{\bar{u},u}^{\gamma} = \left|V_\gamma\right|, \\ 0 \quad otherwise, \end{cases} \tag{14}$$

Equation (14) indicates that only if all VNFRs of an SFCR are mapped on servers, the SFCR is completely mapped. The total number of accepted SFCRs in the network is

$$H_t = \sum_{\gamma\in\Gamma} h_\gamma. \tag{15}$$

In this paper, the network throughput is defined as the traffic of all accepted SFCRs. Our target is to serve as many SFCRs in Γ as possible while guaranteeing the delay requirements of SFCRs and resource constraints on servers and links

$$\begin{aligned} &\max \ \ H_t, \\ &s.t. \ \ Eq.\ (2)\ to\ (15). \end{aligned} \tag{16}$$

References [40] and [41] have turned out the VNF placement problem is NP-hard, so is our VNF placement problem. We design an efficient heuristic solution to solve it in polynomial time.

V. Proposed Heuristic Solution

A. General Concept of TO-DG

In this section, we present the Throughput Optimization and Delay Guarantee VNF placement approach, TO-DG. The approach consists of an Affinity-based SFCR Mapping Algorithm (ASMA) and a VNFR Adjusting Algorithm (VAA).

ASMA is responsible for mapping SFCRs to the servers based on affinity groups. During the mapping phase, we process SFCRs one at a time and preferentially map it to a server as a whole. If the resource requirements of the SFCR exceeds the server's available resources, we divide the SFCR into multiple VNFRs and map them based on affinity groups, thus reducing bandwidth consumption. Based on the above mapping results, the traffic between the servers is minimized so as to consume the least bandwidth. However, we must instantiate almost all kinds of VNF instances on each server. FRO increases with the increasing number of VNF instances.

VAA is designed to adjust VNFRs among servers. Based on the mapping results derived by ASMA, VAA carefully determines the mapping positions of VNFRs. For VNFRs that demand the same type of VNF, VAA carefully determines which VNFRs can be adjusted and then assembles these VNFRs into a smaller group of servers. Then we can efficiently reduce the number of placed VNF instances and save more FROs. Subsequently, we can use the saved resources to accept more SFCRs in the network.

In summary, ASMA calculates an intermediate result for SFCRs, which means the output of ASMA is not the final SFCR mapping scheme. Namely, when the execution of ASMA ends, the SFCRs have not yet been mapped to the servers. Based on the intermediate result derived by ASMA, VAA adjusts the mapping position of some VNFRs and then derives the final mapping schemes. Finally, SFCRs are mapped according to the scheme derived by VAA. Next, we describe the TO-DG approach in detail.

B. Throughput Optimization and Delay Guarantee VNF Placement

Algorithm 1 shows the framework of the affinity-based SFCR mapping algorithm. For each SFCR, we first randomly select a server from *N* as the target server SRV_i (line 3 in Algorithm 1). At line 4, we check whether the server SRV_i can host the SFCR. We also need to ensure that the delay requirement of an SFCR is satisfied. Then we calculate the delay of SFCR by using Eqs. (10) and (12) (line 5 in Algorithm 1). If there exist no delay violations, then we treat the SFCR as a whole and map it to SRV_i (lines 6-8 in Algorithm 1). If the server SRV_i can host the SFCR, and there exist no delay violations, then we treat the SFCR as a whole and map it to SRV_i. The delay violation indicates that there are SFCRs on the target server whose delay requirements are not satisfied. In this way, we can restrict the traffic between VNFRs in the same server, generating less bandwidth consumption. If there exist delay violations or SRV_i cannot host the SFCR, then we decompose the SFCR into multiple VNFRs and map them based on affinity groups (Procedure 1).

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

Algorithm 1 Affinity-Based SFCR Mapping Algorithm

Input: Set of SFCRs: Γ; Physical network: (N, E);
Output: Set of used servers: P_0; Mapping results: R_0;
1: Initialize the mapping result $R_0 = \varnothing$.
2: **for** *each SFCR in* Γ **do**
3: Initialize a target server SRV_p.
4: **if** SRV_p *can host SFCR* γ **then**
5: Calculate the delay of SFCR using Eqs. (10)-(13).
6: **if** *there exist no delay violations* **then**
7: Map SFCR γ on the server SRV_p and update R_0.
8: **else**
9: Obtain the affinity group G_{srv} ←**Function 1**.
10: Divide the SFCR into multiple VNFRs and map them based on affinity groups ← **Procedure 1**.
11: **end if**
12: **else**
13: Obtain the affinity group G_{srv} ←**Function 1**.
14: Divide the SFCR into multiple VNFRs and map them based on affinity groups ← **Procedure 1**.
15: **end if**
16: **end for**
17: **return** P_0, R_0.

Procedure 1 Decompose the SFCR

1: Initialize the mapping status $S = false$.
2: Back up the current mapping result R_0.
3: **for** *each VNFR in current SFCR* γ **do**
4: Set the mapping status $S = false$.
5: **for** *each server in* G_{srv} **do**
6: **if** *the current server can hold the VNFR* **then**
7: Map the VNFR to the server and update R_0.
8: Set the mapping status $S = true$.
9: Break.
10: **end if**
11: **end for**
12: **if** $S = false$ **then**
13: Restore the mapping result R_0.
14: Break.
15: **end if**
16: **end for**
17: **if** *there exist delay violations* **then**
18: Restore the mapping result R_0.
19: **end if**

Function 1 Generate the Affinity Group

Input: The target server SRV_p;
Output: The server affinity group G_{srv};
1: **Initialize**: Get all servers of the physical network N.
2: **for** *each server* SRV_q *in* N **do**
3: **if** $u_q < \omega$ **then**
4: **if** $\left\lfloor \frac{2(p-1)}{k} \right\rfloor = \left\lfloor \frac{2(q-1)}{k} \right\rfloor$ **then**
5: Add SRV_q in affinity group G_a.
6: **else**
7: **if** $\left\lfloor \frac{4(p-1)}{k^2} \right\rfloor = \left\lfloor \frac{4(q-1)}{k^2} \right\rfloor$ **then**
8: Add SRV_q in affinity group G_b.
9: **end if**
10: **else**
11: **if** $\left\lfloor \frac{4(p-1)}{k^2} \right\rfloor \neq \left\lfloor \frac{4(q-1)}{k^2} \right\rfloor$ **then**
12: Add SRV_q in affinity group G_c.
13: **end if**
14: **else**
15: Add SRV_q in affinity group G_d.
16: **end if**
17: **else**
18: Add SRV_q in affinity group G_d.
19: **end if**
20: **end for**
21: Add affinity groups G_a, G_b, G_c, and G_d into G_{srv} in sequence.
22: **return** G_{srv}.

