Throughput Maximization of Delay-Sensitive Request Admissions via Virtualized Network Function Placements and Migrations

Meitian Huang*, Weifa Liang*, Yu Ma*, and Song Guo†
* Research School of Computer Science, The Australian National University, Canberra, ACT 2601, Australia
† Department of Computing, The Hong Kong Polytechnic University, Hong Kong

***Abstract*—Network Function Virtualization (NFV) has attracted significant attentions from both industry and academia as an important paradigm change in network service provisioning. Most existing studies on NFV dealt with admissions of user requests through deploying Virtualized Network Function (VNF) instances for individual user requests, without considering sharing VNF instances among multiple user requests to provide better network services and improve network throughput. In this paper, we study the network throughput maximization problem by adopting two different VNF instance scalings: (i) horizontal scaling by migrating existing VNF instances from their current locations to new locations; and (ii) vertical scaling by instantiating more VNF instances if needed. Specifically, we first propose a unified framework that jointly considers both vertical and horizontal scalings to maximize the network throughput, by admitting as many requests as possible while meeting their resource demands and end-to-end transmission delay requirements. We then devise an efficient heuristic algorithm for the problem. We finally conduct experiments to evaluate the performance of the proposed algorithm. Experimental results demonstrate that the proposed algorithm outperforms a baseline algorithm.**

I. Introduction

Network Function Virtualization (NFV) has been paid a significant attention from both industry and academia as an important paradigm change in network service provisioning. Under this new NFV architecture, a *service chain* that consists of 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). Each of the VNFs can be implemented as a software component, referred to as a *VNF instance*, running on off-the-shelf physical servers [14]. The VNF instances may be relocated or instantiated at different locations (servers) in a network without necessarily purchasing and installing new hardware. By decoupling network functions from the hardware platform on which network functions are executed, NFV has the great potential to lead to significant reductions in both operating expenses (OPEX) and capital expenses (CAPEX) of network service providers, and it will also facilitate the deployment of new services with increased agility and faster time-to-value [14].

Most existing studies concentrated on the optimal placement of VNF instances under a specific optimization objective based on user-specified resource demands [16], [18] or historical traffic patterns or predictions [2], [12], yet little attention has ever been paid on maximizing the network throughput via VNF instance scaling, while meeting the resource demand and end-to-end transmission delay requirement of each admitted user request. To achieve this goal, in this paper we jointly consider horizontal scaling, i.e., migrating existing instances from their current locations to new locations, and vertical scaling, i.e., instantiating new instances, to achieve the goal. Fig. 1 is an illustrative example of vertical and horizontal scalings,

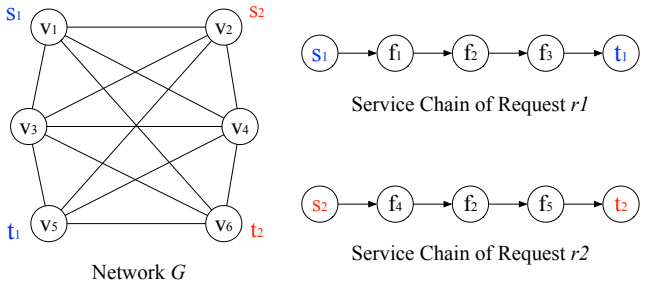


Fig. 1. An example network G with six nodes and two requests r_1 and r_2

where the network G consists of six nodes v_1, v_2, ..., v_6, and it needs to admit two requests r_1 and r_2 with each having its Quality of Service (QoS) requirement in terms of the end-to-end transmission delay. Suppose all nodes in G are attached with servers that can accommodate VNF instances. When request r_1 is admitted, a VNF instance of network function f_2 may be created in v_3, because v_3 is close to the source and destination of request r_1. As a result, when admitting another request r_2 that demands network function f_2, the network operator can either (i) reuse the existing VNF instance of f_2 in node v_3, (ii) migrate the existing VNF instance of f_2 from node v_3 to another location that is closer to the source and destination of request r_2, or (iii) instantiate a new VNF instance of f_2 in a node. Apparently, (i) is feasible only if routing the data traffic of request r_2 through v_3 can satisfy its QoS requirement and the existing VNF instance of f_2 in v_3 can handle both r_1 and r_2 simultaneously; (ii) is applicable only if the QoS 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 not applicable, because it may incur a higher cost.

