Maximizing Throughput of Delay-Sensitive NFV-Enabled Request Admissions via Virtualized Network Function Placement

Meitian Huang, Weifa Liang, *Senior Member, IEEE*, Yu Ma, and Song Guo, *Senior Member, IEEE*

Abstract—Network Function Virtualization (NFV) has attracted significant attention from both industry and academia as an important paradigm change in network service provisioning. Most existing studies on admissions of NFV-enabled requests focused on deploying dedicated Virtualized Network Function (VNF) instances to serve each individual request without exploring the sharing of VNF instances among multiple user requests. However, with ever-growing demands of user services, exclusive usages of VNF instances in most networks drastically degrade the network performance and largely under-utilize the VNF instance resources. In this paper, we jointly explore two different VNF instance scaling techniques, horizontal scaling and vertical scaling techniques, to improve the network throughput while minimizing the operational cost of the network, where horizontal scaling that migrates some existing VNF instances from their current locations to new locations to allow the VNF instances to be shared by multiple requests to reduce the resource consumption and operational cost of the network, and vertical scaling that instantiates new VNF instances to meet the demands of new request admissions if existing VNF instances sharing becomes more expensive or the end-to-end delay requirements of currently executing requests will be violated. We first propose a unified framework of maximizing the network throughput, by admitting as many as NFV-enabled requests while meeting the end-to-end delay requirements of the admitted requests. We then provide an Integer Linear Programming (ILP) solution for the problem when the problem size is small. Otherwise, we devise an efficient algorithm through a non-trivial reduction that reduces the problem to the minimum-weight feedback arc set problem and the generalized assignment problem (GAP). We finally conduct experiments to evaluate the performance of the proposed algorithm. Experimental results demonstrate that the proposed algorithm outperforms a baseline algorithm and achieves a performance on a par with its optimal ILP solution.

Index Terms—Network function virtualization, throughput maximization, scheduling algorithms, VNF instance sharing, resource allocation, end-to-end delay requirement, NFV-enabled request admission, VNF instance horizontal and vertical scalings

1 Introduction

Network Function Virtualization (NFV) has attracted significant attention from both industry and academia as an important paradigm shift on network service provisioning. Under this new NFV architecture, a *service chain* that consists of various network functions such as Firewall, Intrusion Detection System (IDS), WAN optimizer, and Deep Packet Inspection (DPI) can be decomposed into a set of Virtualized Network Functions (VNFs) that are implemented as software components in off-the-shelf physical servers [30], and these VNF implementations are often referred to as *VNF instances*. Each VNF instance can be relocated or instantiated at different locations (servers) in a network without necessarily purchasing and installing new hardware. By decoupling network functions from their traditional dedicated hardware implementation, NFV has great potentials to significantly reduce the operating expenses (OPEX) and capital expenses (CAPEX) of network service providers. NFV also facilitates the deployment of new network function services with agility and faster time-to-value [30].

Most existing studies focused on the optimal placement of VNF instances under a specific optimization objective while meeting user-specified resource demands and Quality of Service (QoS) requirements [33], [37], [38], or on historical VNF instance demand patterns or predictions [3], [22], yet little attention has been paid on maximizing network throughput via VNF instance vertical and horizontal scalings at the same time. In this paper, we focus on maximizing the network throughput by admitting as many as NFV-enabled user requests through jointly exploring VNF instance horizontal and vertical scalings, where *horizontal scaling* is to migrate existing VNF instances from their current locations (servers) to other locations (servers) to meet the delay requirements of both currently executing and newly admitted requests, and *vertical scaling* is to instantiate new VNF instances for newly admitted requests if horizontal scaling is infeasible. Fig. 1 is an illustrative example of vertical and horizontal scalings, where network G consists of six nodes $v_1, v_2, \ldots, v_6$. Given two requests r_1 and r_2 with each having its end-to-end delay requirement, request r_1 requires data traffic from its source s_1 (co-located with v_1) to its destination t_1

- *M. Huang, W. Liang, and Y. Ma are with the Research School of Computer Science, The Australian National University, Canberra, ACT 2601, Australia. E-mail: {Meitian.huang, yu.ma}@anu.edu.au, wliang@cs.anu.edu.au.*
- *S. Guo is with the Hong Kong Polytechnic University Shenzhen Research Institute and Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: song.guo@polyu.edu.hk.*

Manuscript received 28 Dec. 2018; revised 26 Mar. 2019; accepted 2 May 2019. Date of publication 10 May 2019; date of current version 6 Dec. 2021. (Corresponding author: Weifa Liang.)
Recommended for acceptance by C. Wu.
Digital Object Identifier no. 10.1109/TCC.2019.2915835

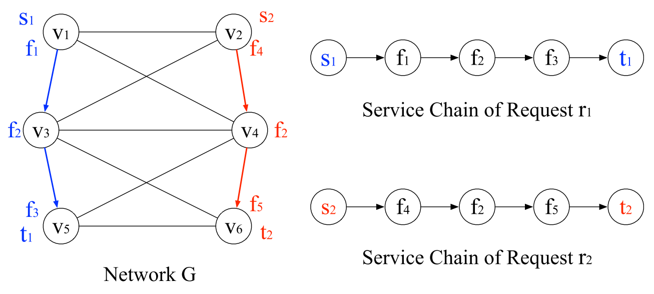


Fig. 1. An example network G with six nodes and two requests r_1 and r_2.

(co-located with v_5) to traverse network functions f_1, f_2, and f_3, while request r_2 requires data traffic from its source s_2 (co-located with v_2) to its destination t_2 (co-located with v_6) to traverse network functions f_4, f_2, and f_5. Suppose that each node in G is co-located with a server (or a server cluster) that can accommodate VNF instances. Assume that request r_1 is admitted first, a VNF instance of network function f_2 is instantiated in v_3 for the request, because v_3 is close to the source and destination of request r_1. As a result, when admitting request r_2 that also requires network function f_2, the network service operator can either (a) let requests r_1 and r_2 share the existing VNF instance of f_2 in node v_3, or (b) migrate the existing VNF instance from node v_3 to another location closer to the source and destination of request r_2, or (c) instantiate a new VNF instance of f_2 in node v_4. Apparently, (a) is feasible only if the data traffic routing of request r_2 through v_3 can satisfy its end-to-end delay requirement and the VNF instance of f_2 in v_3 can handle the data traffic of both requests r_1 and r_2 simultaneously; (b) is applicable only if the end-to-end delay requirement of request r_1 will not be violated after migrating the VNF instance serving r_1 from v_3 to v_4; and (c) should be applied if the first two options are infeasible, because it creates an additional VNF instance and may incur a higher cost.

In this paper, we focus on network throughput maximization by dynamically admitting as many as NFV-enabled requests with end-to-end delay requirements into a network while minimizing the operational cost of the network service provider through joint considerations of VNF instance horizontal and vertical scalings. This problem poses two major challenges. One is to how to admit a new request to meet its service chain, and for each VNF in its service chain, there are three options: sharing existing VNF instances, vertical scaling, and horizontal scaling, which option should be adopted? When different user requests demand the same type of network functions, these requests can be admitted by sharing the same VNF instance in the same server if that instance is able to handle the data traffic of all the requests. Despite lower the operational cost by sharing existing VNF instances among requests, sometimes vertical and horizontal scalings are necessary to avoid the violation of end-to-end delay requirements of some of the requests. In vertical scaling, each user request can be satisfied by instantiating a new instance for each of its network functions individually. However, vertical scaling may incur a high operational cost due to higher resource consumption than needed. On the other hand, in horizontal scaling, user requests can be admitted by migrating existing VNF instances from their current locations to new locations, incurring a one-time scaling cost. Another challenge is how to perform dynamic admissions of incoming requests in a non-disruptive way. When performing either horizontal or vertical scaling, we must ensure that both resource demands and end-to-end delay requirements of all currently executing requests are not violated, yet different admission strategies of admitting incoming requests heavily impact on the currently executing requests. Allowing arriving requests to share the same VNF instance with currently executing requirements will not affect the end-to-end delay experienced by executing requests, provided that the total data traffic of requests sharing the same VNF instance does not exceed the processing capacity of the VNF instance. In comparison, horizontal scaling has a significant impact on currently executing requests and arriving requests, as the VNF instances used by executing requests will be migrated from their current locations to new locations and data traffic of executing requests will be routed via different paths. On the other end of the spectrum, vertical scaling has minimum influence on currently executing requests, because arriving requests are admitted by instantiating new VNF instances in vertical scaling.

In this paper, we will address the aforementioned challenges. To the best of our knowledge, we are the first to provide a unified framework for dynamic NFV-enabled request admissions, by jointly performing both vertical and horizontal scalings, to maximize the number of requests admitted while minimizing the total operational cost of admitting the requests. We also devise an efficient algorithm for the problem.

The main contributions of this paper are given as follows. We first formulate a novel throughput maximization problem by jointly taking into account both vertical and horizon scalings on VNF instances, and formulate an Integer Linear Program (ILP) solution to the problem. The proposed ILP solution however is only applicable when the problem size is small. We then devise an efficient, scalable algorithm for the problem through a non-trivial reduction. We finally evaluate the performance of the proposed algorithm through experimental simulations, based on real and synthetic network topologies. Experimental results demonstrate that the proposed algorithm is very promising, compared to a baseline algorithm and the optimal ILP solution. The results also demonstrate the effectiveness of the joint consideration of both vertical scaling and horizontal scaling.

The rest of the paper is organized as follows. Section 2 will review related work. Section 3 will introduce the system model, notions and notations, and the problem definition. Section 4 will formulate an Integer Linear Programming solution for the problem. Section 5 will propose an efficient algorithm for the problem. Section 6 will conduct a performance evaluation of the proposed algorithm, and Section 7 will conclude the paper.