The details about how to obtain the affinity groups are shown in Function 1. This function takes the current target server SRV_i as input, and outputs the set of affinity groups G_{srv}. In line 3 of Function 1, we check whether the server has enough resources to host SFCRs. We use u_q to indicated the resource utilization of SRV_q. ω represents the resource utilization threshold, and it is set to 85%. Since Algorithm 1 executes more than once, servers with resource utilization higher than ω will occur in the subsequent execution process. Based on the affinity model in Section III-C, servers are added to groups A, B, and C, respectively (lines 4-13 in Function 1). The remaining servers form group D, including servers that already host VNFs and the others (lines 15-19 in Function 1). Finally, we sequentially add the four affinity groups into G_{srv}.

In Procedure 1, VNFRs of the SFCR are mapped to servers one by one in sequence. First, we initialize the mapping status $S = false$, and back up the current mapping result R_0 (lines 1-2 in Procedure 1). Since the affinity groups are arranged, we can preferentially select the servers in the low-cost affinity group to map VNFRs. The algorithm traverses the servers in G_{srv} until a server can hold the VNFR (lines 5-11 in Procedure 1). After mapping the VNFR, we update R_0 and set $S = true$ (lines 7-8 in Procedure 1). Then the traversal stops and starts to process the next VNFR in the SFCR.

The mapping status S is defined to indicate whether a VNFR has been successfully mapped. Before mapping each VNFR, S is initialized to false. If the VNFR is mapped successfully, then we modify S to true. In lines 12-15 in Procedure 1, S is checked before processing the next VNFR. If S is false, the mapping result derived by Procedure 1 is restored to the last mapping decision. Moreover, if the delay requirements of SFCRs on the related servers are violated, R_0 is also restored to the last mapping decision (lines 17-19 in Procedure 1). By exploiting affinity groups, we can effectively reduce the used physical links when mapping SFCRs, thus reducing bandwidth consumption.

Algorithm 2, along with Procedure 2 and 3, shows the process of VNFR adjusting. The main notations used in Algorithm 2 is summarized in Table III. Since the corresponding consumption of an activated server is much higher than

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

TABLE III
NOTATIONS IN ALGORITHMS

Notations	Description
μ	The set of VNFRs that demand the identical type of VNF instance in the same server.
G_s	The list that stores all server groups. g_s is a group in G_s.
res_b	The total resource consumption before transferring μ.
res_a	The total resource consumption after transferring μ.
srv_f	The server that is ready to be adjusted.
srv_l	The server that is ready to receive VNFRs from srv_f.
β_f	The pointer that points to srv_f.
β_l	The pointer that points to srv_l.
i	The regulated value that adjusts the position of β_f.

Algorithm 2 VNFR Adjusting Algorithm

Input: Set of used servers: P_0; Mapping results: R_0;
Output: Set of used servers: P_1; Mapping results: R_1;

1: Classify the used servers into several groups and put these groups into a list G_s.
2: **for** each group g_s in G_s **do**
3: Back up the current network status.
4: Adjust VNFRs within the group. ←**Procedure 2**.
5: **if** *there exist resource violations* **then**
6: Select the server with the most remaining resources from group g_s.
7: Transfer part of VNFRs in the resource-violation server and update R_1.
8: **if** *above process succeeds* **then**
9: Record the current results.
10: **else**
11: Restore the network status.
12: **end if**
13: **else**
14: Record the current results.
15: **end if**
16: **end for**
17: Adjust VNFRs among all used servers. ←**Procedure 3**.
18: **return** P_1, R_1

Procedure 2 Adjust VNFRs Within the Group

1: **while** *there exists unprocessed* μ *in* g_s **do**
2: Find the μ that consumes the minimal server resource in each server in group g_s, and store them in a list L_{temp}.
3: Sort the μs in L_{temp} in ascending order based on their resource consumption.
4: **for** *each* μ *in* L_{temp} **do**
5: Calculate resource consumption res_b before transferring μ.
6: Select the target server from the current group to transfer μ.
7: Calculate the delay of μ-related SFCRs using Eqs. (10)-(13).
8: **if** *delay constraints of* μ*-related SFCRs are satisfied* **then**
9: Transfer μ to the target server.
10: **end if**
11: **if** *delay constraints of old SFCRs are satisfied* **then**
12: Calculate resource consumption res_a after the transfer operation and tag the μ as processed.
13: **if** $res_a < res_b$ **then**
14: Replace the previous result with the current one.
15: Remove the corresponding VNF instances from the related server.
16: **else**
17: Continue.
18: **end if**
19: **else**
20: Tag the μ as processed.
21: **end if**
22: **end for**
23: **end while**

other expenditures, we should activate as few servers as possible to optimize the total resource consumption. To briefly describe our algorithm, we consider the VNFRs of the same type in the same server as a whole, symbolized as μ. In other words, for each μ, the VNFRs in it require the same type of VNF instance. We classify all used servers into several groups and store them in a list G_s (line 1 in Algorithm 2). That is, servers belong to the same pod are classified into the same group. The servers in the same group are close to each other. Next, Procedure 2 is conducted to adjust VNFRs within each group (line 4 in Algorithm 2).

In Procedure 2, we take each μ as a whole to adjust VNFRs. First, we figure out the μ that consumes the minimal server resource in each server belonging to g_s, and store them into a list L_{temp} (line 2 in Procedure 2). Based on resource consumption, the μs in L_{temp} are sorted in ascending order (line 3 in Procedure 2). Next, for each μ in L_{temp}, we select the target server from the current group (lines 4-21 in Procedure 2). The qualified target servers must satisfy the following conditions:

1) The resource utilization of the target server is higher than that of the server, where μ is located initially.
2) The target server holds the corresponding VNF instance and has sufficient server resources left.

We calculate the delay of μ-related SFCRs by using Eqs. (10)-(13) (line 7 in Procedure 2). When we transfer a μ to the target server, the delay requirements of all μ-related SFCRs must be satisfied (lines 8-10 in Procedure 2). However, the delay requirements of SFCRs that are initially on the target server may also be affected. To describe the process more clearly and briefly, we use "old SFCRs" to refer to these SFCRs. Similarly, we obtain the delay of old SFCRs using Eqs. (10)-(13). Then we check whether the delay requirements of all old SFCRs are still satisfied (line 11 in Procedure 2). If so, the transfer operation succeeds. Then we calculate the total resource consumption res_a after the transfer operation and tag μ as processed. Otherwise, we only tag μ as processed, to avoid duplicate processing. In lines 13-18 in Procedure 2, we check whether the total resource consumption is reduced compared to its previous value. If so, then we replace the last placement result with the current one, and the VNF instance related to μ is removed from the corresponding server. The

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

while loop will not be terminated until every μ in g_s is labeled as processed.