In this paper, we study the problem of network throughput maximization by dynamically admitting requests while mini-

978-1-5386-3180-5/18/$31.00 ©2018 IEEE

mizing the accumulative instance scaling cost. This problem poses great challenges. First, it requires a consideration between reusing existing instances, vertical scaling, and horizontal scaling. When different user requests demand the same type of network functions, all these requests can be admitted by reusing the same VNF instance in the same server, provided that the instance is capable of handling all requests. Despite lower costs associated with this, sometimes vertical and horizontal scalings are necessary to realize the requests. In vertical scaling, each user request can be satisfied by individually instantiating a new instance of the network function. By doing so however may consume much more resources. In horizontal scaling, user requests can be satisfied by migrating existing VNF instances from old locations to new locations. Secondly, both vertical and horizontal scalings must be performed in a non-disruptive way. When performing either of the scalings in order to admit new requests, the resource requirements of all admitted, being executed requests must not be violated. Thirdly, user requests often have stringent QoS requirements that further complicate their admissions. In this paper, we will address the mentioned challenges. To the best of our knowledge, we are the first to provide a unified framework for dynamic request admissions that jointly considers vertical scaling and horizontal scaling. On the contrary, existing studies considered either vertical scaling or horizontal scaling, which may not be sufficient to handle user requests due to the characteristics of service provider networks and user requests.

The main contributions of this paper are as follows. We first formulate a novel throughput maximization problem by jointly taking into account both vertical and horizon scalings on VNF instances. We then devise a heuristic algorithm through a non-trivial reduction. We finally evaluate the performance of the proposed algorithm through simulations, based on real and synthetic network topologies. Experimental results demonstrate that the proposed algorithm is very promising, compared to a baseline algorithm. The results also demonstrate the effectiveness of joint consideration of both vertical scaling and horizontal scaling.

The rest of the paper is organized as follows. Section II will review related work. Section III will introduce the system model and notations, and define the problem. Section IV will propose an efficient heuristic algorithm. Section V will conduct performance evaluation of the proposed algorithm. Section VI will conclude the paper.

II. Related Work

Admitting user requests has been extensively studied for NFV in various settings. For instance, Jia *et al.* [11] and Xu *et al.* [18] devised the very first algorithms with performance guarantees for NFV-enabled unicasting and multicasting, respectively. However, the aforementioned studies considered the admission of each 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 design of migration mechanisms [9], [15], and the most relevant studies are [3], [4], [7], [12], [17]. In [7], the authors provided one of the earliest studies of vertical scaling of VNFs with the objective of minimizing the sum of energy consumptions of network function instances and the reconfiguration cost of network instances. Specifically, with the growth or decrease of data traffic, the computing capacities of VNF instances increase or decrease accordingly. In [17], 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.* [3], [4] 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 (QoS). They provided an integer linear program solution to the problem. While Kawashima *et al.* [12] 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.

We distinguish our work here from these mentioned ones as follows. Existing studies only dealt with either VNF migration [7], [17] or creating multiple VNF instances [3], none of them jointly considered the benefit of the two scalings, we instead propose a unified framework for this. In addition, we take the QoS requirements of user requests into account, which has not been considered by any of the mentioned studies.

III. Preliminaries

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

A. 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 switch $v \in V_S$ is characterized by its computing capacity cap_v. Each link $e \in E$ has a transmission delay D_e. We assume that switches in G are connected via high-capacity optical links so that the bandwidth capacity of each link is negligible.

B. 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. Let c_i be the computing resource demand of VNF $f_i \in \mathcal{F}$. An instance of a network function can be instantiated in a switch node $v \in V_S$ if the server attached to it has sufficient computing resource, and there may exist multiple instances of a network function at different locations. We distinguish different instances of the same network function by denoting the j-th instance of network function f_i as f_{i_j}. A VNF instance may be used to serve multiple user requests, yet the maximum traffic rate that each instance of a network function f_i can process is m_i. If existing VNF instances of network function

f_i are inadequate to handle user demands, additional instances of f_i can be instantiated with each incurring a setup cost of C_i^{setup}.

C. User requests

Each user 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 transmission 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 user 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 j-th VNF in the service chain of request r_k.

D. Request handling model

We assume that time is equally slotted. When a request r_k arrives, it will be handled 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. Due to limited network resources, the network service provider of G may only accommodate some, if not all, 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 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 occupies the system resources in the network at least for time slot t.