2 Related Work

While network functions are widely used to guarantee security and performance of routing data packet traffic in contemporary computer networks, the deployment of traditional hardware network functions incurs high capital investment and operational cost [34], [35]. To tackle these issues, recent efforts on new frameworks and architectures for NFV [1], [13], [15], [28], [34], have been demonstrated as promising

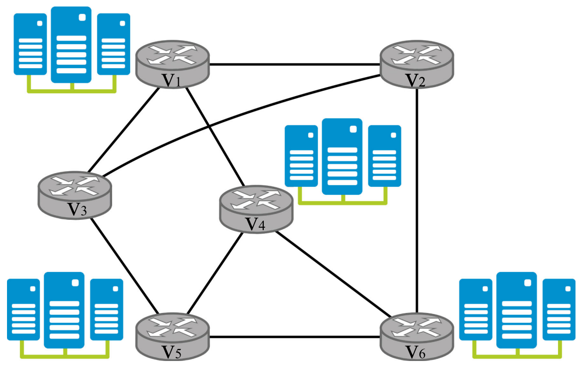


Fig. 2. An example network.

alternatives to traditional hardware by virtualized network functions in virtual machines. For example, Sekar et al. devised an architecture CoMb [34] that focused on software-based implementations of network functions on a shared hardware platform. Qazi et al. developed SIMPLE [31] that enforces high-level routing policies for networks function-specific traffic. Fayazbakhsh et al. proposed FlowTags [11], because traditional flow rules do not suffice in the presence of dynamic modifications performed by network functions. Martins et al. [28] introduced a virtualization platform to improve network performance by revising existing virtualization technologies to support the deployment of modular, virtual middleboxes on lightweight VMs.

Admitting NFV-enabled requests has been extensively studied in various settings. For example, Jia et al. [19], [20], Ma et al. [25], and Xu et al. [37], [38] devised the very first algorithms with performance guarantees for NFV-enabled unicasting and multicasting, respectively. However, the aforementioned studies considered the admission of each request separately from others without taking into account potential migrations of VNF instances. As a result, the network resources may be overloaded and no future requests can be admitted. Several studies of NFV migrations primarily dealt with the development of migration mechanisms [4], [5], [9], [14], [18], [22], [32]. For example, the authors in [9] provided one of the earliest studies of vertical scaling of VNF instances with the objective of minimizing the sum of energy consumptions of network function instances and the re-configuration cost of network instances. Specifically, with the increase or decrease of data traffic, the computing capacities of VNF instances increase or decrease accordingly. In [18], Xia et al. studied the problem of migrating network functions such that the migration cost, defined as the aggregated transferring traffic rate during the migration progress, is minimized. Carpio et al. [4], [5] recently tackled the NFV migration problem, by using replications in place of migrations, because of the adverse effects that migrations have on Quality of Service. They provided an Integer Linear Programming solution to the problem. While Kawashima et al. [22] provided a solution to the dynamic placement of VNFs in a network to follow the traffic variation, they did not take into account the cost of migrating VNFs. There are recent works in this topic too [12], [21], [24], [36]. For example, Fei et al. [12] proposed a proactive approach that provides extra VNF instances for overloaded network functions in advance, based on the estimation of future flow rates. They first adopted an efficient online learning method with the objective of minimizing the error in predicting the demands of different network functions. Based on the proposed method, the requested instances with variable processing capacities can be derived. Wang et al. [36] proposed online algorithms with the aims of minimizing the VNF provisioning cost for cases with one service chain and multiple service chains, respectively. Liu et al. [24] studied the optimization of service chains deployment for new users and readjustment of active users service function chains while taking into account resource consumption and operational overhead. Jia et al. [21] investigated the dynamic placement of VNF service chains across geo-distributed data centers in order to serve user requests with the objective of minimizing the operational cost of the network service provider. They proposed an efficient online algorithm for the problem based on the regularization approach. Ma et al. [26], [27] investigated dynamic request admissions with service delay and function chain requirements in a distributed cloud. The objective of the problem considered in [26], [27] is to maximize the profit collected by the service provider. Xu et al. [39] considered user request admissions with NFV service and QoS requirements by placing new VNF instances into clodlets and sharing existing VNF instances among requests [39].

We distinguish our work in this paper from the aforementioned works as follows. Existing studies only dealt with either VNF migrations [9], [18] or creating multiple VNF instances [4], [12], [21], [24]. We focus on NFV-enabled request admissions while meeting their end-to-end delay requirements. We aim to maximize the network throughput while minimizing the operational cost of the network operator, by jointly exploring VNF instance sharing, instantiations and migrations. We propose a unified framework and develop an efficient algorithm for the problem. It must be mentioned that this paper is an extended version of our conference paper [16].

3 Preliminaries

In this section, we first introduce the system model, notations and notions used in this paper. We then define the problem formally.

3.1 System Model

The service provider network is represented by an undirected graph $G = (V, E)$, where V is the set of switch nodes and E is the set of links that interconnect the nodes in V. A subset of switch nodes V_S ($\subseteq V$) is attached with servers that can implement network functions as VNF instances. Each server attached to switch $v \in V_S$ has computing capacity cap_v, and the connection between the server and its attached switch is by a high-speed optical cable, their communication bandwidth and communication delay usually are negligible. Each link $e = (u, v) \in E$ between two switch nodes has a transmission delay D_e $(= D_{u,v})$. We assume that switches in G are connected via high-capacity optical links so that the bandwidth capacity of the link is not a bottleneck. Fig. 2 demonstrates a service provider network G with a set of switch nodes $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and a subset of switch nodes $V_S = \{v_1, v_4, v_5, v_6\}$ that are attached with servers.

3.2 Virtualized Network Functions

The operator of network G offers L types of VNFs to serve different users. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_L\}$ be the set of VNFs and let c_i be the computing resource requirement of VNF $f_i \in \mathcal{F}$. An instance of network function f_i can be instantiated in a switch node $v \in V_S$ if the server attached to v has sufficient computing resource to accommodate the instance, i.e., its residual computing resource is no less than the computing resource demand c_i by a VNF instance of f_i. For a given network function f_i, there may exist multiple its VNF instances. We thus distinguish different instances of the same network function by denoting the jth instance of f_i as I_{i_j}. A VNF instance may be used to serve multiple user requests at the same time, yet *the maximum data packet processing rate* m_i that each instance of a network function f_i can process is given in advance. We assume that each VNF instance can serve at least one request. If existing VNF instances of network function f_i are inadequate to handle request demands, its additional VNF instances can be instantiated, and each newly instantiated one will incur a setup cost of C_i^{setup} with $1 \le i \le L$. Denote by $C_{ser}(i)$ the service cost of a VNF instance of network function f_i per time slot.

3.3 User Requests

Each *NFV-enabled user request* r_k consists of a traffic routing request from its source to its destination, a service chain, and the end-to-end delay requirement, where each data packet traffic must pass through each network function in the service chain, while the end-to-end delay of its routing path is no greater than its specified one. Formally speaking, each request r_k is defined as a quintuple $(s_k, t_k, b_k, d_k, SC_k)$, where s_k and t_k are the source and destination nodes, respectively, b_k is the request packet rate, d_k is the end-to-end delay requirement, and SC_k is the service chain of the request. Specifically, SC_k is a sequence of ℓ_k (≥ 1) network functions that request r_k requires, i.e., $SC_k = \langle f_{k_1}, f_{k_2}, \ldots, f_{k_{\ell_k}} \rangle$, where $f_{k_j} \in \mathcal{F}$ is the jth VNF in the service chain of request r_k. In other words, each data packet traffic of request r_k must pass through the VNF instances in SC_k in their specified order. Assuming that the data traffic of r_k is routed via a routing path $P_k = (v_1 = s_k, v_2, \ldots, v_n = t_k)$ in G between s_k and t_k, the end-to-end delay $D(P_k)$ on the routing path experienced by r_k is

$$D(P_k) = \sum_{i=1}^{n-1} D_{v_i, v_{i+1}}, \tag{1}$$

where $D_{v_i, v_{i+1}}$ is the transmission delay on link $(v_i, v_{i+1}) \in E$. We focus on network transmission delays experienced by requests because they are the dominating factor.

In order to admit an NFV-enabled request r_k, a routing path P_k in G between source s_k and destination t_k needs to be identified, such that (i) path P_k must traverse VNF instances of the specified sequence of network functions in $SC_k = \langle f_{k_1}, f_{k_2}, \ldots, f_{k_{\ell_k}} \rangle$; and (ii) the end-to-end transmission delay $D(P_k)$ according to Eq. (1) is no greater than its delay requirement d_k.

3.4 Dynamic Admissions of User Requests

We assume that time is equally slotted into *time slots*, and these time slots are numbered as $1, 2, \ldots, T$, where T is the length of a monitoring period. When a request r_k arrives, it will be either admitted or rejected in the beginning of the next time slot of its arrival.

Denote by $\mathcal{R}(t)$ the set of newly arrived requests in the beginning of time slot $t \in \{1, 2, \ldots, T\}$. Due to limited network resources, the network service provider of G may only accommodate a subset of requests in $\mathcal{R}(t)$ by allocating demanded network resources to them. Once a request is admitted, its data traffic will occupy or share the allocated resources it demanded for a certain amount of time. When its data traffic finishes, all resources occupied by its data traffic will be released back to the network for the admissions of other requests. Accordingly, let $\mathcal{A}(t)$ be the set of admitted requests that have not departed from the system in the beginning of time slot t with $\mathcal{A}(0) = \emptyset$. In other words, $\mathcal{A}(t)$ is the set of admitted requests that still occupy the system resources at least for time slot t. Similarly, for each network function $f_i \in \mathcal{F}$, denote by $\mathcal{I}_i(t)$ the set of VNF instances of f_i in the end of time slot $t-1$ with $\mathcal{I}_i(0) = \emptyset$.

As requests are dynamically admitted or departed, the amounts of residual resources in the beginning of different time slots are different. Let $RE_v(t)$ be the residual computing capacity of the server attached to $v \in V_S$ in the beginning of time slot t with $RE_v(0) = cap_v$.

3.5 Vertical and Horizontal Scalings

The deployment of VNF instances of a specific network function typically is based on information available at the time of its deployment. For instance, the initial deployment of VNF instances can be based on historical traffic data or previous requests that requested which types of network functions, the request data packet rates, their QoS requirements, and so on. When new requests arrive, either vertical scaling or horizontal scaling should be performed. However, it is worth noting that resource and delay requirements of existing user requests must not be violated, regardless of what scaling should be performed for a particular network function.

