Profit Maximization of NFV-Enabled Request Admissions in SDNs

Yu Ma†, Weifa Liang†, Meitian Huang†, and Song Guo¶
† The Australian National University, Canberra, ACT 2601, Australia
¶ The Hong Kong Polytechnic University, Hong Kong
Email: yu.ma@anu.edu.au, wliang@cs.anu.edu.au, meitian.huang@anu.edu.au, cssongguo@comp.polyu.edu.hk

***Abstract*—Network Function Virtualization (NFV) and Software-Defined Networking (SDN) have been envisioned an essential milestone in the evolution of communication networks. Their integration provides a more flexible and easier manageable software-based network environment that induces high expectations for reducing capital expenditures (CAPEX) and operational costs (OPEX) of network service providers. They also introduce technical challenges. One such challenge is to manage the placement of virtualized network functions (VNFs) and to steer the data traffic of each NFV-enabled request through its specified network functions. In this paper, we opportunistically adopt the flexibility and cost-efficiency of NFV and SDN for NFV-enabled request admissions in SDNs and formulate profit maximization problems for static and dynamic NFV-enabled request admissions. We first provide an integer linear programming (ILP) solution to the problem in the static version if the problem size is small; otherwise, we devise a fast approximation algorithm with a provable approximation ratio for the static request admissions. We then propose an efficient online algorithm for dynamic request admissions, by leveraging VNF instance migrations and idle VNF instance releases back to the system. We finally evaluate the performance of the proposed algorithms through experimental simulations. Simulation results demonstrate that the proposed algorithms are very promising.**

I. Introduction

Admitting user requests and routing their data traffic are the fundamental functionalities of any communication networks. To ensure performance and security of data transfer in networks, each data traffic usually needs to pass through a certain specified network functions such as Firewalls, Load Balancers, Network Address Translators, before reaching its destination. Traditional networks are installed dedicated hardware middleboxes to implement such network functions, which are not only expensive but also error-prone, since each service requires a dedicated hardware device that needs to be individually configured with its own command syntax [15].

With the fast evolution of the Internet, more and more new network functions are added into networks. Meanwhile, ever-growing demands by users require network services. The network service providers are facing great challenges in deploying new network services easily, quickly and cost-efficiently. These motivate the study of Network Function Virtualization (NFV) and Software-Defined Networking (SDN), which can address these great challenges [2], [4]. NFV leverages generic servers to implement different network functions as software components in data centers instead of using purpose-specific hardware middleboxes, which introduces a new dimension for cost savings and network function deployment flexibility [2]. Furthermore, SDN that introduces the separation of the control plane from the data plane provides more flexibility in steering network traffic flows [4]. The integration of NFV and SDN thus enables flexible implementations of network functions, dynamic routing of user requests, and optimizes network resources utilization, which has been envisioned as the key technologies for the next generation of networking [15], by significantly reducing the CAPEX and OPEX of network service providers. The adoption of SDN and NFV technologies poses several fundamental challenges for network service providers. One is to choose which data centers in the network to implement the VNFs of each NFV-enabled request. Since different data centers have different VNF instances installed and different user requests request different VNF instances. The other is to dynamically allocate resources to admit user requests without the knowledge of their future arrivals.

There have been several studies on NFV-enabled user request routing [8], [11], [13], [16]. However, most of these studies focused on exact solutions by formulating the problem as an Integer Linear Programming (ILP) or provided heuristics without any performance guarantees. For example, paper [11] aimed to develop novel architectures for NFV-enabled networks, by formulating ILP solutions to optimize network performance. Kuo *et al.* [9] studied the NFV-enabled user request routing problem by fully utilizing existing network function instances at different servers. They developed a dynamic programming algorithm for throughput maximization. Racheg *et al.* [14] considered profit-driven resource allocations in SDNs by proposing several heuristics that mainly rely on enumerating all possible embedding of VNF instances and data transmission paths into networks, thus they are not applicable to large-scale problems. Lukovszki and Schmid [12] considered dynamic request admission under the assumption that each request requires the same type of service chain and each computing node has the same capacity. None of the mentioned studies considered the utilization of existing VNF instances and VNF instance migrations and releases to reduce the implementation cost of NFV-enabled requests. Thus, it may be not cost effective especially when request patterns change frequently in reality.

In this paper, an approximation algorithm with a performance guarantee for a set of NFV-enabled requests is devised, and an efficient online algorithm for dynamic NFV-enabled request admissions is also proposed through utilizing existing VNF instances, live VNF instance migrations, and idle VNF instance releases to reduce the implementation cost of NFV-enabled requests.