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 $RE_v(0) = cap_v$, and let $m_{i_j}(t)$ be the residual computing capacity of the j-th instance of network function f_i in the beginning of time slot t.

E. Vertical and horizontal scalings

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 reused 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 admitted, executing requests and the newly arrived requests.

Vertical scaling of VNF instances is applicable if existing VNF instances can be migrated to new locations such that resource requirements of both newly arrived requests in $\mathcal{R}(t)$ and executing admitted 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 f_{i_j} in the beginning of time slot t. As the location of a VNF instance f_{i_j} may change at different time slots, denote by $l_{i,j}(t)$ ($\in V_S$) the location of instance f_{i_j} in the beginning of time slot t. The VNF instance f_{i_j} may be migrated during time slot t, and thus its location in the beginning of time slot $t+1$ will become $l_{i,j}(t+1)$. The cost of migrating an existing VNF instance f_{i_j} from its current location $v_a^{i,j}$ to its destination location $v_b^{i,j}$ through a given path $P^{i,j} = (v_a^{i,j} = v_0^{i,j}, v_1^{i,j}, \ldots, v_n^{i,j} = v_b^{i,j})$ is

$$C_{i,j}^{mig}(P^{i,j}) = \sum_{r_k \in \mathcal{A}_{i,j}(t)} b_k \cdot |P^{i,j}|, \tag{1}$$

where b_k is the bandwidth demand of request r_k and $|P^{i,j}|$ is the number of links on path $P^{i,j}$.

Notice that if VNF instance f_{i_j} is migrated from $l_{i,j}(t)$ 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 f_{i_j} in location $l_{i,j}(t+1)$, instead of $l_{i,j}(t)$, and consequently their traffic need to be re-routed through different paths. To avoid any service disruption to admitted requests in $\mathcal{A}_{i,j}(t)$, the migration of every VNF instance f_{i_j} should be performed only if no delay requirement violation of admitted requests in it can occur.

However, sometimes due to the network characteristics or user requests, vertical 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}.

F. Problem definition

Given a network $G = (V, E)$, in which a set of admitted requests $\mathcal{A}(t)$ are still being executed in the beginning of time slot t, and a set of newly arrived requests $\mathcal{R}(t)$, the *network throughput maximization problem via VNF instance scalings* in G is to admit as many requests in $\mathcal{R}(t)$ as possible while minimizing the sum of the instance scaling costs through both horizontal and vertical scalings, subject to computing capacity constraints on servers in G.

G. NP-hardness

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

The NP-hardness of the problem can be shown by reducing of a well known NP-hard problem – the Generalized Assignment Problem (GAP) to it. Due to space limit, the proof detail is omitted.

IV. Heuristic Algorithm

Since the network throughput maximization problem via VNF instance scalings is NP-hard, in the following we devise an efficient heuristic for the problem.

The basic idea is that we first order network functions in $\mathcal{F}$, by taking into VNF dependency in service chains of requests in $\mathcal{R}(t) \cup \mathcal{A}(t)$, and then reduce the problem to a series of instances of GAP based on the ordering of network functions.

Each GAP instance will determine which type of the two types of scalings should be applied.

A. VNF ordering

Each user request r_k in $\mathcal{R}(t) \cup \mathcal{A}(t)$ has a service chain SC_k, and the network functions in the service chain must be applied to its data traffic in the specified order. Also, the service chains of different user requests restrict the orders in which network functions should be considered.

Take four requests in Fig. 2 for instance. 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 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.

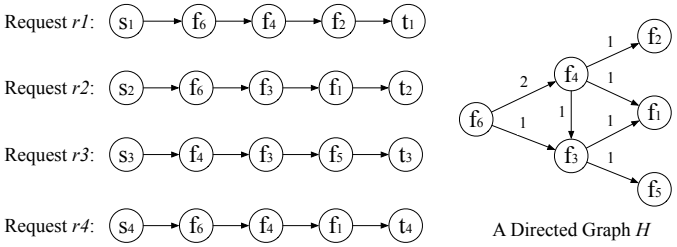


Fig. 2. The four service chains demanded by four requests r_1, r_2, r_3, and r_4 can be represented by the directed graph H

We thus identify an ordering $\tilde{\mathcal{F}}$ of network functions in $\mathcal{F}$ such that if a network function f_i appears before another network function f_j in the order specified by $\tilde{\mathcal{F}}$, then the number of requests that require f_j before f_i is minimized, preferably zero. The details of finding such an ordering are described as follows.