In the beginning of each time slot t, assume that some of VNF instances have been instantiated for request admissions in previous time slots. Some of the VNF instances can be shared to admit newly arrived requests in $\mathcal{R}(t)$, provided that (i) the resource requirements of existing admitted requests in $\mathcal{A}(t)$ are not violated; and (ii) the computing capacities of existing VNF instances are not violated. If either of the two conditions is not met, additional VNF instances need to be instantiated or existing VNF instances need to be migrated to other locations to meet the requirements of both currently executing requests and newly admitted requests.

Horizontal scaling of VNF instances is applicable if existing VNF instances can be migrated to new locations such that the resource requirements of both newly admitted requests in $\mathcal{R}(t)$ and currently executing requests in $\mathcal{A}(t)$ can be satisfied simultaneously.

Denote by $\mathcal{A}_{i,j}(t)$ $(\subseteq \mathcal{A}(t))$ the set of requests processed by the VNF instance I_{i_j} in the beginning of time slot t. As the location of a VNF instance I_{i_j} may change at different time slots, denote by $l_{i,j}(t)$ $(\in V_S)$ the location of instance I_{i_j} during time slot t. The VNF instance I_{i_j} may be migrated from $l_{i,j}(t-1)$ in the beginning of time slot t. *The cost of migrating an existing VNF instance* I_{i_j} from its current location $v_a^{i,j}$ to its destination location $v_b^{i,j}$ through a given path

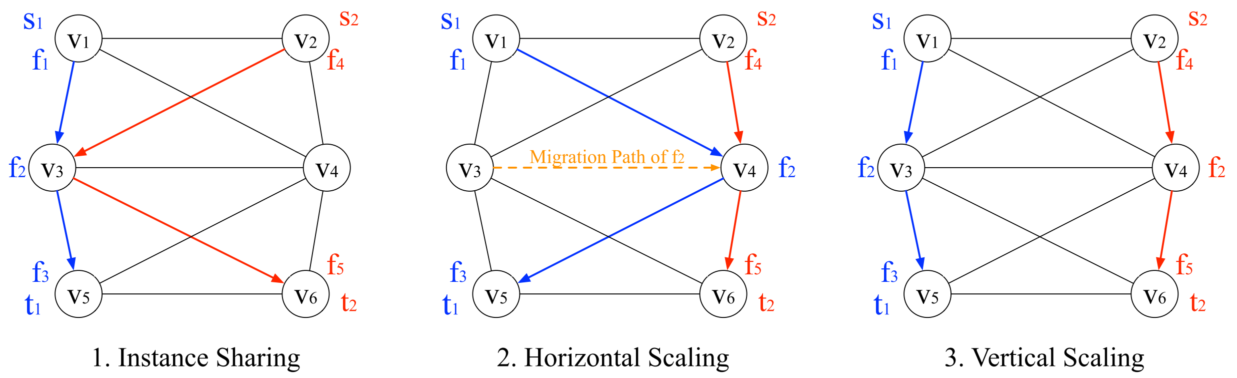


Fig. 3. Different options for admitting r_2 after request r_1 has been admitted in network G in Fig. 1.

$P^{i,j} = (v_a^{i,j} = v_0^{i,j}, v_1^{i,j}, \ldots, v_n^{i,j} = v_b^{i,j})$ is

$$C_{i,j}^{mig}(P^{i,j}) = \sum_{r_k \in \mathcal{A}_{i,j}(t)} b_k \cdot |P^{i,j}|, \tag{2}$$

where b_k is the packet rate (the bandwidth demand) of request r_k and $|P^{i,j}|$ is the number of links on path $P^{i,j}$, while $P^{i,j}$ usually is the shortest path in G from node v_i to node v_j with $1 \leq i, j \leq |V|$.

Notice that if VNF instance I_{i_j} is migrated from location $l_{i,j}(t)$ to location $l_{i,j}(t+1)$, the end-to-end transmission delays experienced by requests in $\mathcal{A}_{i,j}(t)$ may change because these requests need to be served by the VNF instance I_{i_j} in location $l_{i,j}(t+1)$, instead of location $l_{i,j}(t)$. Consequently, the traffic of these executing requests needs to be rerouted through different paths. To avoid any service disruption to currently executing requests in $\mathcal{A}_{i,j}(t)$, the migration of every VNF instance I_{i_j} should be performed only if none of the delay requirement of all admitted requests in it will be violated. However, sometimes due to the network characteristics or user requests, horizontal scaling cannot satisfy newly arrived requests that demand network function f_i, because it will violate the delay requirements of admitted requests. As a result, new instances of f_i must be instantiated at the expense of *the setup cost* of C_i^{setup}.

Take the two requests r_1 and r_2 in Fig. 1 for instance. Fig. 3 demonstrates different strategies for the admission of request r_2 after r_1 has been admitted: (1) instance sharing, i.e., r_1 and r_2 share the instance of f_2 in node v_3; (2) horizontal scaling, i.e., migrating the instance of f_2 from v_3 to v_4 in order to serve both requests r_1 and r_2; and (3) vertical scaling, i.e., instantiating a new instance of f_2 in node v_4 to serve request r_2 only.

3.6 The Operational Cost

The network service provider of G is interested in minimizing the network operational cost to maximize its profit. Recall that $\mathcal{I}_i(t)$ is the set of VNF instances of network function $f_i \in \mathcal{F}$ at the end of time slot t. Accordingly, $\mathcal{I}_i(t-1)$ is the set of VNF instances in network G at the beginning of time slot t. The network operational cost at each time slot t consists of the following two costs.

Service cost: Each VNF instance I_{i_j} in $\mathcal{I}_i(t)$ will have a service cost of $C_{ser}(i)$. The total service cost $SC(t)$ in time slot t thus is the sum of service costs of all VNF instances, i.e.,

$$SC(t) = \sum_{f_i \in \mathcal{F}} \sum_{I_{i_j} \in \mathcal{I}_i(t)} C_{ser}(i). \tag{3}$$

Scaling cost: Each VNF instance $I_{i_j} \in \mathcal{I}_i(t)$ is also associated with a scaling cost, which is defined as follows.

$$C_{scal}(I_{i_j}, t) = \begin{cases} 0, & \text{if } I_{i_j} \in \mathcal{I}_i(t-1) \text{ and} \\ & l_{i,j}(t) = l_{i,j}(t-1) \\ C_{i,j}^{mig}(P^{i,j}), & \text{if } I_{i_j} \in \mathcal{I}_i(t-1) \text{ and} \\ & l_{i,j}(t) \neq l_{i,j}(t-1) \\ C_i^{setup}, & \text{if } I_{i_j} \notin \mathcal{I}_i(t-1), \end{cases} \tag{4}$$

where $l_{i,j}(t)$ is the location of VNF instance I_{i_j} at the end of time slot t, Case 1 in Eq. (4) corresponds to the case in which no scaling is applied to instance I_{i_j}, i.e., instance I_{i_j} already exists in previous time slot $t-1$ and the location of I_{i_j} does not change; Case 2 in Eq. (4) corresponds to the case in which horizontal scaling is applied to instance I_{i_j}, i.e., instance I_{i_j} already exists in previous time slot $t-1$ yet its location changes; and Case 3 in Eq. (4) corresponds to the case in which vertical scaling is applied to instance I_{i_j}, i.e., instance I_{i_j} does not exist in previous time slot $t-1$.

The total scaling cost of all VNF instances in network G for time slot t is

$$C_{scal}(t) = \sum_{f_i \in \mathcal{F}} \sum_{I_{i_j} \in \mathcal{I}_i(t)} C_{scal}(I_{i_j}, t). \tag{5}$$

The total operational cost of network G in time slot t is

$$TC(t) = SC(t) + C_{scal}(t). \tag{6}$$

Table 1 lists the notations used in this paper.

3.7 Problem Definition

Given the defined service provider network $G = (V, E)$, a time slot $t \in \{1, 2, \ldots, T\}$, in which there are a set of executing requests $\mathcal{A}(t)$ and a set of VNF instances $\mathcal{I}_i(t-1)$ of network function $f_i \in \mathcal{F}$ at the end of time slot $t-1$, and a set of newly arrived NFV-enabled requests $\mathcal{R}(t)$ with end-to-end delay requirements, the *network throughput maximization problem via VNF instance scalings* in G is to admit as many as requests in $\mathcal{R}(t)$ by determining the set $\mathcal{I}_i(t)$ of VNF instances and their locations for every network function $f_i \in \mathcal{F}$, and the assignment of each admitted request to one or multiple VNF instances, while minimizing the total operational

TABLE 1
Table of Symbols

Notations	Descriptions
$G = (V, E)$	the service provider network
$V_S\ (\subseteq (V))$	the subset of switch nodes with servers attached
cap_v	the computing capacity of server attached to $v \in V_S$
D_e	the transmission delay of link $e \in E$
$\mathcal{F}\ (= \{f_1, f_2, \ldots, f_L\})$	the set of network functions
c_i	the computing resource demand of VNF $f_i \in \mathcal{F}$
$\mathcal{I}_i(t)$	the set of VNF instances of f_i in the end of time slot $t-1$
$I_{i_j}\ (\in \mathcal{I}_i(t))$	the jth instance of f_i
m_i	the maximum data packet processing rate that each instance of a network function f_i
C_i^{setup}	the setup cost of a new instance of f_i
$C_{ser}(i)$	the service cost of a VNF instance network function f_i per time slot
$C_{scal}(I_{i_j}, t)$	the scaling cost of all NFV instances in I_{i_j} in time slot t
$r_k\ (= (s_k, t_k, b_k, d_k, SC_k))$	a user request with resource s_k, destination t_k, request packet rate b_k, end-to-end delay requirement d_k, and service chain SC_k
$\mathcal{R}(t)$	the set of newly arrived requests in the beginning of time slot t
$\mathcal{A}(t)$	the set of admitted requests that have not departed from the system in the beginning of time slot t
$l_{i,j}(t)\ (\in V_S)$	the location of instance I_{i_j} during time slot t
$C_{i,j}^{mig}(P^{i,j})$	the cost of migrating an existing VNF instance I_{i_j} via a given path $P^{i,j}$

cost of the network service provider, subject to computing capacities on servers in G.

3.8 NP-Hardness of the Defined Problem

Lemma 1. *The network throughput maximization problem via VNF instance scalings in G is NP-hard.*

Proof. We show NP-hardness of the problem by a reduction from a special case of the Generalized Assignment Problem (GAP). Since this special case of the GAP is NP-hard [6], the network throughput maximization problem via VNF instance scalings is NP-hard as well.