978-1-5386-4727-1/18/$31.00 ©2018 IEEE

The main contributions of this paper are summarized as follows. We consider the profit maximization problem of NFV-enabled request admissions in SDNs. We first formulate an Integer Linear Programming (ILP) solution for the problem in the static version when the problem size is small; otherwise, we devise an approximation algorithm for a set of requests. We then propose an online algorithm for dynamic request admissions without the knowledge of future request arrivals by performing live VNF instance migrations and idle VNF instance releases. We finally evaluate the performance of the proposed algorithms by simulations.

The rest of the paper is organized as follows. Section II introduces notions, notations, and problem definitions. Section III formulates an ILP solution, while Section IV devises an approximation algorithm for the problem. Section V develops an efficient online heuristic for dynamic request admissions. Section VI evaluates the proposed algorithms empirically, and Section VII concludes the paper.

II. Preliminaries

In this section we introduce the network model, notations and problem definitions.

Consider a Software Defined Network (SDN) that is modelled by an undirected graph $G = (\mathcal{SW} \cup V, E)$, where $\mathcal{SW}$ is the set of SDN-enabled *switches* and V is the set of *data centers*, and E is the set of *links* between SDN-enabled switches and between SDN-enabled switches and data centers. We assume that $|V| \ll |\mathcal{SW}|$. Each data center has abundant computing resource for implementing various *virtualized network functions* (VNFs) such as Firewalls, Proxies, Network Address Translators (NATs), and so on. Denote by C_v the computing capacity of data center $v \in V$. We assume the bandwidth of each link $e \in E$ is abundant for NFV-enabled request admissions. We further assume that time is slotted into equal time slots.

Consider a *NFV-enabled request* $r_j = \langle s_j, t_j; b_j, f_j^{(k)}, d_j, R_j \rangle$ that consists of a source node $s_j \in \mathcal{SW}$, a destination node $t_j \in \mathcal{SW}$, the packet rate b_j, the duration d_j of the request implementation in the system, the payment R_j to network service providers, and a network function or a consolidated network functions $f_j^{(k)}$ on r_j [8] that its data traffic must pass through one instance of VNF $f_j^{(k)}$ for processing. Without loss of generality, we assume there are K different types of VNFs in the system, i.e., $1 \leq k \leq K$. Each VNF is implemented by a VM which is *an instance of VNF* in a data center. Implementing VNF instances in data centers consumes their computing resources. Denote by $C(f^{(k)})$ the computing resource consumption of an instance of type-k VNF. Assume that there are $n_v^{(k)} \in \mathbb{N}$ instances of type-k VNF pre-installed in data center $v \in V$. Sometimes, the number of existing instances of a specific type of VNF in a data center may not meet the demand of a new admitted request. If there is sufficient available computing resource in the data center, then new instances of that type of VNF can be created at that data center.

A cloud service provider typically charges its users by admitting their requests on a pay-as-you-go basis through adopting a common revenue collection model [14]. The gain on the revenue by admitting a request r_j depends not only on its demands on computing and bandwidth resources, i.e., the selling price of the VNF resources and the bandwidth, but also the duration of the use of the resources. The *revenue* by admitting request r_j thus is, $R_j = d_j \cdot \big(\alpha \cdot C(f_j^{(k)}) + \beta \cdot b_j\big)$, where α and β are positive constant selling prices for computing and bandwidth resources.

Provisioning VNF instances for a request at different data centers incurs different costs, due to diverse specifications of servers or diverse electricity prices at different data centers. Let $c_v(f_j^{(k)})$ be the *data processing cost* of an instance of VNF $f_j^{(k)}$ in data center v for a request r_j. Denoted by $c_v^i(f_j^{(k)})$ the *VNF instantiation cost* of creating an instance of $f_j^{(k)}$ in data center v. Furthermore, transferring data along links incurs the *data transferring cost*. Let c_e be the cost to transfer one unit of data along a link $e \in E$. The data transferring cost along the segment $P_{s_j,v}$ of a routing path from its source node s_j to a data center v is $c_e(P_{s_j,v}) = \sum_{e' \in P_{s_j,v}} b_j \cdot c_{e'}$. Similarly, the data transferring cost along the segment P_{v,t_j} of the routing path from data center v to its destination node t_j is $c_e(P_{v,t_j}) = \sum_{e' \in P_{v,t_j}} b_j \cdot c_{e'}$. Thus, the total cost $c(r_j, v)$ of admitting request r_j via the data center v is $c(r_j, v) = d_j \cdot \big(c_v(f_j^{(k)}) + c_e(P_{s_j,v}) + c_e(P_{v,t_j})\big) + c_v^i(f_j^{(k)})$. Notice that, if the data flow of a request r_j is processed by an instance of pre-installed VNF, $c_v^i(f_j^{(k)}) = 0$; otherwise, $c_v^i(f_j^{(k)}) > 0$. The *profit* by admitting a request r_j and implementing its VNF at data center $v \in V$ thus is