Given a set of arrived requests $\mathcal{R}(t)$, 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 $1 \le j \le \ell_i - 1$, if set A does not contain arc $\langle f_{i_j}, f_{i_{j+1}} \rangle$, one arc $\langle f_{i_j}, f_{i_{j+1}} \rangle$ is added to A and the weight on it is set to 1; otherwise, its weight is incremented by one. As a result, graph $H = (N, A; \omega)$ may not be a directed acyclic graph (DAG), 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 is desirable to eliminate directed cycles by removing a minimum weight set of arcs in a graph, as doing so means fewer dependencies in different service chains are violated. However, identifying the minimum weight set of arcs in H is NP-hard [6], we thus 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 its NP-hardness, we instead remove a set of arcs from H to make it a DAG, by applying an approximation algorithm due to Demetrescu *et al.* [6]. Denote by $H' = (N, A'; \omega)$ the resulting subgraph of H, which is a DAG after the removal of some of the arcs from H.

We then find a topological ordering of nodes in H'. Let $\tilde{\mathcal{F}}$ be the topological ordering of nodes in N, which is also an ordering of network functions in $\mathcal{F}$. The detailed procedure of obtaining the ordering $\tilde{\mathcal{F}}$ is given in `Procedure 1`.

Procedure 1 Finding an ordering $\tilde{\mathcal{F}}$ of network functions

Input: a service provider network $G = (V, E)$, 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: an ordering $\tilde{\mathcal{F}}$ of network functions

```
Initialize a weighted directed graph H = (N, A; ω) with N = F and A = ∅;
for each request r_i ∈ R(t) ∪ A(t) with service chain SC_i = ⟨f_{i_1}, f_{i_2}, ..., f_{i_{ℓ_i}}⟩ do
    for k ← 1 to ℓ_i − 1 do
        if A contains arc ⟨f_{i_k}, f_{i_{k+1}}⟩ then
            ω(⟨f_{i_k}, f_{i_{k+1}}⟩) ← ω(⟨f_{i_k}, f_{i_{k+1}}⟩) + 1;  /* increment the weight of an existing arc */
        else
            A ← A ∪ {⟨f_{i_k}, f_{i_{k+1}}⟩};  /* add a new arc to A and set its weight to 1 */
            ω(⟨f_{i_k}, f_{i_{k+1}}⟩) ← 1;
Obtain a DAG H' = (N, A'; ω') by applying the algorithm due to Demetrescu et al. [6] on H;
return a topological ordering F̃ of nodes N;
```

B. VNF placements and migrations

Having an ordering $\tilde{\mathcal{F}}$ of network functions in $\mathcal{F}$, we examine the network functions one by one in the increasing order. Let f_i be the network function that is currently being examined. We will determine which type of scalings should be applied to it, by constructing an instance of GAP as follows.

Each existing VNF instance f_{i_j} of f_i is represented by a bin with capacity $cap(f_{i_j}) = m_{i_j}(t)$, where $m_{i_j}(t)$ is the residual computing capacity of f_{i_j} in the beginning of time slot t. Each bin representing f_{i_j} corresponds to reusing the existing instance f_{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 $\mathcal{V}_u^i$ is added to a collection $\mathcal{B}$ of bins, representing instantiating a new VNF instance at the node u. The capacity of each of these bins is $cap(\mathcal{V}_u^i) = m_i$. For each existing VNF instance f_{i_j} at node u ($\in V_S$) and $v \in V_S \setminus \{u\}$, $\mathcal{H}_{u,v,j}^i$ is added to the set of bins $\mathcal{B}$. Note that migrating an existing VNF instance f_{i_j} may violate the end-to-end network transmission requirements

of admitted requests that are still executing in the network. Thus, bin $\mathcal{H}^i_{u,v,j}$ is added only if migrating VNF instance f_{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 f_{i_j}. The capacity of bin $\mathcal{H}^i_{u,v,j}$ is $cap(\mathcal{H}^i_{u,v,j}) = m_{i_j}(t)$, where $m_{i_j}(t)$ is the residual computing capacity of VNF instance f_{i_j} in the beginning of time slot t.

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 $b \in \mathcal{B}$. The profit $profit(r_k, b)$ of putting item r_k into bin $b \in \mathcal{B}$ is defined as follows.

If bin b is of the form f_{i_j}, which represents reusing the existing VNF instance f_{i_j}, then $profit(r_k, f_{i_j})$ is set to 1 because reusing 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. 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. (1).