Given a GAP instance consisting of a set $\mathcal{B}$ of bins, a set $\mathcal{I}$ of items, bin capacities $cap : \mathcal{B} \mapsto \mathbb{R}^+$, and the size of placing item $i \in \mathcal{I}$ in bin $b \in \mathcal{B}$: $size(i, b)$, the GAP is to assign a maximum subset $U \subseteq \mathcal{I}$ of items to the bins in $\mathcal{B}$ such that the capacity of each bin is not violated.

We can generate an instance of the network throughput maximization problem via VNF instance scalings as follows. We first construct network $G = (V, E)$. We create a node i for each item i in $\mathcal{I}$, and a node b for each bin b in $\mathcal{B}$. We also add a virtual node s that serves as the destination of all requests. In order to construct the edge set, we create a link between each node i and each node b, and a link from b to node s. That is, $V = \mathcal{I} \cup \mathcal{B} \cup \{s\}$ and $E = \{(i, b) \mid i \in \mathcal{I}, b \in \mathcal{B}\} \cup \{(b, s) \mid b \in \mathcal{B}\}$.

We then generate a set of requests $\mathcal{R}(t)$: For each item $i \in \mathcal{I}$, we add to $R(t)$ a request $r_k = \langle s_k, t_k, b_k, d_i, SC_k \rangle$, where s_k is set to the node $i \in V$, t_k is set to node s, the traffic rate b_k is $size(i, m)$, and $d_i = \infty$. Therefore, routing the set of requests $\mathcal{R}(t)$ into network G is an instance of the network throughput maximization via VNF instance scalings problem. □

4 Integer Linear Programming

In this section, we formulate the network throughput maximization problem via VNF instance scalings as an Integer Linear Program as shown in Fig. 4 For brevity, denote by $N(v)$ the set of neighbors of node v in G. The ILP includes the following decision variables.

$H_{i_j}^{u,v}$ is a decision variable of value 1 if VNF instance I_{i_j} is migrated from node u to node v and value 0 otherwise. $M_{i_j}^{u,v}(r, s)$ is a decision variable that is 1 if the migration of VNF instance I_{i_j} from node u to node v is routed through edge (r, s), and 0 otherwise.

V_j^v is a decision variable that has value 1 if an additional instance of VNF f_j is instantiated in node v.

$Y_{i,k}^v$ is a variable that has value 1 if the k-th network function required by request r_i is satisfied by an instance in switch node v and value 0 otherwise.

$W_i^k(u, v)$ is a decision variable that is 1 if edge (u, v) carries the traffic of request r_i after k-th network function in the service chain is applied and before $(k+1)$-th network function is applied; otherwise, the value is 0.

X_i is 1 if request r_i is admitted or 0 otherwise.

Constraint (8) calculates the total cost of horizontal scaling according to Eq. (2). Meanwhile, Constraint (9) calculates the total cost of vertical scaling. These calculations reflect with those defined in Eq. (4).

In order to capture the objective of the problem, i.e., maximizing the network throughput while minimizing the total operational cost of the network operator, we here introduce the concept of a *budget B* on the total operational cost. Specifically, Constraint (10) requires that the total operational cost of G, which is calculated according to Eq. (6), is no greater than a given budget B. The value of B can be initially set to a large value, and a tighter bound of B can be obtained later by binary search iteratively.

Constraint (11) enforces that switch v can be used to implement network function f_j required by a request r_i only if a new instance is instantiated in v or an existing instance is migrated to v.

Constraints (12) and (13) capture traffic changing at switch nodes that accommodate VNF instances that process traffic of requests and traffic conservation at non-destination switches. Specifically, if request r_i is processed at $v \in V_S$, then (i) exactly one incoming edge of v carries the unprocessed traffic and

$$maximize \quad \sum_{1\le i\le|\mathcal{A}(t)|} X_i, \tag{7}$$

subject to

$$C^H_{total} = \sum_{1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)|}\sum_{1\le j\le L}\sum_{u,v\in V_S,u\ne v}\sum_{r,s\in V} M^{u,v}_{i,j}(r,s) \tag{8}$$

$$C^V_{total} = \sum_{1\le j\le L}\sum_{v\in V_S} C^{setup}_j \cdot V^v_j \tag{9}$$

$$C^H_{total} + C^V_{total} + \sum_{f_i\in\mathcal{F}}\sum_{I_{i_j}\in\mathcal{I}_i(t)} C_{ser}(i) \le B \tag{10}$$

$$Y^v_{i,k} \le V^v_j + \sum_u H^{u,v}_{i_j} \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)|,\ 1\le j\le L,\ 1\le k\le l_i,\ v\in V \tag{11}$$

$$\sum_{u\in N(v)} (W^k_i(u,v) - W^k_i(v,u)) = Y^v_{i,k}, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)|,\ 1\le k\le l_i,\ SC_{i_k}=f_j,\ v\in V \tag{12}$$

$$\sum_{u\in N(v)} (W^{k+1}_i(v,u) - W^{k+1}_i(u,v)) = Y^v_{i,k}, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)|,\ 1\le k\le l_i,\ SC_{i_k}=f_j,\ v\in V \tag{13}$$

$$\sum_{u\in N(s_i)} W^0_i(u,s_i) = 0, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)| \tag{14}$$

$$\sum_{u\in N(t_i)} W^{l_i}_i(t_i,u) = 0, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)|,\ v\in V \tag{15}$$

$$\sum_{u\in N(s_i)} W^0_i(s_i,u) = X_i - Y^{s_i}_{i,1}, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)| \tag{16}$$

$$\sum_{u\in N(t_i)} W^{l_i}_i(u,t_i) = X_i - Y^{t_i}_{i,k}, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)| \tag{17}$$

$$\sum_{u\in N(s_i)} W^{l_i}_i(s_i,u) = Y^{s_i}_{i,K-1}, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)| \tag{18}$$

$$\sum_{u,v\in V}\sum_{0\le j\le l_i} \left(W^j_i(u,v)\cdot D_{(u,v)}\right) \le d_i, \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)| \tag{19}$$

$$\sum_{r_i\in\mathcal{R}(t)\cup\mathcal{A}(t)} c_i\cdot Y^v_{i,k} \le cap_v(t), \qquad \forall v\in V \tag{20}$$

$$\sum_{r_i\in\mathcal{R}(t)\cup\mathcal{A}(t)} b_i\cdot Y^v_{i,k} \le m_i, \qquad \forall v\in V \tag{21}$$

$$\sum_{w\in N(v)} M^{u,v}_{i_j}(w,v) - \sum_{w\in N(v)} M^{u,v}_{i_j}j(v,w) = 0, \qquad \forall I_{i_j}\in\mathcal{I}_i(t), 1\le j\le L \tag{22}$$

$$\sum_{w\in N(u)} M^{u,v}_{i_j}j(w,v) = H^{u,v}_{i_j}, \qquad \forall I_{i_j}\in\mathcal{I}_i(t), 1\le j\le L \tag{23}$$

$$\sum_{w\in N(v)} M^{u,v}_{i_j}j(u,w) = H^{u,v}_{i_j}, \qquad \forall I_{i_j}\in\mathcal{I}_i(t), 1\le j\le L \tag{24}$$

$$X_i = 1 \qquad \forall r_i\in\mathcal{A}(t) \tag{25}$$

$$X_i \in\{0,1\} \qquad \forall r_i\in\mathcal{R}(t) \tag{26}$$

$$Y^v_{i,k}\in\{0,1\} \qquad \forall 1\le i\le|\mathcal{A}(t)\cup\mathcal{R}(t)|,\ 1\le k\le L,\ v\in V \tag{27}$$

$$V^v_j\in\{0,1\} \qquad \forall 1\le j\le L,\ v\in V_S \tag{28}$$

$$V^v_j = 0 \qquad \forall 1\le j\le L,\ v\in V\setminus V_S \tag{29}$$

$$H^{u,v}_{i_j} = H^{v,u}_{i_j} = H^{u,u}_{i_j} = 0 \qquad \forall 1\le j\le L,\ v\in V\setminus V_S,\ u\in V \tag{30}$$

$$H^{u,v}_{i_j}\in\{0,1\} \qquad \forall 1\le j\le L,\ v,u\in V_S,\ v\ne u \tag{31}$$

$$W^j_i(u,v)\in\{0,1\}, \qquad \forall e\in E,\ i=1,\ldots,|\mathcal{R}(t)|, j \tag{32}$$

$$M^{u,v}_{i_j}(r,s)\in\{0,1\}, \qquad \forall u,v,r,s\in V \tag{33}$$

Fig. 4. An ILP formulation of the network throughput maximization via VNF instance scaling problem.

none of the outgoing edges of v carries the unprocessed traffic; and (ii) exactly one of the outgoing edges of v carries the processed traffic, and none of the incoming edges of v carries the processed traffic. Otherwise, if the traffic of r_i is not processed at switch $v \in V_S$ but goes through v, either (i) exactly one incoming edge and one outgoing edge of v carries the unprocessed traffic, or (ii) exactly one incoming edge and one outgoing edge of v carries the processed traffic.

Constraints (14) and (15) ensure that if the traffic of a given request r_i any source switch s_i and no traffic leaves the terminal switch t_i. Constraints (16) and (17) handle the cases where the traffic of a request v_i is processed at the source

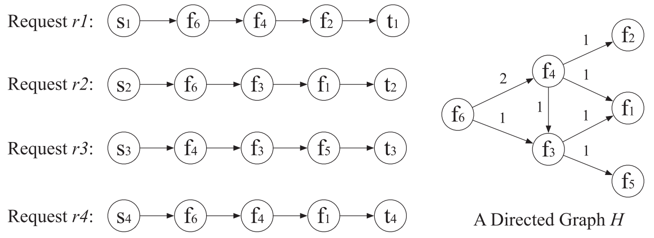

Request r1: s1 f6 f4 f2 t1
Request r2: s2 f6 f3 f1 t2
Request r3: s3 f4 f3 f5 t3
Request r4: s4 f6 f4 f1 t4
A Directed Graph H

Fig. 5. The four service chains demanded by four requests r_1, r_2, r_3, and r_4 can be represented by the directed graph H.

switch s_i or the terminal switch t_i. Constraint (18) handles the cases where VNF instances for request r_i are instantiated in its source node s_i.