After gathering the VNFRs that require the same type VNF instance, the corresponding VNF instance on some servers can be removed. Therefore, the FROs in these servers can be reduced. However, the above process may lead to resource violations on servers. Therefore, we try to eliminate resource violations in lines 5-15 in Algorithm 2. We transfer the VNFRs, one by one, from the resource-violation server to servers with sufficient resources until the corresponding server has no resource violations. To receive VNFRs from resource-violation servers, we select the server with the most remaining resources in group g_s (line 6 in Algorithm 2). Empty servers cannot be selected to receive VNFRs. If the process of elimination succeeds, then we record the current results (lines 8-10 in Algorithm 2). Otherwise, we abandon the results derived from Procedure 2 and restore the network status.

After the adjustment process in each group, almost every server obtains some idle resources. We try to transfer VNFRs from servers with low resource utilization to the servers with high utilization to make the best use of the network resource. In lines 1-2 in Procedure 3, we store all the used servers in a list L_s and sort them in ascending order according to their resource utilization. To make the adjustment process more efficient, we define two pointers β_f and β_l and let them point to the first server srv_f and the last server srv_f in L_s, respectively (line 3 in Procedure 3). Next, we transfer the VNFRs from srv_f to srv_l (lines 4-37 in Procedure 3). If all VNFRs in srv_f are transferred, then we release the server and modify the network status (lines 12-15 in Procedure 3). Otherwise, we point β_l to a new server to continue the transfer operation in srv_f (line 16 in Procedure 3).

Lines 19-36 are responsible for checking the conditions that end the outer *while* loop. When $\beta_f = \beta_l$, it means that we have traversed all servers whose utilization is higher than that of srv_f, and the VNFRs in srv_f cannot be completely moved out. Therefore, we restore the network to its previous status (line 20 in Procedure 3). Then we set the regulated value i to $i+1$ to select the next server and redirect β_l to the end of L_s (lines 21-22 in Procedure 3). For $\beta_f+i \geq \beta_l$, it implies that we have treated each used server as srv_f once, and Procedure 3 should be terminated (lines 23-24 in Procedure 3). If not, we swap two servers corresponding to $\beta_f + i$ and β_l, and select the server which is indicated by β_f+i as the next srv_f (line 26 in Procedure 3).

In the condition of $\beta_f < \beta_l$, it indicates that srv_f holds no VNFRs. So we select the next server in L_s as the new srv_f (line 29 in Procedure 3). For $\beta_f = \beta_l$, it implies that we have treated each used server as srv_f once, and we terminate the adjustment process (lines 31-33 in Procedure 3).

Procedure 3 Adjust VNFRs Among all Used Servers

1: Calculate the resource utilization of each used servers.
2: Store all used servers in a list L_s, and sort them based on resource utilization in ascending order.
3: Define two pointers, β_f and β_l, and a regulated value i. Let β_f point to the first server srv_f in L_s, let β_l point to the last server srv_l in L_s, and set $i = 0$.
4: **while** 1 **do**
5: Back up the current network status.
6: **while** $\beta_f < \beta_l$ **do**
7: **for** *each VNFR in* srv_f **do**
8: **if** srv_l *can hold the VNFR* **then**
9: Transfer the VNFR to srv_l.
10: **end if**
11: **end for**
12: **if** srv_f *is empty* **then**
13: Release srv_f and modify the network status.
14: Break.
15: **else**
16: $\beta_l = \beta_l - 1$
17: **end if**
18: **end while**
19: **if** $\beta_f = \beta_l$ **then**
20: Restore the network to its previous status.
21: $i = i + 1$.
22: Let β_l point to the last server of L_s.
23: **if** $\beta_f + i \geq \beta_l$ **then**
24: Break.
25: **else**
26: Swap the servers corresponding to $\beta_f + i$ and β_f.
27: **end if**
28: **else**
29: $\beta_f = \beta_f + 1$.
30: Let β_l point to the last server of L_s.
31: **if** $\beta_f = \beta_l$ **then**
32: Break.
33: **else**
34: Continue.
35: **end if**
36: **end if**
37: **end while**

C. Complexity Analysis

In Algorithm 1, the iteration time of the outer *for* loop is $|\Gamma|$. In Function 1, the *for* loop (lines 2-20 in Function 1) and the operation at line 21 both run N times, where N is the number of servers. So the time complexity of Function 1 is:

$$N + N = 2N, \tag{17}$$

which is at the level of $\mathbf{O}(N)$. For Procedure 1, we define a variable σ to indicate the VNFRs in an SFCR. So the outer *for* loop runs σ times. In lines 5-11 in Procedure 1, the inner *for* loop runs $|N|$ times. Hence, the time complexity of Procedure 1 is $\sigma \cdot N$, and the time complexity of Algorithm 1 is:

$$|\Gamma| \cdot (2N + \sigma N) = |\Gamma| N(\sigma + 2), \tag{18}$$

which is at the level of $\mathbf{O}(\Gamma N \sigma)$.

To illustrate the time complexity of Algorithm 2 more concisely, we introduce two variables: G_c and ε. G_c is the number of groups that contain used servers, and ε is the number of used servers in one group. So the *for* loop (lines 2-15 in Algorithm 2) needs to run $|G_c|$ times. As for Procedure 2, since each server contains at most Θ types of VNFRs, the complexity of line 2 in Procedure 2 is $\Theta \cdot \varepsilon$ in the worst case. The

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه
puyamoshaver@gmail.com | شماره تماس : ۰۹۳۵۴۹۴۱۴۳۴

time complexity of the sorting process at line 3 is $\log|\varepsilon|$, and the iteration time of the *for* loop (lines 4-21 in Procedure 2) is $|\varepsilon|$. The iteration time of the *while* loop in Procedure 2 relies on the number of VNFR types, so the time complexity of this process is Θ. Then lines 2-15 in Algorithm 2 has a total time complexity of $G_c \cdot \Theta \cdot (\varepsilon + \Theta \cdot \varepsilon + log\varepsilon)$, which is at the level of $\mathbf{O}(G_c\varepsilon\Theta^2)$.

Another time-consuming part of Algorithm 2 is Procedure 3. There are $G_c \cdot \varepsilon$ used servers and $\sum_{\gamma=0}^{|\Gamma|-1} |V_\gamma|$ VNFRs in total. Then each server holds $\sum_{\gamma=0}^{|\Gamma|-1} |V_\gamma|/(G_c \cdot \varepsilon)$ VNFRs on average. Therefore, the inner *while* loop (lines 6-18 in Procedure 3) has a complexity of $\sum_{\gamma=0}^{|\Gamma|-1} |V_\gamma|/(G_c\varepsilon) \cdot (G_c\varepsilon) = \sum_{\gamma=0}^{|\Gamma|-1} |V_\gamma|$. The iteration time of the outer *while* loop depends on the number of used servers, which is $G_c \cdot \varepsilon$. So the total time complexity of Procedure 3 is $G_c\varepsilon \sum_{\gamma=0}^{|\Gamma|-1} |V_\gamma|$.