The constructed instance of the GAP can be solved using an approximation algorithm due to Cohen *et al.* [5] with a $(1/(2+\epsilon))$-approximation ratio, where ϵ is a constant with $0 < \epsilon \leq 1$. Now (i) if request r_k is assigned for placement in a bin representing reusing existing instance f_{i_j}, then the request should reuse the existing VNF instance f_{i_j}; (ii) if the request is placed in a bin $\mathcal{V}^i_u$, which represents instance instantiation at u, then a new instance of network function f_i should be instantiated at u; (iii) otherwise, if it is placed in a bin $\mathcal{H}^i_{u,v,j}$, which represents horizontal scaling from u to v, then the VNF instance f_{i_j} in u should be migrated to v. If request r_i will not be assigned to any bin, then it should be rejected.

C. Algorithm

The aforementioned process handles the instance placements and migrations of one network function at a time. By repetitively applying the process for every network function in the order specified by $\tilde{\mathcal{F}}$, we can consider VNF instance scaling of all network functions. Note that resource allocations for vertical and horizontal scalings are not actually performed on G until all network functions have been examined.

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 is not met or the algorithm fails to identify a VNF instance for one or more network functions in its service chain, request r_k will be rejected, and its related scalings will not be performed.

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 reuse, 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 is not performed, because r_k should be rejected and no scaling should be applied. Note that the horizontal scaling of existing VNF instances should not violate the delay requirements of admitted requests in $\mathcal{A}(t)$. As a result, we need to calculate the accumulative delay of every request r_k in $\mathcal{A}(t)$, and if the accumulative delay of request r_k is greater than its delay requirement d_k, we know that the violation is due to instance migrations of network functions r_k demands. To resolve this potential violation of the delay requirement of request r_k, for each VNF instance that is used by r_k needs to be migrated from an old location u to a new location v, the VNF instance is not migrated to v, and a new instance is instantiated in v instead. The detailed algorithm is given in `Algorithm` 1.

Algorithm 1 A fast heuristic for admitting a set of requests $\mathcal{R}(t)$ into a service provider network G

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

$\mathcal{R}'(t) \leftarrow \mathcal{R}(t)$; /* the set of requests that have not been marked as non-admitted */
for each request r_k in $\mathcal{R}'(t)$ **do**
    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 $\emptyset$;
Construct an ordering $\tilde{\mathcal{F}}$ of network functions $\mathcal{F}$ by invoking `Procedure` 1;
for each network function f_i in the order specified by $\tilde{\mathcal{F}}$ **do**
    Let $\mathcal{R}_i(t)$ be the subset of requests in $\mathcal{R}'(t) \cup \mathcal{A}(t)$ that require network function f_i;
    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;
    Solve the GAP instance by invoking the algorithm due to Cohen *et al.* [5];
    for each request r_k in $\mathcal{R}_i(t)$ **do**
        if request r_k is not assigned to a bin in the solution obtained in Step 8 **then**
            $\mathcal{R}'(t) \leftarrow \mathcal{R}'(t) \setminus \{r_k\}$; /* Mark request r_k as a non-admitted request */
        else
            Add to $r_k.scalings$ the scaling corresponding to the bin to which request r_k is assigned in the obtained solution;
for each request $r_k \in \mathcal{R}'(t)$ **do**
    if the end-to-end requirement of r_k is not met **then**
        $\mathcal{R}'(t) \leftarrow \mathcal{R}'(t) \setminus \{r_k\}$;
return the set of scalings $r_k.scalings$ to be performed for every request r_k in $\mathcal{R}'(t)$;

D. Algorithmic analysis

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

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.,* $\ell_{\max}=\max\{$*the length* ℓ_k *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 phases: (i) finding an ordering $\tilde{\mathcal{F}}$ of network functions by constructing an auxiliary directed graph and eliminating possible directed cycles in the auxiliary graph; and (ii) constructing an instance of the GAP and finding a solution for each network function in the ordered specified by $\tilde{\mathcal{F}}$.

Phase (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.* [6] takes $O(|N|\cdot|A|)=O(L^3)$ time. The running time of Phase (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.* [5] 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. ■

V. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm through experimental simulations. We start with the experimental environments, and then evaluate the performance of the proposed algorithm.