Constraint (19) enforces the end-to-end delay requirement of each admitted request.

Constraints (20) and (21) model the computing capacity constraint of each switch $v \in V_{pm}$ and the processing capacity constraint of each VNF instance, respectively.

Constraints (22) to (24) deal with the migration paths. They ensure that the migration of VNF instance I_{i_j} follows a valid path.

Constraints (25) to (33) restrict the ranges of decision variables to 0 and 1, if applicable.

The formulated ILP serves for two purposes. One is that it can find an optimal solution to the problem when its size is small; another is that it serves as a benchmark for the performance evaluation of proposed algorithms.

5 Heuristic Algorithm

Since the problem is NP-hard and the proposed ILP solution takes prohibitive time when the problem size is large, it is only applicable when the problem size is small. In this section, we devise an efficient, scalable algorithm for the problem. We start by introducing the basic idea behind the proposed algorithm. We then detail the two stages of the proposed algorithm and the description of the algorithm itself. We finally analyze the time complexity of the proposed algorithm.

5.1 Overview of the Proposed Algorithm

The basic idea of the proposed algorithm is described as follows. In order to place VNF instances of different network functions to servers while meeting the end-to-end delay requirements of NFV-enabled requests, we are not just considering each request alone, we instead take all the VNF instances needed by all requests into consideration. To this end, we use a directed graph to model the relationships between the VNF instances of different network functions among the service chains of all being considered requests. We then prioritize the VNF instance scalings of different network functions. That is, which VNF instance should be considered first. We order the network functions in $\mathcal{F}$, by incorporating VNF dependency in service chains of requests in $\mathcal{R}(t) \cup \mathcal{A}(t)$. We finally perform VNF instance scalings by reducing this subproblem to a series of GAP instances, while each GAP instance will determine which type of scaling should be performed. The algorithm thus consists of two stages: (i) *prioritize the order of network functions in $\mathcal{F}$*; and (ii) *perform VNF instance vertical or horizontal scaling for each ordered network function in $\mathcal{F}$* one by one. In the following, we deal with these two stages in detail.

5.2 VNF Ordering

Each request r_k in $\mathcal{R}(t) \cup \mathcal{A}(t)$ has a service chain SC_k. The network functions in the service chain must be applied to each data packet traffic of the request in their specified order. Furthermore, the appearance order of a specific network function in the service chains of different requests is different, which makes prioritizing scaling orders of VNF instances of different network functions become difficult. For example, take four requests in Fig. 5 for consideration. If we consider scaling network function f_4 for request r_3 first, we may not fully exploit the fact that requests r_1 and r_2 require network function f_6 to be executed before network function f_4, and thus the VNF instances of these two network functions should be placed in reasonably close proximity to each other to meet the delay requirement of requests r_1 and r_2 and reduce their resource consumptions. As a result, considering VNF instance scaling for f_4 first will be not as effective as considering f_4 after f_6.

We thus utilize the constraints imposed by the service chains of requests to order network functions in $\mathcal{F}$ such that if a network function f_i appears before another network function f_j in the ordering, then the number of requests that require f_j before f_i is minimized. The detailed ordering of network functions is described as follows.

If network function f_i appears immediately before f_j in the service chain of request r_k, then there exists an ordering constraint between f_i and f_j. Let $f_i \prec_{r_k} f_j$ represent the ordering constraint between f_i and f_j in request r_k, e.g., $f_6 \prec_{r_1} f_4$ and $f_4 \prec_{r_3} f_3$ for Fig. 5. Then, given a set of requests in $\mathcal{R}$, ordering constraints inducted by these requests form a set $\mathcal{O} = \{f_{k_j} \prec_{r_k} f_{k_{j+1}} \mid r_k \in \mathcal{R}, 1 \leq j \leq l_k - 1\}$. For instance, given the service chains of request r_1, r_2, and r_3 in Fig. 5, the set of ordering constraints is $\{f_6 \prec_{r_1} f_4,\ f_4 \prec_{r_1} f_2,\ f_6 \prec_{r_2} f_3,\ f_3 \prec_{r_2} f_1,\ f_4 \prec_{r_3} f_3,\ f_3 \prec_{r_3} f_5\}$. Having the set of ordering constraints $\mathcal{O}$, we can identify an ordering of network functions. For instance, given three service chains in Fig. 5 with ordering constraints $\mathcal{O} = \{f_6 \prec_{r_1} f_4,\ f_4 \prec_{r_1} f_2,\ f_6 \prec_{r_2} f_3,\ f_3 \prec_{r_2} f_1,\ f_4 \prec_{r_3} f_3,\ f_3 \prec_{r_3} f_5\}$, one ordering of the network functions in $\mathcal{F}$ is $\langle f_6, f_4, f_3, f_2, f_1, f_5\rangle$. Notice that there exist many other orderings that satisfy the ordering constraints including $\langle f_6, f_4, f_3, f_2, f_5, f_1\rangle$ and $\langle f_6, f_4, f_3, f_1, f_2, f_5\rangle$.

Given a set of arrived requests $\mathcal{R}(t)$ and a set of admitted requests $\mathcal{A}(t)$, and a set of network functions $\mathcal{F}$, a weighted directed graph $H = (N, A; \omega)$ is constructed, where each network function $f_i \in \mathcal{F}$ is represented by a node in N, i.e., $N = \mathcal{F}$. For each request $r_i \in \mathcal{R}(t) \cup \mathcal{A}(t)$ with service chain $SC_i = \langle f_{i_1}, f_{i_2}, \ldots, f_{i_{\ell_i}}\rangle$ and $1 \leq j \leq \ell_i - 1$, if set A does not contain arc $\langle I_{i_j}, f_{i_{j+1}}\rangle$, one arc $\langle I_{i_j}, f_{i_{j+1}}\rangle$ is added to A and the weight on it is set to 1; otherwise, its weight is incremented by one. As a result, graph $H = (N, A; \omega)$ is a directed graph that may be or may not be a Directed Acyclic Graph (DAG) [7], because it is very likely that for some network functions f_i and f_j, network function f_i appears before network function f_j in the service chain of one request, yet f_i appears after f_j in the service chain of another request, resulting in a directed cycle in H. If such a cycle exists, an ordering of network functions in $\mathcal{F}$ is not achievable. Instead, all directed cycles in H must be removed to make the resulting graph a DAG. It thus is desirable to eliminate all directed cycles in H by removing a minimum weight set of arcs from H to minimize the number of requests whose network function order in their service chains are violated. However, identifying such a minimum weight set of removal arcs from H is NP-hard [8], we have the following lemma.

Lemma 2. *Given a set of admitted requests $\mathcal{A}(t)$, a set of arrived requests $\mathcal{R}(t)$, and a set of network functions $\mathcal{F}$, the auxiliary weighted, directed graph $H = (N, A; \omega)$ constructed may contain directed cycles. Eliminating all directed cycles by removing a minimum weight directed set of arcs from A is NP-hard.*

Due to NP-hardness of the problem of concern, we instead remove a set E' ($E' \subset A$) of arcs from H to make the resulting graph become a DAG, and E' can be obtained by applying an approximation algorithm due to Demetrescu et al. [8], and the approximate solution E' is bounded by the number of arcs of a longest simple cycle of the directed graph [8].

Denote by $H' = (N, A'; \omega)$ the DAG after the removal of arcs in E' from H. We then perform topological sorting on H'. As a result, each network function (corresponding a node in H) is sorted and all network functions in $\mathcal{F}$ have their sorted order.

The detailed procedure of ordering network functions in $\mathcal{F}$ is given in `Procedure` 1.

Procedure 1. Ordering Network Functions in $\mathcal{F}$

```
Input: a set of network functions $\mathcal{F}$, a set of arrived requests
    $\mathcal{R}(t)$, and a set of admitted requests $\mathcal{A}(t)$
Output: each network function in $\mathcal{F}$ is ordered at time slot $t$
    with $0 < t \le T$.
 1: Construct a directed weighted graph $H = (N, A; \omega)$ with
    $N = \mathcal{F}$, using the service chains of requests in $\mathcal{R}(t) \cup \mathcal{A}(t)$
    and $A = \emptyset$;
 2: for each request $r_i \in \mathcal{R}(t) \cup \mathcal{A}(t)$ with service chain $SC_i =$
    $\langle f_{i_1}, f_{i_2}, \ldots, f_{i_{\ell_i}} \rangle$ do
 3:   for $k \leftarrow 1$ to $\ell_i - 1$ do
 4:     if $A$ contains arc $\langle f_{i_k}, f_{i_{k+1}} \rangle$ then
 5:       $\omega(\langle f_{i_k}, f_{i_{k+1}} \rangle) \leftarrow \omega(\langle f_{i_k}, f_{i_{k+1}} \rangle) + 1$; /* increment the
          weight of an existing arc */
 6:     else
 7:       $A \leftarrow A \cup \{\langle f_{i_k}, f_{i_{k+1}} \rangle\}$; /* add a new arc to $A$ and set
          its weight to 1 */
 8:       $\omega(\langle f_{i_k}, f_{i_{k+1}} \rangle) \leftarrow 1$;
 9: Find an approximate, minimum weight set $E'$ ($\subset A$), by
    applying the approximation algorithm due to Demetrescu
    et al. [8] on graph $H$;
10: A DAG $H' = (N, A'; \omega')$ is obtained, by removing all arcs in
    $E'$ from $H$, where $A' = A \setminus E'$;
11: return a topological ordering of nodes (or network functions
    in $\mathcal{F}$) in $N$;
```

5.3 VNF Instance Placements and Migrations

We then deal with the second stage of the proposed algorithm. Having ordered network functions in $\mathcal{F}$, we now determine which scaling among the two scaling techniques should be applied to the VNF instances of each network function, and we examine the network functions one by one by its topological sorting order.

Let f_i be the network function that is currently being examined. We determine which scaling should be applied to a VNF instance of f_i, by constructing an instance of the Generalized Assignment Problem that is defined as follows.

Given a set of items $\mathcal{I}$ and a set of bins $\mathcal{B}$, where each bin $b \in \mathcal{B}$ has a capacity $cap(b)$, each item $i \in \mathcal{I}$ has a size of $size(i, b)$ and a profit $profit(i, b)$ if item i is placed in bin b, the problem is to assign a subset of items U ($\subseteq \mathcal{I}$) to bins $\mathcal{B}$ such that the total profit of the items in U is maximized, while the sum of sizes of placed items in each bin is no more than the capacity of the bin.