Based on the above analysis, the total complexity of TO-DG is $\mathbf{O}(\Gamma N\sigma + G_c\varepsilon\Theta^2 + G_c\varepsilon \sum_{\gamma=0}^{|\Gamma|-1} |V_\gamma|)$. In the worst case, $G_c \cdot \varepsilon = N$. Moreover, $\sum_{\gamma=0}^{|\Gamma|-1} |V_\gamma| = \Gamma \cdot \sigma$. We assume that the number of chains is much larger than the chain length. Hence, the time complexity of TO-DG is at the level of $\mathbf{O}(\Gamma N + N\Theta^2)$.

VI. Performance Evaluation

This section reports the performance evaluation of TO-DG. First, we introduce the simulation settings used to evaluate the algorithms in Section VI-A. Next, the performance of TO-DG is compared with three different schemes from different aspects. We conduct all of the simulations in a computer with a 2.6-GHz Intel Core i7-6700 processor and 16 GB RAM. We summarize the related parameter settings in Table IV.

A. Simulation Setup

We implemented TO-DG and the contrastive schemes in Java based on ALEVIN [42], a widely used evaluation environment for VNF placement. Alevin employs a generator named Barabasi-Albert to create random network topologies. Based on Alevin, we generate a 3-layer fat-tree network topology [39]. The size of a k-ary fat-tree depends on the number of ports in the switches. The value of k is set to 8, so an 8-ary fat-tree topology contains 16 core switches, 128 servers, and eight pods; each pod contains four aggregation switches and four edge switches.

For each server, we set its CPU capacity and memory capacity to 1000 MIPS and 1000 MB, respectively. For each link, its bandwidth capacity is set to 1000 Mbps. There are 10 types of VNFs can be placed in the datacenter network. The number of VNFRs per SFCR is a random variable varying from [3, 4, 5, 6]. For each VNFR, we set the CPU, memory, and bandwidth requirements ($cpu_{\bar{u}}^{\gamma}$, $mem_{\bar{u}}^{\gamma}$, and $b_{\bar{u}\bar{w}}^{\gamma}$) as random values distributed between [10, 50]; the CPU FRO and memory FRO (fro_{θ}^{cpu} and fro_{θ}^{mem}) are set to 50 MIPS and 50 MB, respectively. A shareable VNF instance can simultaneously serve at most 10 VNFRs. Each SFCR should satisfy the constraints in Eqs. (2)-(15); otherwise, it cannot be accepted in the network.

TABLE IV
Parameter Settings in Evaluation

Description		Setting
Network topology		fat-tree topology
Number of switches		16
Number of servers		128
Number of VNF types		10
Number of VNFRs per SFCR		[3, 6]
Packet rate of an SFCR		[400, 4000] packets/second
Size of each packet		64 KB
Parameters	**Description**	**Value**
C_u^{cpu}	CPU capacity of server u	1000 MIPS
C_u^{mem}	Memory capacity of server u	1000 MB
$C_{(u,w)}^{link}$	Link capacity of link (u, w)	1000 Mbps
fro_{θ}^{cpu}	CPU FRO for instantiating a type θ VNF	50 MIPS
fro_{θ}^{mem}	Memory FRO for instantiating a type θ VNF	50 MB
$cpu_{\bar{u}}^{\gamma}$	CPU consumption of VNFR $\bar{u}$ in SFCR γ	[10, 50] MIPS
$mem_{\bar{u}}^{\gamma}$	Memory consumption of VNFR $\bar{u}$ in SFCR γ	[10, 50] MB
$b_{\bar{u},\bar{w}}^{\gamma}$	Bandwidth requirement of virtual link $(\bar{u}, \bar{w})$ in SFCR γ	[10, 50] Mbps
d_u^{proc}	Per-packet processing delay on node u	[0.045, 0.3] ms
$d_{(u,w)}^{prop}$	Propagation delay on link (u, w)	[1, 2] ms
$d_{(u,w)}^{trans}$	Transmission delay on link (u, w)	[2, 5] ms
D_{max}^{γ}	Maximum tolerable delay of SFCR γ	[50, 100] ms

By referring to literature [32], [43], the packet rate of an SFCR is randomly drawn between [400, 1000] packets/second, and the size of each packet is 64 KB. The per-packet processing (d_u^{proc}) delay is set as value between [0.045, 0.3] ms, the propagation delay (d_{uw}^{prop}), and transmission delay (d_{uw}^{trans}) are set as values varied randomly between [1, 2] ms and [2, 5] ms respectively. The maximum tolerable delay of each SFCR is between [50, 100] ms.

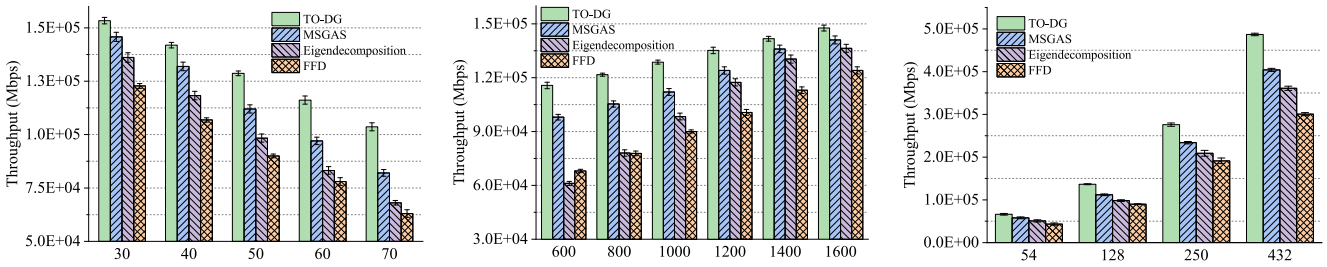
B. Introduction of Contrastive Algorithms

We compare TO-DG with the following algorithms.