A. Experimental environment

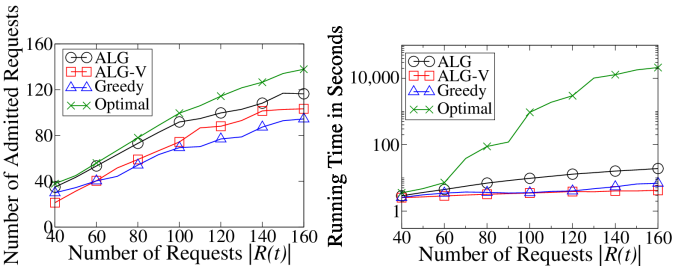
We adopt both real network topologies [13] and synthetic network topologies [1] in the simulations. The parameter settings are consistent with previous studies, including (i) the number of servers [10], (ii) the computing capacity of each server and link delays [17], and (iii) types of network function and their computing demands [7], [10]. In the case of a single time slot, some VNF instances are randomly placed in the network. Each request $r_i\in\mathcal{R}(t)$ is generated by randomly picking two nodes in V as its source s_i and destination t_i, and assigning its traffic rate b_i and its end-to-end delay requirement d_i as per [10]. The generation of service chain SC_i of request r_i is consistent with [7], [17]. The default accuracy parameter ϵ in solving the GAP is set to 0.1. The running time is obtained based on a machine with a 3.40GHz Intel i7 Quad-core CPU and 16 GB RAM.

We evaluate the performance of the proposed algorithm against a greedy baseline algorithm similar to a solution in [8]. The baseline algorithm is described as follows.

For every request $r_i\in\mathcal{R}(t)$, the greedy baseline 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 greedy baseline 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 existing VNF instances. We refer to `Algorithm 1`, `Algorithm 1` with vertical scaling and no horizontal scaling, and the greedy baseline algorithm as `ALG`, `ALG-V`, and `Greedy` respectively. Each value in figures is the average of the results of 30 trials.

B. Algorithm performance within a single time slot

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

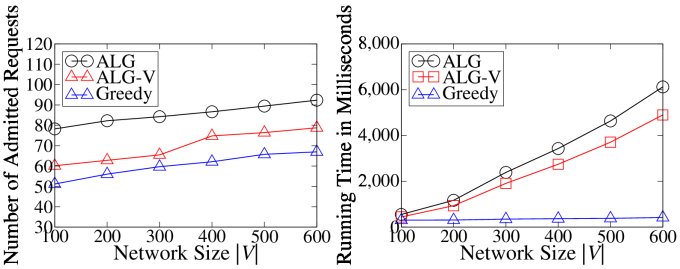


(a) The number of admitted requests (b) The running time

Fig. 3. Performance of different algorithms within one time slot using a real network topology with 40 nodes

Fig. 3 (a) shows the number of requests admitted by different algorithms in a real network, by varying the number of requests. It can be seen that algorithm `ALG` achieves the highest throughput. Specifically, when the number of requests is small, all algorithms perform well by admitting 70% to 90% of all requests. However, with the increase in the number of requests, only the proposed algorithm `ALG` can achieve high network throughput. Despite the narrow gap between `ALG` and `Greedy` when the number of requests is 40, `ALG` admits 20% more requests when there are 160 requests. This demonstrates the superiority of the proposed algorithm. Meanwhile, it can be seen from Fig. 3 (a) that `Greedy` outperforms `ALG-V` only when the number of requests is small. The reason is that when the number of requests is small, `ALG-V` can accommodate a limited number of requests by performing vertical scaling only. Fig. 3 (b) illustrates the running times of different algorithms. Meanwhile, `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 constructed by `ALG-V`, whereas `ALG` has additional $O(|V|^2)$ bins to represent horizontal scaling.

Fig. 4 (a) plots the curves of numbers of requests admitted by different algorithms at a single time slot, when the network



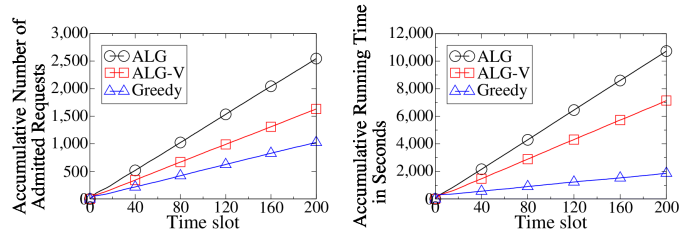
(a) The number of admitted requests (b) The running time

Fig. 4. Performance of different algorithms within one time slot using synthetic networks of various sizes

size changes from 100 to 600, with the number of requests being set to 100. Similarly, it can be observed that the proposed algorithm ALG achieves the highest throughput. However, ALG-V outperforms the baseline Greedy in all network sizes. Meanwhile, Fig. 4 (b) shows some interesting results, in particular, both ALG and ALG-V have large running times. This can be explained by noticing that during the construction of a GAP instance, 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 ALG. Thus, the difference in the sizes of GAP instances constructed by ALG and ALG-V is small and these algorithms run in similar amounts of time.