We construct an instance of the GAP for the currently considering network function f_i as follows. Each existing VNF instance I_{i_j} of f_i is represented by a bin with capacity $cap = m_i$. Each bin representing I_{i_j} corresponds to sharing the existing instance I_{i_j} of f_i. For each node $u \in V_S$ with sufficient residual computing capacity to accommodate an instance of f_i, a bin $\mathcal{V}^i_u$ is added to a collection $\mathcal{B}$ of bins, representing instantiating a new VNF instance at the node u. The capacity of each of the bins is $cap(\mathcal{V}^i_u) = m_i$.

For each existing VNF instance I_{i_j} at node u ($\in V_S$) and $v \in V_S \setminus \{u\}$, $\mathcal{H}^i_{u,v,j}$ is added to the set of bins $\mathcal{B}$. Note that migrating an existing VNF instance I_{i_j} of f_i may violate the end-to-end network transmission requirements of currently executing requests. Thus, bin $\mathcal{H}^i_{u,v,j}$ is added only if migrating VNF instance I_{i_j} from node u to node v does not violate the end-to-end network transmission requirements of executing requests in $\mathcal{A}(t)$ that use I_{i_j}. The capacity of bin $\mathcal{H}^i_{u,v,j}$ is $cap(\mathcal{H}^i_{u,v,j}) = m_i$.

Each request r_k in $\mathcal{R}(t) \cup \mathcal{A}(t)$ with SC_k containing f_i is represented as an item r_k. The size $size(r_k, b)$ is the traffic rate b_k of request r_k for all bins $b \in \mathcal{B}$. Notice that GAP is a maximization problem, whereas the objective of the network throughput maximization problem via VNF instance scalings is to admit as many as requests while minimizing the total operational cost. The profit $profit(r_k, b)$ of putting item r_k into bin $b \in \mathcal{B}$ thus is $profit(r_k, b)$, which is defined as follows.

If bin b is of the form I_{i_j}, which represents sharing the existing VNF instance I_{i_j}, then $profit(r_k, I_{i_j})$ is set to 1, because sharing an existing VNF instance incurs little additional costs and it should be highly encouraged. If bin b is of the form $\mathcal{V}^i_u$, which represents instantiating a new VNF instance at u, then $profit(r_k, \mathcal{V}^i_u)$ is the reciprocal of the setup cost of VNF instances for network function f_i, C^{setup}_i, i.e., $profit(r_k, \mathcal{V}^i_u) = 1/C^{setup}_i$. If bin b is of the form $\mathcal{H}^i_{u,v,j}$, which represents migrating the existing VNF instance from u to v, then $profit(r_k, \mathcal{H}^i_{u,v,j})$ is the reciprocal of the migration cost as given by Eq. (2).

The rationale behind the construction of a GAP instance for each network function is that it aims to maximize the number of requests admitted while minimizing the total operational cost of the network operator. The profit setting reflects the costs associated with different scaling options defined in Eq. (5). Thus, the solution to the GAP instance, in terms of assignments of a subset of items to bins, corresponds to a solution in which as many as requests are admitted and the total operational cost of their admissions is minimized.

Given each network function $f_i \in \mathcal{F}$, its corresponding instance of the GAP can be solved, using an approximation algorithm with a $\frac{1}{2+\epsilon}$-approximation ratio due to Cohen et al. [6], where ϵ is a constant with $0 < \epsilon \le 1$. Let S be the solution delivered by the approximation algorithm, which consists of pairs of admitted requests and bins. If request r_k is assigned to a bin representing sharing an existing VNF instance I_{i_j}, then the request should share the existing VNF instance I_{i_j}; (ii) if the request is placed in a bin $\mathcal{V}_u^i$, which represents a VNF instance instantiation of f_i at node u, then a new VNF instance of f_i should be instantiated at node u; (iii) otherwise, if it is placed in a bin $\mathcal{H}_{u,v,j}^i$, which represents horizontal scaling from u to v, then the VNF instance I_{i_j} in u should be migrated to v. If request r_i will not be assigned to any bin, then the request should be rejected.

5.4 Algorithm

The aforementioned process handles the instance placements and migrations of one network function f_i at each time. By repetitively applying the process for every network function in its topological sorting order, we can consider VNF instance scaling of all network functions in $\mathcal{F}$. Note that resource allocations for vertical (VNF instance instantiation) and horizontal scalings (VNF instance migration) are not actually performed on G until all network functions have been examined. Instead, in each iteration of the process, we work on a copy of the original network.

After iteratively examining all network functions, we then calculate the end-to-end network transmission delays of all involved requests. If the end-to-end delay requirement of a request $r_k \in \mathcal{R}(t)$ is not met or the algorithm fails to identify a VNF instance for one or more network functions in its service chain SC_k, request r_k will be rejected, and its related scalings will not be performed unless those scalings are also demanded by the other requests.

Due to the construction of the GAP instance for each network function f_i, request r_k may be assigned to a bin that represents either an instance sharing, a vertical scaling, or a horizontal scaling. If r_k is assigned to a bin that represents either vertical scaling or horizontal scaling and r_k is the only request that is assigned to the bin, then the scaling corresponds to the bin will not be performed, because r_k should be rejected and no scaling should be applied for the sake of it. Note that the horizontal scaling of existing VNF instances should not violate the end-to-end delay requirements of all admitted requests in $\mathcal{A}(t)$.

To calculate the end-to-end delay of every request r_k in $\mathcal{A}(t)$ experienced at each VNF instance of its service chain, and if this delay is greater than its delay requirement d_k, the violation occurs due to the VNF instance migrations of network functions demanded by request r_k. To resolve this potential violation on the end-to-end delay requirement of request r_k, for each VNF instance that is used by r_k needs to be migrated from its current location u to a new location v, the VNF instance is not migrated to v, and a new VNF instance of the network function is instantiated in v instead.

The detailed algorithm is given in `Algorithm` 1.

5.5 Analysis of the Proposed Algorithm

In the following, we first show that the solution delivered by `Algorithm` 1 is feasible. We then analyze its time complexity.

Theorem 1. *Given a network $G = (V, E)$ with a set V of switches and a set E of links, a subset $V_S \subseteq V$ of switches with attached servers to implement VNF instances, a set of network functions $\mathcal{F}$ provided by the network, a set of admitted requests $\mathcal{A}(t)$ that are still being executed in G in the beginning of time slot t, and a set of newly arrived requests $\mathcal{R}(t)$, there is an algorithm, `Algorithm 1`, for the network throughput maximization problem via VNF instance scalings, which delivers a feasible solution in $O(\ell_{\max}(|\mathcal{R}(t)| + |\mathcal{A}(t)|) + L^3 + L|V|^2 \cdot \frac{|\mathcal{R}(t) \cup \mathcal{A}(t)|^3}{\epsilon})$ time, where L is the number of network functions provided by G, i.e., $L = |\mathcal{F}|$, $\ell_{\max}$ is the maximum length of service chains of all requests, i.e., $\ell_{\max} = \max\{\text{the length } \ell_k \text{ of the service chain of } \; r_k \mid r_k \in \mathcal{A}(t) \cup \mathcal{R}(t)\}$, and ϵ is a given constant with $0 < \epsilon \leq 1$.*

Proof. The solution delivered by `Algorithm` 1 is feasible because each instance of the GAP is constructed from the subgraph of G that only includes the resources with sufficient residual capacities for request admissions. For instance, a bin that represents vertical scaling at node u is added only if u has sufficient residual computing capacity to accommodate an additional VNF instance. Furthermore, the construction of each GAP instance ensures that the capacity constraints of servers and VNF instances are not violated. Consequently, the scaling and admission decisions based on solving each the GAP instance are feasible.

We now analyze the running time of `Algorithm` 1 as follows. Recall that `Algorithm` 1 consists of two stages: (i) finding an ordering of network functions in $\mathcal{F}$ by constructing an auxiliary directed graph and eliminating all directed cycles in the auxiliary graph; and (ii) constructing an instance of the GAP and finding a solution for each network function in its topological order. in the resulting auxiliary graph.

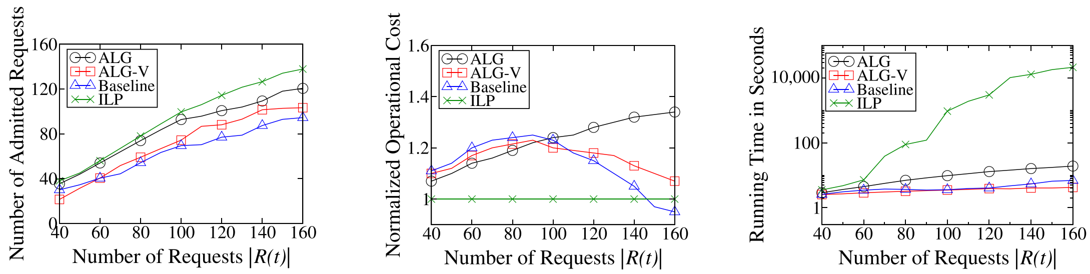
Stage (i) takes $O(\ell_{\max}(|\mathcal{R}(t)| + |\mathcal{A}(t)|) + L^3)$ time, because the procedure examines the service chain of every request to construct the auxiliary graph $H = (N, A; \omega)$, which results in $O(\ell_{\max}(|\mathcal{R}(t)| + |\mathcal{A}(t)|))$ running time, and removing cycles in H using the algorithm due to Demetrescu et al. [8] takes $O(|N| \cdot |A|) = O(L^3)$ time. The running time of Stage (ii) is dominated by the time required to solve a GAP instance for each of L network functions in $\mathcal{F}$. Given $O(|\mathcal{R}(t)| + |\mathcal{A}(t)|)$ items and $O(|V_S|^2) = O(|V|^2)$ bins, the algorithm due to Cohen et al. [6] takes $O(|V|^2 \cdot \frac{|\mathcal{R}(t) \cup \mathcal{A}(t)|^3}{\epsilon})$ time for each of L network function. The theorem thus holds. □

6 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm through experimental simulations using both real and synthetic network topologies. We start with the experimental environments, and then evaluate the performance of the proposed algorithm.