- *First Fit Descending (FFD):* it first sorts all the VNFRs in descending order based on their average resource requirements and then exploits the first-fit algorithm to map the VNFRs on the servers one by one. FFD is a kind of classical greedy algorithm. It is generally used as a reference objective for other solutions.
- *Multi-Stage Graph Algorithm With the Accessible Scope (MSGAS):* it first selects all the required VNFs for each SFCR based on the accessible scope [16]. For all selected VNFs, MSGAS orchestrates them to match the predefined order of the SFCR. Next, MSGAS calculates the resource costs between VNFs, including VNF deployment cost, traffic forwarding costs, energy costs, and SLA violation penalties. Finally, MSGAS employs a multi-stage graph algorithm [14] to obtain the optimal chaining result and VNF placement solution. MSGAS also considered the conflict between server resource and bandwidth resource. They focus on achieving a good trade-off to improve the acceptance ratio of SFCRs.
- *Eigendecomposition [23]:* it formulates the physical network as an adjacent matrix (M_1) and calculates the weight of each element using the Widest-Shortest Path Routing (WSPR) algorithm. It generates an adjacent

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه



(a) Throughput with different FROs (b) Throughput with different link capacities (c) Throughput with different number of servers in fat-tree topology

Fig. 4. Throughput Comparisons with Different Factors.

matrix (M_2) for SFCRs based on their resource requirements. Then the conjugate matrices (C_1 and C_2) are derived using the eigenvector matrices of M_1 and M_2. The algorithm multiplies C_1 by C_2 to obtains the product C_p. Finally, it uses the positions with the maximum value in each row of C_p as the scheme of VNF placement. Similar to TO-DG, Eigendecomposition supported the concept of multi-tenancy and focus on accepting more SFCRs to maximizing provider revenue. So we select it as a compared approach.

C. Simulation Results

To reduce the accidental errors, we conduct 20 times of experiments for each group of results and use the average results as the final results. Error bars represent the 95% confidence intervals in each figure. We mainly observe how many SFCRs different approaches can accept when resources are exhausted. In the experiment, the number of SFCRs is set to a huge value (for example, different approaches can accept about 300-500 SFCRs on average, and we provide 1000 SFCRs). For different approaches, their available resources are the same. Therefore, each approach exhausts the available resources to accept as many SFCRs as possible.

1) Throughput Comparisons With Different Factors: We first evaluate the impact of FROs on TO-DG performance. Since the CPU and memory resources are simultaneously consumed when placing VNFs, we set the CPU FRO and memory FRO values to be the same. As shown in Fig. 4(a), it is evident that as FRO increases, the throughput of TO-DG is always higher than the benchmarks'. The reason is that TO-DG assembles VNFRs that require the same type of VNF, which effectively reduces the number of placed VNF instances. Therefore, with the same FRO, TO-DG can save more server resources compared to the benchmarks. It allows TO-DG to serve more SFCRs, leading to higher network throughput. Besides, the higher the FRO is, the better the performance of TO-DG.

The link capacity is another factor that may influence the performance of our approach. Fig. 4(b) shows that with the increasing link capacity, TO-DG always outperforms the benchmarks. More bandwidth resources help TO-DG adjust VNFRs more reasonably, so TO-DG can accept more SFCRs. Besides, with scarce bandwidth resources, our approach's throughput is much higher than that of the benchmarks. Since TO-DG tends to map the entire SFCR on one server, resulting in less inter-server traffic among the servers. It means that TO-DG has low dependence on bandwidth resources and faces fewer link bottlenecks. We can also observe that the network throughput in Eigendecomposition is sensitive to the link capacity. The WSPR algorithm in it derives longer traffic paths for SFCRs, which generates a large amount of bandwidth consumption. Therefore, when link capacity increases, its throughput rises obviously.

Next, we evaluate the performance of TO-DG with different number of servers in fat-tree topology. In Fig. 4(c), by varying the network size from 54 to 432, the throughput of all approaches increases. Since a larger network provides more available resources, more VNF instances can be placed to serve more SFCRs. Moreover, TO-DG has a consistent advantage over contrastive solutions. When the network size is large, the gap between TO-DG and the benchmarks is noticeable. The reason is that TO-DG considers the affinity between different servers during the mapping process. With the same amount of available network resources, TO-DG can utilize these resources more adequately than the benchmarks. As a result, a larger network size leads to more obvious advantages of TO-DG.

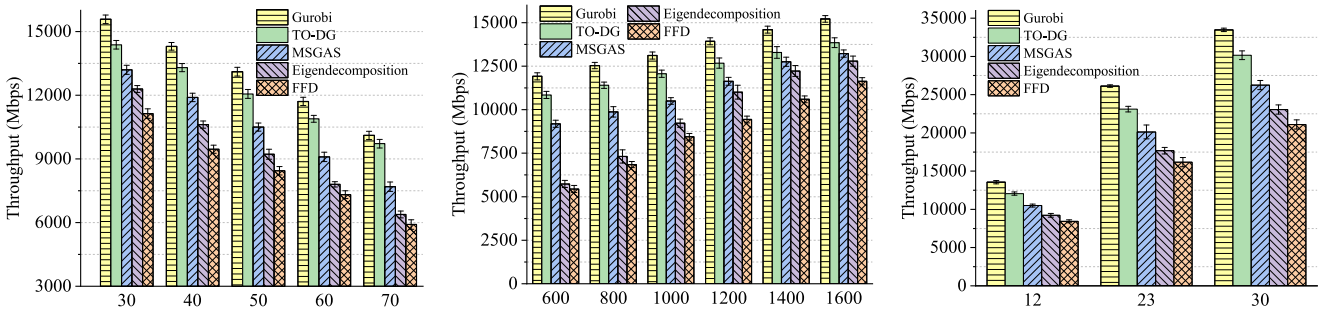
2) Throughput Comparisons With Gurobi: We reveal the gap between the performance of TO-DG and the ILP solver in this part. For small-scale problems, we can exploit Gurobi to solve the ILP problem. In Fig. 5 (a) and (b), we conduct the experiments with a physical network of 12 nodes and 15 links. In Fig. 5 (c), we take the network size as the variable. The impacts of different factors on different solutions' performance are consistent with the results in Fig. 4. In different scenarios, the average network throughput derived by TO-DG is about 7%, 10%, and 10% lower than the results of Gurobi. We can see that TO-DG can acquire near effect compared with the results derived by Gurobi when the problem scale is small. Moreover, TO-DG also performs better than the benchmarks.

3) Throughput Comparisons With Different SFCR Delay Requirements: In this part, we evaluate the impact of SFCR delay requirements on the performance of different solutions. We have two groups of SFCRs in the simulations:

- *Group A:* The delay requirements of SFCRs are between [50, 75] ms.
- *Group B:* The delay requirements of SFCRs are between (75, 100] ms.