C. Algorithm performance 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 to the network, before handling newly arrived requests.



(a) Accumulative number of requests admitted (b) Accumulative running time

Fig. 5. Performance of different algorithm within a finite time horizon using a real network topology

The results are summarized in Fig. 5. It can be seen from Fig. 5 (a) that algorithm Greedy has the lowest network throughput among all three aforementioned algorithms. On the contrary, algorithms ALG and ALG-V can admit more requests through effectively scaling VNF instances to meet the resource requirements of requests. Meanwhile, Fig. 5 (b) shows that the running time of algorithm Greedy 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 Greedy is approximately one-fifth of that of ALG, Greedy only admits half as many requests as ALG does at the end of the time horizon.

VI. Conclusions

In this paper, we studied the problem of maximizing the network throughput, through jointly considering both vertical scaling by instantiating new VNF instances and horizontal scaling by migrating existing VNF instances to new locations. We first provided a formulation of the problem. Due to its NP-hardness, we then proposed an efficient heuristic for the problem. We finally conducted experiments to evaluate the performance of the proposed algorithm. Experimental results demonstrated that the proposed algorithm outperforms a baseline algorithm.

References

[1] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, Vol. 286, pp. 509–512, 1999.

[2] C. Bu, X. Wang, M. Huang, and K. Li. SDNFV-based dynamic network function deployment: Model and mechanism. *IEEE Communication Letters*, doi: 10.1109/LCOMM.2017.2654443.

[3] F. Carpio, W. Bziuk, and A. Jukan. Replication of virtual network functions: Optimizing link utilization and resource costs. *Proc. of ICC*, IEEE, 2017.

[4] F. Carpio and A. Jukan. Balancing the migration of virtual network functions with replications in data centers. arXiv:1705.05573.

[5] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, Vol. 100, pp. 162–166, Elsevier, 2006.

[6] C. Demetrescu and I. Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, Vol. 86, pp. 129–136, Elsevier, 2006.

[7] 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. *Trans. on Networking*, DOI: 10.1109/TNET.2017.2668470, IEEE, 2017.

[8] 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, IEEE, 2017.

[9] A. Gember-Jacobson *et al.* OpenNF: Enabling innovation in network function control. *Proc. of SIGCOMM*, ACM, 2014.

[10] M. Huang, W. Liang, Z. Xu, and Song Guo. Efficient Algorithms for Throughput Maximization in Software-Defined Networks With Consolidated Middleboxes *IEEE Transactions on Network and Service Management*, Vol. 14, issue 3, pp. 631–645, 2017.

[11] M. Jia, W. Liang, M. Huang, Z. Xu, and Y. Ma. Throughput maximization of NFV-enabled unicasting in Software-Defined Networks. *Proc of IEEE Globecom'17*, Dec, 2017.

[12] K. Kawashima *et al.* Dynamic placement of virtual network functions based on model predictive control. *Proc. of NOMS*, IEEE/IFIP, 2016.

[13] S. Knight *et al.* The internet topology zoo. *J. Selected Areas in Communications*, Volume 29, pp. 1765–1775, IEEE, 2011.

[14] R. Mijumbi *et al.* Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, IEEE, 2016.

[15] S. Rajagopalan *et al.* Split/Merge: System support for elastic execution in virtual middleboxes. *Proc. of NSDI*, USENIX, 2013.

[16] Y. Sang *et al.* Provably efficient algorithms for joint placement and allocation of virtual network functions *Proc. of INFOCOM*, IEEE, 2017.

[17] J. Xia, Z. Cai, and M. Xu. Optimized virtual network functions migration for NFV. *Proc. of International Conference on Parallel and Distributed Systems*, IEEE, 2016.

[18] Z. Xu *et al.* Approximation and online algorithms for NFV-enabled multicasting in SDNs. *Proc. of ICDCS*, IEEE, 2017.