6.1 Experimental Environment

We adopt both real network topologies [23] and synthetic network topologies [2] in the simulations. The real network topologies are adopted from the Internet Zoo Project by Knight et al. [23], which is an ongoing initiative to collect data network topologies of different countries. We also adopt the widely used Barabási-Albert model [2], which can generate sythentic networks that follow the well-known network characteristic - scale-free, i.e., the degree distribution in a network typically follows a power law. Both sets of topologies are widely used in evaluating algorithm performance.



(a) The number of admitted requests (b) The normalized operational cost (c) The running time

Fig. 6. Performance of different algorithms within one time slot, using a real network topology with 40 nodes.

Algorithm 1. An Efficient Heuristic for Admitting a Set of Requests $\mathcal{R}(t)$ into G at Time Slot t

```
Input: a service provider network $G = (V, E)$, a set of network
    functions $\mathcal{F}$, a set of user requests $\mathcal{R}(t)$, and a set of admitted
    requests $\mathcal{A}(t)$ that are still executing in the network
Output: If each request in $\mathcal{R}(t)$ should be admitted and the
    VNF instances used for each admitted request
1: $\mathcal{R}'(t) \leftarrow \mathcal{R}(t)$; /* the set of requests that have not been
   marked as non-admitted */
2: for each request $r_k$ in $\mathcal{R}'(t)$ do
3:     Associate $r_k$ with an attribute $r_k.scalings$, which is the set
       of scalings to be performed for request $r_k$, and initialize
       the attribute $r_k.scalings$ to $\emptyset$;
4: Order network functions in $\mathcal{F}$ by invoking Procedure 1;
5: for each network function $f_i$ in the sorted order do
6:     Let $\mathcal{R}_i(t)$ be the subset of requests in $\mathcal{R}'(t) \cup \mathcal{A}(t)$ that
       require network function $f_i$;
7:     Construct an instance of the GAP with the set of items
       representing requests $\mathcal{R}_i(t)$ and the set of bins representing
       different scaling options;
8:     Let $S$ be an approximate solution to the GAP instance, by
       invoking the algorithm due to Cohen et al. [6];
9:     for each request $r_k$ in $\mathcal{R}_i(t)$ do
10:        if request $r_k$ is not assigned to a bin in the solution
           obtained in Step 8 then
11:            $\mathcal{R}'(t) \leftarrow \mathcal{R}'(t) \setminus \{r_k\}$; /* Mark request $r_k$ as a non-
               admitted request */
12:        else
13:            Add to $r_k.scalings$ the scaling corresponding to the
               bin to which request $r_k$ is assigned in the obtained
               solution;
14: for each request $r_k \in \mathcal{R}'(t)$ do
15:     if the end-to-end requirement of $r_k$ calculated according
        to Eq. (1) is not met then
16:         $\mathcal{R}'(t) \leftarrow \mathcal{R}'(t) \setminus \{r_k\}$;
17: return the set of scalings $r_k.scalings$ to be performed for
    every request $r_k$ in $\mathcal{R}'(t)$;
```

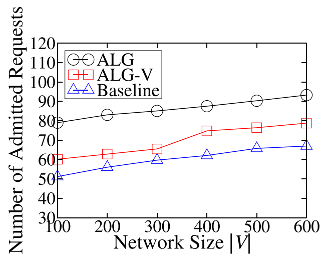
The parameter settings are consistent with previous studies, including the number of servers [17], the computing capacity of each server and link delays [18], and types of network functions and their computing resource demands [9], [17]. In the case of a single time slot, we assume that some VNF instances are randomly placed in the network already. Each request $r_i \in \mathcal{R}(t)$ is generated randomly by choosing two nodes in V as source s_i and destination t_i, and assigning its traffic rate b_i and end-to-end delay requirement d_i as per [17]. The generation of the service chain SC_i of each request r_i is consistent with the one in [9], [18]. The default accuracy parameter ϵ in solving the GAP is set at 0.1. The running time is obtained based on a machine with a 3.40 GHz Intel i7 Quad-core CPU and 16 GB RAM.

We evaluate the performance of the proposed algorithm against a baseline algorithm that is inspired by the devised algorithms in [10], [29]. The baseline algorithm is described as follows. For every arriving request $r_i \in \mathcal{R}(t)$, the greedy algorithm constructs a multi-stage graph, in which each stage corresponds to a network function in the service chain SC_i of request r_i, and then the baseline algorithm finds a shortest path in the multi-stage graph from s_i to t_i. Additionally, we evaluate the impact of horizontal scaling by running a variant of `Algorithm` 1 without horizontal scaling, i.e., the algorithm does not migrate any existing VNF instances. We refer to the ILP solution, `Algorithm` 1, `Algorithm` 1 with vertical scaling and without horizontal scaling, and the greedy algorithm as `ILP`, `ALG`, `ALG-V`, and `Baseline` respectively. Each value in figures is the average of the results of 30 trials.

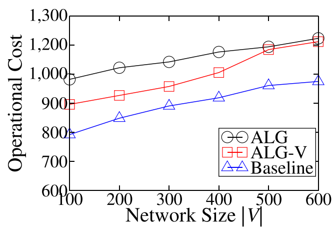
6.2 Performance Evaluation of Different Algorithms within a Single Time Slot

We first study the performance of different algorithms at a single time slot.

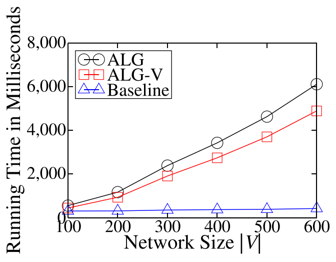
Fig. 6a shows the number of requests admitted by different algorithms in a real network, by varying the number of requests. Due to the small problem size, we are able to obtain optimal solutions using `ILP`. It can be seen that algorithm `ALG` achieves near-optimal throughput and performs the best among different algorithms. Specifically, when the number of requests is small, all algorithms perform well by admitting from 70 to 90 percent of all requests. However, with the increase on the number of requests, only the proposed algorithm `ALG` can achieve high network throughput. Despite the narrow performance gap between algorithms `ALG` and `Baseline` when the number of requests is 40, algorithm `ALG` admits 20 percent more requests when there are 160 requests. This demonstrates the superiority of the proposed algorithm. Meanwhile, it can be seen from Fig. 6a that algorithm `ALG-V` outperforms algorithm `Baseline` except when the number of requests is very small. The reason is that there is abundant network resource in the system if the number of reuqests is small and the demanded resource by the requests is small too. Therefore, the impact of VNF instance sharing and scaling is not as significant as when there is a number of requests, and `Baseline` can easily find a path in the constructed multi-stage graph that satisfies the delay requirement.



(a) The number of requests admitted



(b) The operational cost



(c) The running time

Fig. 7. Performance of different algorithms within one time slot, using synthetic networks of various sizes.

Fig. 6b shows the operational costs of different algorithms for a single time slot. The costs have been normalized with respect to the optimal cost found by the ILP for ease of discussion. When the number of requests is less than 100, we notice two trends. First, the operational costs of all algorithms except the ILP increase with more requests due to more request admissions. Second, among algorithms ALG, ALG-V, and Baseline, ALG has the lowest operational cost, because it constructs instances of the GAP that accurately capture the costs of different options to admit user requests, thereby network resources being efficiently utilized. Moreover, when the number of requests is greater than 100, only the normalized operational cost of algorithm ALG increases whereas those of algorithms ALG-V and Baseline decrease. This is due to different numbers of requests admitted by different algorithms. Although the solution delivered by algorithm ALG has a higher operational cost, the algorithm achieves a much higher network throughput than the other two comparison algorithms.

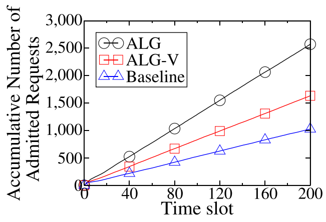
Fig. 6c illustrates the running times of different algorithms. Meanwhile, algorithm ALG-V is an order of magnitude faster than ALG, because there are only $O(|V|)$ bins that represent vertical scaling in the GAP instances by algorithm ALG-V, whereas algorithm ALG has additional $O(|V|^2)$ bins to represent horizontal scaling of VNF instances. More importantly, Fig. 6 demonstrates that the running time of ILP is prohibitively high, thus, the algorithm suffers from poor scalability and is not applicable to the problem with large scale. This necessitates an effective, scalable method that can find near-optimal solutions in a much shorter amount of time, for which algorithm ALG is an excellent candidate.

Fig. 7a plots the curves of numbers of requests admitted by different algorithms at a single time slot, by varying the network size from 100 to 600 while fixing the number of requests at 100. Notice that due to the large network sizes, ILP does not finish in a reasonable amount of time, thus the optimal results are not included in these figures. Similar to Fig. 6a, it can be observed that the proposed algorithm ALG achieves the highest throughput among all algorithms, and algorithm ALG-V outperforms algorithm Baseline in all network sizes. Fig. 7b plots the operational costs of different algorithms, from which we notice that the operational cost of an algorithm is highly correlated to the number of requests admitted by it. Take algorithm ALG for instance. It has the highest network throughput and the highest operational cost as well among the three algorithms. The rationale behind is that the more requests admitted, the larger the resource consumption needed to meet the resource demands of admitted requests. In addition, Fig. 7c shows the running times of different algorithms. Both algorithms ALG and ALG-V have large running times. This can be explained by noticing that during the construction of a GAP instance, algorithm ALG adds a bin that represents horizontal scaling of an instance only if the scaling does not violate the resource requirements. When the network size is large, the possible violation of resource requirements increases, thereby reducing the size of the GAP instance constructed by algorithm ALG. Thus, the difference in the sizes of GAP instances constructed by algorithms ALG and ALG-V is small and these algorithms run in similar amounts of time.

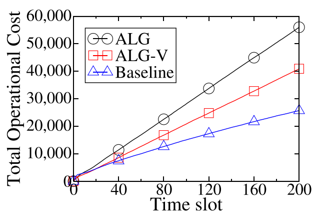
6.3 Performance Evaluation of Different Algorithms within a Finite Time Horizon

We then evaluate the performance of the proposed algorithm within a time horizon consisting of 200 time slots. The number of requests at each time slot follows a Poisson distribution with a mean of 30, and each admitted request spans 1 to 10 time slots randomly. In the beginning of each time slot, some executing requests will depart from the network and the allocated resources for them will be released back to the network, before handling newly arrived requests.