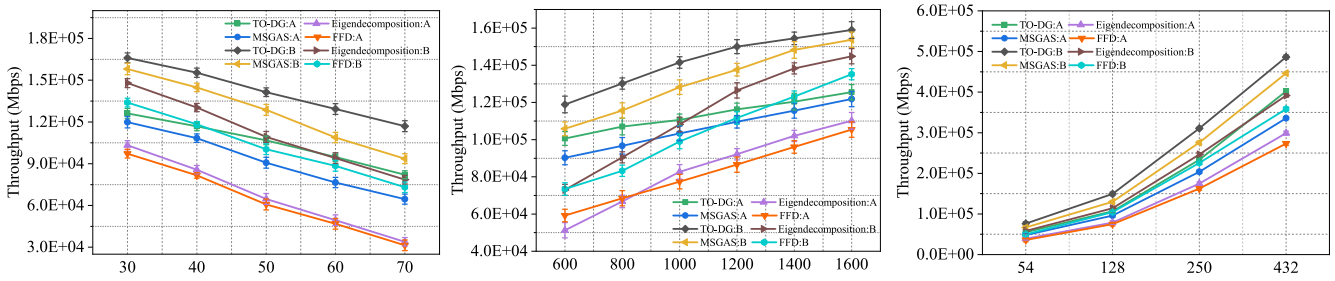
Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه



(a) Throughput with different FROs (b) Throughput with different link capacities (c) Throughput with different number of servers

Fig. 5. Throughput Comparisons with Gurobi.



(a) Throughput with different FROs (b) Throughput with different link capacities (c) Throughput with different number of servers in fat-tree topology

Fig. 6. Throughput Comparisons with Different SFCR Delay Requirements.

In Fig. 6(a), when FRO increases, the throughput of different algorithms gradually decreases. For the same group of SFCRs, TO-DG gets the highest performance, and the performance of MSGAS does better than Eigendecomposition. TO-DG combines the VNFRs that demand the same type of VNF together, and then fewer VNF instances are placed, so FROs are fewer. For SFCRs in group A, the performance of Eigendecomposition is much worse than TO-DG's. The Widest-Shortest Path Routing (WSPR) algorithm prefers to produce long routing paths for SFCRs, which leads to higher delay and more resource consumption. We can also observe that the bigger the volume of FROs is, the better TO-DG performs. When FRO reaches 70, TO-DG's throughput is about 1.3, 2.4, and 2.6 times the other results in group A; 1.2, 1.5, and 1.6 times the other results in group B. The compared approaches pay no attention to reducing the FROs in the network. Therefore, as FRO increases, resource bottlenecks are more likely to appear in compared approaches, resulting in worse performance.

In Fig. 6(b), as the link capacity increases, the throughput of different algorithms gradually increases. Moreover, TO-DG performs much better than the benchmarks when the bandwidth resource is scarce. When the link capacity is 600, TO-DG's throughput is about 1.1, 1.9, and 1.7 times the other results in group A; 1.1, 1.6, and 1.6 times the other results in group B. Because TO-DG is inclined to map the whole SFCR in one server, and then there is less traffic between the servers. Subsequently, there are fewer link bottlenecks in the network, and the resources of server can be utilized more adequately. Due to the WSPR algorithm, Eigendecomposition derives longer flow paths for SFCRs. Therefore, when it processes the SFCRs in group A, its performance is more likely to be affected. Moreover, its performance improves significantly as the link capacity increases.

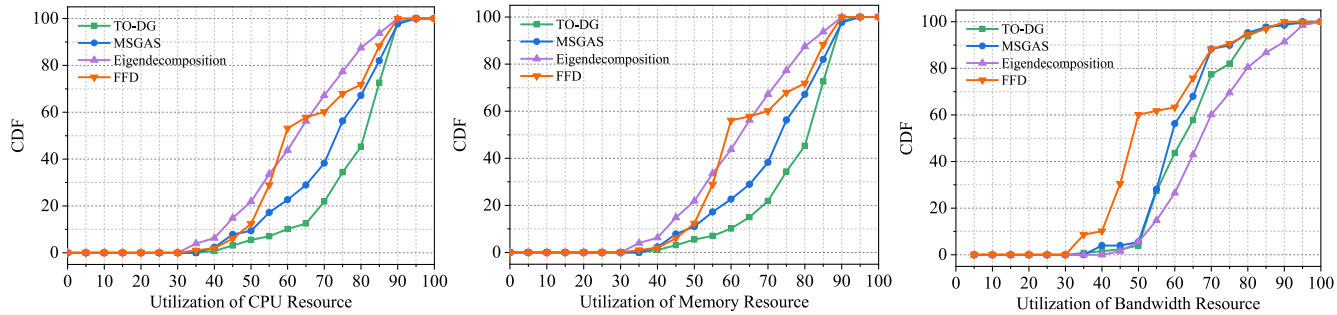
As shown in Fig. 6(c), TO-DG outperforms the benchmarks in both groups. More available resources lead to a more adequate adjustment of TO-DG so that it can accept more SFCRs. When the network size is large, the gap between TO-DG and the benchmarks is noticeable. Because TO-DG considers the affinity between different servers during the mapping process. With the same amount of available network resources, TO-DG can utilize these resources more adequately than the benchmarks. The performance of MSGAS is second only to that of TO-DG because MSGAS regards SFCR delay as its optimization objective, while TO-DG only regards SFCR delay as a constraint. The average delay of routing paths in MSGAS is lower than that of TO-DG. However, MSGAS ignores server resources' optimization, so resource bottlenecks are likely to occur, reducing its throughput.

In terms of TO-DG, the throughput of group A is obviously lower than that of group B. Because the resource utilization of servers and links directly affects the delay of SFCR. When the resource utilization of servers and links increases, the delay of SFCRs on servers and links increases. Compared with group B, the resource utilization of servers and links in group A has to be lower, which can meet the delay requirements of SFCRs in group A. Therefore, when the available network resources are the same, TO-DG performs better in group B.

4) Comparison of Resource Utilization: Fig. 7 presents the cumulative distribution function (CDF) curves about the

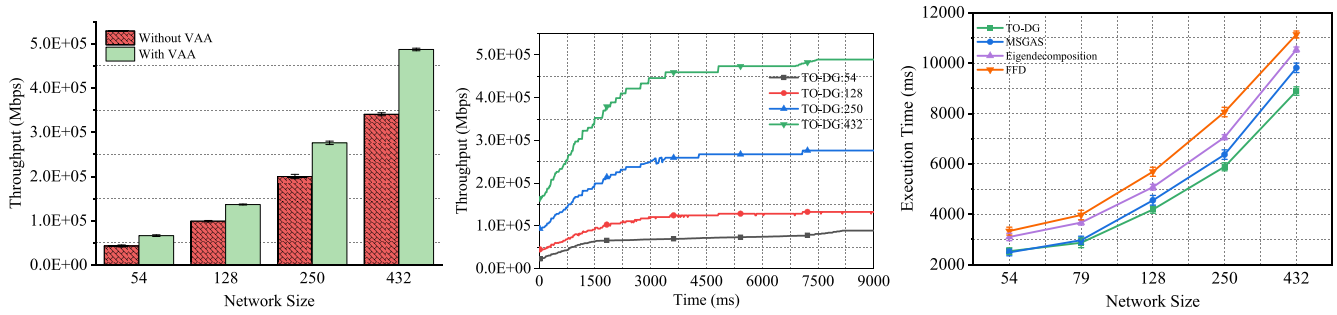
Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه



(a) Utilization of CPU Resource (b) Utilization of Memory Resource (c) Utilization of Bandwidth Resource

Fig. 7. CDF of Different Resource Utilization.



(a) With VAA versus without VAA. (b) Network Throughput over Time. (c) Comparison of Execution Time.

Fig. 8. With VAA versus without VAA, Network Throughput over Time, and Comparison of Execution Time.

resource utilization of different approaches. We calculate the CDF of resource utilization as

$$CDF(k_0) = \frac{N(k \le k_0)}{R_t}, \tag{19}$$

where R_t indicates the number of total activated servers or links, and $N(k \le k_0)$ denotes the number of servers or links those have a resource utilization k under the threshold k_0.

Figs. 7(a) and 7(b) demonstrates the CPU and memory utilization of different solutions. For TO-DG, we can observe that TO-DG shows better resource utilization, and 80% of the activated servers have CPU and memory utilization exceeding 70%, which is higher than the benchmarks. It owes to the consideration of the affinity groups when selecting servers for mapping SFCRs. Besides, the VAA algorithm in TO-DG transfers the VNFRs on servers with low resource utilization to those with high utilization, which helps enhance CPU and memory utilization. Eigendecompostion shows worse server resource utilization, only about 40% of the servers have resource utilization above 65%.

As shown in Fig. 7(c), TO-DG's bandwidth utilization is second only to Eigendecomposition, and about 60% activated links have bandwidth utilization exceeding 60%. For links that have utilization less than 55%, the results of TO-DG and MSGAS are close. TO-DG maps most parts of one SFCR to the same server, so the flows that inner the SFCRs are restricted in the servers, consuming fewer bandwidth resources. Eigendecomposition shows better bandwidth utilization, and 60% of the links have bandwidth utilization above 65%. Eigendecomposition tends to place the VNF instances used by one SFCR into different servers. The flows go through more links to accomplish an SFC, and then the bandwidth utilization is higher.

5) With VAA Versus Without VAA: In our proposed approach, the Affinity-based SFCR Mapping Algorithm (ASMA) and VNFR Adjusting Algorithm (VAA) work together. That is, ASMA calculates an intermediate result for SFCRs, which means the output of ASMA is not the final SFCR mapping scheme. Based on the intermediate result derived by ASMA, VAA adjusts the mapping position of some VNFRs and then derives the final mapping schemes. Fig. 8(a) shows the impact of VAA on the performance of the proposed approach.

The performance of the intermediate scheme (without VAA) is obviously lower than the complete scheme (with VAA). VAA increased the average throughput by approximately 39%. We can also observe that as the network size increases, the effect of VAA becomes more significant. The reason is that a larger network size means that VAA has more room for adjustment. When the network size increase to 432, VAA increased the throughput by approximately 43%.

6) Network Throughput Over Time: Fig. 8(b) shows the network throughput of TO-DG in different network sizes: 54, 128, 250, and 432. As time varies, the results in different network sizes all tend to be stable, which means the TO-DG approach can obtain effective VNF placement solutions. In different network sizes, TO-DG can converge in about 2000 ms, 3000 ms, 3500 ms, and 4500 ms, respectively. Moreover, TO-DG needs more time to converge when increasing network size. As network size increases from 54 to 432, TO-DG's converge time increases from about 2000 ms to 4500 ms. This

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه
puyamoshaver@gmail.com | شماره تماس : ۰۹۳۵۴۹۴۱۴۳۴

TABLE V
AVERAGE EXECUTION TIME

Topology	Gurobi	TO-DG
12 servers, 15 links	36.96s	0.686s
23 servers, 43 links	1208.23s	1.197s
79 servers, 147 links	∞	2.87s

figure also confirms that our proposed approach is scalable with network size.

7) Execution Time: In this part, we exploit Gurobi [12] to solve the ILP model and compare the average execution times of Gurobi with that of TO-DG. We use three sizes of network in the experiment. Table V shows the average execution times of Gurobi and TO-DG in three sizes of network. For the third topology, Gurobi crashes and fails to solve the problem, while TO-DG can generate the solution of the same problem within 3 seconds. As we can see, the execution time of TO-DG is several orders of magnitude faster than Gurobi, and TO-DG's performance is close to Gurobi's.

Fig. 8(c) demonstrates the execution time of the four solutions in different network sizes. We can perceive that TO-DG runs faster than the benchmarks for the same network size, and the execution times of TO-DG and MSGAS are close. As the network size becomes larger, the execution times of different solutions also increase, and their execution times are at the same level. It indicates that as the network size expands, these approaches need to take longer to derive VNF placement solutions.

VII. CONCLUSION

This paper studies the VNF placement problem for mapping SFCRs in NFV-enabled networks, aiming to maximize the throughput for SFCRs while guaranteeing their delay requirements. Then we propose a polynomial-time approach called TO-DG to solve this problem. In TO-DG, the affinity-based SFCR mapping algorithm first classifies the servers into different affinity groups and then maps SFCRs based on these groups. The VNFR adjusting algorithm is proposed to adjust VNFRs among servers, efficiently reducing the number of placed VNF instances and enhancing network resource utilization. Performance evaluation results demonstrate that TO-DG obtains a higher network throughput than the benchmarks in different scenarios and significantly enhances network resource utilization.

As for our future work, we plan to focus on VNF scheduling to optimize the delay of SFCRs and combine it with our VNF placement algorithms. We also plan to devise a more efficient VNF placement algorithm combined with a proactive resource allocation mechanism.