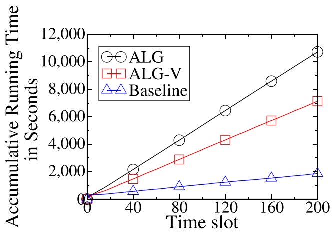
The results are summarized in Fig. 8. It can be seen from Fig. 8a that algorithm Baseline has the lowest network throughput among all three aforementioned algorithms. On



(a) The number of requests admitted for a period



(b) The total operational cost



(c) The total running time

Fig. 8. Performance of different algorithms within a finite time horizon, using a real network topology.

the contrary, algorithms `ALG` and `ALG-V` can admit more requests through effectively scaling VNF instances to meet the resource requirements of requests. The total operational cost of each algorithm in the end of each time slot is shown in Fig. 8b, from which we can see the similar patterns as shown in Fig. 7b. That is, due to the larger number of requests admitted by `ALG`, the operational cost of algorithm `ALG` is higher than the other two algorithms. The differences in operational costs of different algorithms do not mean algorithm `ALG` is inferior, because the network throughput is the main optimization objective, and algorithm `ALG` indeed has the highest network throughput. Meanwhile, Fig. 8c shows that the running time of algorithm `Baseline` is much smaller in comparison with the ones of algorithms `ALG` and `ALG-V`. It must be reiterated that this running time comes at the expense of admitting much fewer requests. Specifically, although the running time of algorithm `Baseline` is approximately one-fifth of that of algorithm `ALG`, algorithm `Baseline` only admits half the number of requests as algorithm `ALG` does in the end of the time horizon. The performance difference demonstrates the effectiveness of the proposed algorithm in maximizing the throughput of a network.

7 Conclusions

In this paper, we studied the network throughput maximization problem, by admitting as many as NFV-enabled requests while meeting their QoS requirements, through jointly considering vertical scaling by instantiating new VNF instances and horizontal scaling by migrating existing VNF instances to new locations. We first formulated the problem as an ILP. We then proposed an efficient heuristic for the problem. We finally evaluated the performance of the proposed algorithm through conducting experiments. Experimental results demonstrated that the proposed algorithm outperforms a baseline algorithm, and the solution quality of the proposed algorithm is on a par with that of the optimal solution delivered by the ILP.

Acknowledgments

The authors really appreciate the three anonymous referees and the associate editor for their expertise comments and constructive suggestions, which helped them improve the quality and presentation of the paper greatly.

References

[1] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xOMB: Extensible open middleboxes with commodity servers," in *Proc. 8th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, 2012, pp. 49–60.

[2] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Sci.*, vol. 286, pp. 509–512, 1999.

[3] C. Bu, X. Wang, M. Huang, and K. Li, "SDNFV-based dynamic network function deployment: Model and mechanism," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 93–96, Jan. 2018.

[4] F. Carpio, W. Bziuk, and A. Jukan, "Replication of virtual network functions: Optimizing link utilization and resource costs," *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 521–526.

[5] F. Carpio, A. Jukan, and R. Pries, "Balancing the migration of virtual network functions with replications in data centers," *Proc. Netw. Operations Manage. Symp.*, 2018, pp. 1–8.

[6] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, pp. 162–166, 2006.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* Cambridge, MA, USA: MIT Press, 2009.

[8] C. Demetrescu and I. Finocchi, "Combinatorial algorithms for feedback problems in directed graphs," *Inf. Process. Lett.*, vol. 86, pp. 129–136, 2006.

[9] 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.

[10] 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, https://ieeexplore.ieee.org/abstract/document/7883907.

[11] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags," in *Proc. 11th USENIX Conf. Networked Syst. Des. Implementation*, 2014, pp. 533–546.

[12] X. Fei, F. Liu, H. Xu, H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 486–494.

[13] A. Gember, A. Krishnamurthy, S. St. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *arXiv:1305.0209*, 2013.

[14] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling innovation in network function control," *Proc. ACM Conf. SIGCOMM*, 2014, pp. 163–174.

[15] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A software defined internet exchange," *Proc. ACM Conf. SIGCOMM*, 2014, pp. 551–562.

[16] M. Huang, W. Liang, Y. Ma, and S. Guo, "Throughput maximization of delay-sensitive request admission via virtualized network function placements and migrations," *Proc IEEE Int. Conf. Commun.*, 2018, pp. 1–7.

[17] M. Huang, W. Liang, Z. Xu, and S. Guo, "Efficient algorithms for throughput maximization in Software-Defined Networks with consolidated middleboxes," *IEEE Trans. Netw. Serv. Manage.*, vol. 14, no. 3, pp. 631–645, Sep. 2017.

[18] J. Xia, Z. Cai, and M. Xu, "Optimized virtual network functions migration for NFV," *Proc. Int. Conf. Parallel Distrib. Syst.*, 2016, pp. 340–346.

[19] M. Jia, W. Liang, M. Huang, Z. Xu, and Y. Ma, "Throughput maximization of NFV-enabled unicasting in software-defined networks," *Proc. IEEE Global Commun. Conf.*, Dec., 2017, pp. 1–6.

[20] M. Jia, W. Liang, M. Huang, Z. Xu, and Y. Ma, "Routing cost minimization and throughput maximization of NFV-enabled unicasting in software-defined networks," *IEEE Trans. Netw. Serv. Manage.*, vol. 15, no. 2, pp. 732–745, Jun. 2018.

[21] Y. Jia, C. Wu, Z. Li, F. Le, and A. Liu, "Online scaling of nvf service chains across geo-distributed datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 699–710, Apr. 2018.

[22] K. Kawashima, T. Otoshi, Y. Ohsita, and M. Murata, "Dynamic placement of virtual network functions based on model predictive control," *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2016, pp. 1037–1042.

[23] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *J. Select. Areas Commun.*, vol. 29, pp. 1765–1775, 2011.

[24] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv Manage.*, vol. 14, no. 3, pp. 543–553, Sep. 2017.

[25] Y. Ma, W. Liang, and J. Wu, "Online NFV-enabled multicasting in mobile edge cloud networks," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2019.

[26] Y. Ma, W. Liang, and Z. Xu, "Online revenue maximization in NFV-enabled SDNs," *Proc. IEEE Int. Conf. Commun.*, May, 2018, pp. 1–8.

[27] Y. Ma, W. Liang, Z. Xu, and S. Guo, "Profit maximization for admitting requests with network function services in distributed clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1143–1157, May 2019.

[28] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," *Proc. 11th USENIX Conf. Networked Syst. Des. Implementation*, 2014, pp. 459–473.

[29] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Trans. Netw. Serv. Manage.*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[30] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De 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, Jan.-Mar. 2016.

[31] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 27–38.

[32] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/Merge: System support for elastic execution in virtual middleboxes," *Proc. 10th USENIX Conf. Networked Syst. Des. Implementation*, 2013, pp. 227–240.

[33] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Yek, "Provably efficient algorithms for joint placement and allocation of virtual network functions," *Proc. INFOCOM*, 2017, pp. 1–9.

[34] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," *Proc. NSDI*, 2012, pp. 323–336.

[35] 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," *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2012, pp. 13–24.

[36] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau, "Online vnf scaling in datacenters," *Proc. CLOUD*, 2016, pp. 140–147.

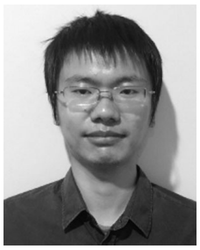
[37] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis, "Approximation and online algorithms for NFV-enabled multicasting in SDNs," *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 625–634, Jun. 2017.

[38] 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.

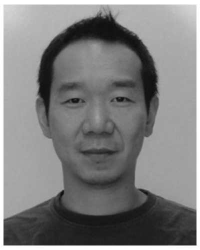
[39] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao, "Task offloading with network function services in a mobile edge-cloud network," *IEEE Trans. Mobile Comput.*, https://ieeexplore-ieee-org.virtual.anu.edu.au/document/8502709

Meitian Huang received the BSc degree with the first class honors in computer science from the Australian National University, in 2015. He currently is working toward the PhD degree in the Research School of Computer Science, Australian National University. His research interests include software-defined networking, virtualized network function services, algorithm design and analysis, and cloud computing.

Weifa Liang (M'99-SM'01) received the BSc degree from Wuhan University, China, in 1984, the ME degree from the University of Science and Technology of China, in 1989, and the PhD degree from the Australian National University, in 1998, all in computer science. He is currently a professor with the Research School of Computer Science, Australian National University. His research interests include design and analysis of energy efficient routing protocols for wireless ad hoc and sensor networks, mobile edge computing (MEC), network function virtualization, software-defined networking (SDN), design and analysis of parallel and distributed algorithms, approximation algorithms, combinatorial optimization, and graph theory. He is a senior member of the IEEE.



Yu Ma received the BSc degree with the first class honors in computer science from the Australian National University, in 2015. He is currently working toward the PhD degree in the Research School of Computer Science, Australian National University. His research interests include software defined networking, internet of things (IoT), and social networking.



Song Guo (M'02-SM'11) received the PhD degree in computer science from the University of Ottawa and was a professor with the University of Aizu. He is a full professor with the Department of Computing, The Hong Kong Polytechnic University. His research interests include big data, cloud computing and networking, and distributed systems with more than 400 papers published in major conferences and journals. His work was recognized by the 2016 Annual Best of Computing: Notable Books and Articles in Computing in *ACM Computing Reviews*. He is the recipient of the 2017 IEEE Systems Journal Annual Best Paper Award and another five Best Paper Awards from IEEE/ACM conferences. He was an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and an IEEE ComSoc distinguished lecturer. He is now on the editorial board of the *IEEE Transactions on Emerging Topics in Computing*, the *IEEE Transactions on Sustainable Computing*, the *IEEE Transactions on Green Communications and Networking*, and the *IEEE Communications*. He also served as general, TPC and symposium chair for numerous IEEE conferences. He currently serves as a director and member of the Board of Governors of ComSoc. He is a senior member of the IEEE.