REFERENCES

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.
[2] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.
[3] G. Gibb, H. Zeng, and N. McKeown, "Outsourcing network functionality," in *Proc. ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 73–78.
[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architect. Protocols Comput. Commun.*, 2012, pp. 13–24.
[5] L. Chen, S. Patel, H. Shen, and Z. Zhou, "Profiling and understanding virtualization overhead in cloud," in *Proc. 44th Int. Conf. Parallel Process. (ICPP)*, Beijing, China, 2015, pp. 31–40.
[6] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual network function placement considering resource optimization and SFC requests in cloud datacenter," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1664–1677, Jul. 2018.
[7] C.-P. Bezemer and A. Zaidman, "Multi-tenant SaaS applications: Maintenance dream or nightmare?" in *Proc. Joint ERCIM Workshop Softw. Evol. (EVOL) Int. Workshop Principles Softw. Evol. (IWPSE)*, 2010, pp. 88–92.
[8] D. Li, P. Hong, K. Xue, and J. Pei, "Availability aware VNF deployment in datacenter through shared redundancy and multi-tenancy," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1651–1664, Dec. 2019.
[9] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and QoS-aware path allocation and VNF placement for service function chaining," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 1, pp. 374–388, Mar. 2019.
[10] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vNF placement for service chaining: Joint sampling and matching approach," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 172–185, Jan./Feb. 2020.
[11] M. Dieye *et al.*, "CPVNF: Cost-efficient proactive VNF placement and chaining for value-added services in content delivery networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, pp. 774–786, Jun. 2018.
[12] Gurobi. (2016). *Gurobi Optimizer Reference Manual, Version 7.0*. [Online]. Available: http://www.gurobi.com/documentation/7.0/refman.pdf
[13] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2nd Quart., 2019.
[14] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.
[15] D. Li, P. Hong, and K. Xue, "Virtual network function placement and resource optimization in NFV and edge computing enabled networks," *Comput. Netw.*, vol. 152, pp. 12–24, Apr. 2019.
[16] D. Qi, S. Shen, and G. Wang, "Towards an efficient VNF placement in network function virtualization," *Comput. Commun.*, vol. 138, pp. 81–89, Apr. 2019.
[17] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2179–2192, Oct. 2019.
[18] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve service chaining performance with optimized middlebox placement," *IEEE Trans. Services Comput.*, vol. 10, no. 4, pp. 560–573, Jul./Aug. 2017.
[19] Y. Liu, J. Pei, P. Hong, and D. Li, "Cost-efficient virtual network function placement and traffic steering," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, 2019, pp. 1–6.
[20] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and VNF placement in operator data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 3, pp. 530–543, Mar. 2019.
[21] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 643–655, Mar. 2019.
[22] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, no. 14, pp. 3330–3341, Jul. 15, 2016.
[23] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 533–546, Sep. 2016.
[24] S. Umeyama, "An Eigendecomposition approach to weighted graph matching problems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 10, no. 5, pp. 695–703, Sep. 1988.
[25] G. Sun, Y. Li, H. Yu, A. V. Vasilakos, X. Du, and M. Guizani, "Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks," *Future Gener. Comput. Syst.*, vol. 91, pp. 347–360, Feb. 2019.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.

Downloaded from https://iranpaper.ir
http://pooyamoshaver.com
Puyamoshaver@gmail.com
مشاوره انجام پروپزال و پایان نامه

[26] F. Chiti, R. Fantacci, F. Paganelli, and B. Picano, "Virtual functions placement with time constraints in fog computing: A matching theory perspective," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 3, pp. 980–989, Sep. 2019.
[27] V. Eramo, E. Miucci, M. Ammar, and 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.*, vol. 25, no. 4, pp. 2008–2025, Aug. 2017.
[28] V. Eramo, M. Ammar, and F. G. Lavacca, "Migration energy aware reconfigurations of virtual network function instances in NFV architectures," *IEEE Access*, vol. 5, pp. 4927–4938, 2017.
[29] L. Qu, C. Assi, and K. B. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.
[30] T. Wen, H. Yu, G. Sun, and L. Liu, "Network function consolidation in service function chaining orchestration," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kuala Lumpur, Malaysia, 2016, pp. 1–6.
[31] Z. Xu, W. Liang, A. Galis, and Y. Ma, "Throughput maximization and resource optimization in NFV-enabled networks," in *Proc. IEEE Int. Conf. Commun.*, Paris, France, 2017, pp. 1–7.
[32] Z. Xu, W. Liang, A. Galis, Y. Ma, Q. Xia, and W. Xu, "Throughput optimization for admitting NFV-enabled requests in cloud networks," *Comput. Netw.*, vol. 143, pp. 15–29, Oct. 2018.
[33] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis, "Efficient NFV-enabled multicasting in SDNs," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 2052–2070, Mar. 2019.
[34] M. Jia, W. Liang, and Z. Xu, "QoS-aware task offloading in distributed cloudlets with virtual network function services," in *Proc. 20th ACM Int. Conf. Model. Anal. Simulat. Wireless Mobile Syst. (MSWiM)*, 2017, pp. 109–116.
[35] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao, "Task offloading with network function requirements in a mobile edge-cloud network," *IEEE Trans. Mobile Comput.*, vol. 18, no. 11, pp. 2672–2685, Nov. 2019.
[36] Y. Ma, W. Liang, J. Wu, and Z. Xu, "Throughput maximization of NFV-enabled multicasting in mobile edge cloud networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 393–407, Feb. 2020.
[37] A. Dwaraki and T. Wolf, "Adaptive service-chain routing for virtual network functions in software-defined networks," in *Proc. ACM SIGCOMM Workshop Hot Topics Middleboxes Netw. Funct. Virtualization*, 2016, pp. 32–37.
[38] C.-J. Guo, W. Sun, Z.-B. Jiang, Y. Huang, B. Gao, and Z.-H. Wang, "Study of software as a service support platform for small and medium businesses," in *Proc. New Front. Inf. Softw. Serv.*, 2011, pp. 1–30.
[39] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.
[40] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for NFV chaining in packet/optical datacenters," *J. Lightw. Technol.*, vol. 33, no. 8, pp. 1565–1570, Apr. 15, 2015.
[41] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Hong Kong, 2015, pp. 1346–1354.
[42] A. Fischer, J. F. Botero, M. Duelli, D. Schlosser, X. Hesselbach, and H. de Meer, "ALEVIN—A framework to develop, compare, and analyze virtual network embedding algorithms," *Electron. Commun. ECEASST*, vol. 37, pp. 1–12, Jan. 2011.
[43] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 459–473.

Yi Yue (Student Member, IEEE) received the B.S. degree from Hebei Normal University, Shijiazhuang, China, in 2014. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing. His current research interests include network function virtualization, network slicing, and resource allocation managements.

Bo Cheng (Member, IEEE) received the Ph.D. degree in computer science from the University of Electronic Science and Technology of China in 2006. He is currently a Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include multimedia communications and services computing.

Meng Wang (Student Member, IEEE) received the B.S. degree from Beijing Jiaotong University, Beijing, China, in 2016. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing. His current research interests include network function virtualization, network slicing, and resource allocation managements.

Biyi Li received the B.S. degree from Jilin University, Changchun, China, in 2017. He is currently pursuing the Ph.D. degree in computer science and technology with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), Beijing, China. His research interests include network function virtualization, software-defined network, and cloud computing.

Xuan Liu (Student Member, IEEE) received the bachelor's degree in network engineering from the Beijing University of Posts and Telecommunications, where he is currently pursuing the Ph.D. degree in computer science and technology with the State Key Laboratory of Networking and Switching Technology. His research interests include cloud computing, service computing, mobile networks, and distributed computing.

Junliang Chen is a Professor with the Beijing University of Posts and Telecommunications. His research interests in the area of service creation technology. He was elected as a member of the Chinese Academy of Science in 1991, and the Chinese Academy of Engineering in 1994.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on March 04,2022 at 12:55:09 UTC from IEEE Xplore. Restrictions apply.