$$p(r_j, v) = R_j - c(r_j, v). \tag{1}$$

A. Live VNF migrations and idle VNF releases

A *live VNF instance* is the one that is currently processing the data traffic of a request, and an *idle VNF instance* is the one that has not been used and ready to admit a new request. In a VNF instance adjustment period, live VNF instances can be migrated between data centers, idle VNF instances can be released back to the system, and new VNF instances of other types can be created if needed. Denote by $c_j^{mig}(v, v')$ the migration cost of the live VNF instance of request r_j from its current data center v to its new data center v'. The migration cost $c_j^{mig}(v, v')$ is proportional with the amount of bandwidth usage for the VNF migration [6], while we assume that releasing an idle VNF instance to the system does not incur any cost.

B. Problem definitions

We consider two profit maximization problems under both a given set of requests and a sequence of requests arriving without the knowledge of future request arrivals. Specifically, the two problems are defined as follows.

Definition 1: Given an SDN $G = (\mathcal{SW} \cup V, E)$, let $\mathcal{R} = \{r_j = \langle s_j, t_j; b_j, f_j^{(k)}, d_j, R_j \rangle\}$ be a set of NFV-enabled requests, for each data center $v \in V$, the number of instances of $f^{(k)}$ is $n_v^{(k)}$, the *profit maximization problem* in G is to admit as many requests in $\mathcal{R}$ as possible such that the profit collected

is maximized, subject to computing resource constraint on each data center $v \in V$.

Definition 2: Given an SDN $G = (\mathcal{SW} \cup V, E)$ and a finite time horizon T, for each data center $v \in V$, the number of instances of $f^{(k)}$ is $n_v^{(k)}$, let $r_1, r_2, \ldots, r_j$ be the sequence of arriving requests within a given time horizon T, the *online profit maximization problem* in G is to maximize the profit during the period of T by admitting a sequence of requests without the knowledge of future request arrivals, subject to computing resource constraint on each data center in G.

The NP-hardness of the profit maximization problem can be proved by a reduction from the knapsack problem. The detailed proof is omitted due to space limitation.

III. ILP FOR THE PROFIT MAXIMIZATION PROBLEM

Given $G = (\mathcal{SW} \cup V, E)$ and a set of requests $\mathcal{R} = \{r_j = \langle s_j, t_j; b_j, f_j^{(k)}, d_j, R_j \rangle\}$, $n_v^{(k)}$ is the number of instances of VNF $f^{(k)}$ at each data center $v \in V$. For each request $r_j \in \mathcal{R}$, a_j^k is a constant, with $a_j^k = 1$ if r_j is a type k request; otherwise, $a_j^k = 0$. We can either use a pre-installed VNF instance or create a new instance of the VNF for the request. Thus, we introduce decision variables $x_{j,v} \in \{0, 1\}$ and $y_{j,v} \in \{0, 1\}$, with $x_{j,v} = 1$ if request r_j is assigned to a pre-installed VNF instance in data center $v \in V$, while $y_{j,v} = 1$ if request r_j is assigned to a newly created VNF instance in data center v. The $0-1$ ILP is described as follows.

$$maximize \quad \sum_{v \in V} \sum_{r_j \in \mathcal{R}} p(r_j, v) \cdot \big(x_{j,v} + y_{j,v}\big), \tag{2}$$

subject to

$$\sum_{1 \le k \le K} n_v^{(k)} \cdot C(f^{(k)}) + \sum_{r_j \in \mathcal{R}} \sum_{1 \le k \le K} y_{j,v} \cdot a_j^k \cdot C(f_j^{(k)}) \le C_v, \quad \forall v \in V \tag{3}$$

$$\sum_{v \in V} x_{j,v} + y_{j,v} \le 1, \quad \forall r_j \in \mathcal{R} \tag{4}$$

$$\sum_{r_j \in \mathcal{R}} x_{j,v} \cdot a_j^k \le n_v^{(k)}, \quad \forall v \in V,\ 1 \le k \le K \tag{5}$$

$$x_{j,v} \in \{0, 1\}, \quad \forall v \in V,\ r_j \in \mathcal{R} \tag{6}$$

$$y_{j,v} \in \{0, 1\}, \quad \forall v \in V,\ r_j \in \mathcal{R} \tag{7}$$

where $n_v^{(k)}$ is a given integer which is the number of pre-installed instances of VNF $f^{(k)}$ in data center v with $n_v^{(k)} \ge 0$.

The objective (2) asks for admitting a set of requests with the maximum profit gain. Constraint (3) states that the total computing resource consumption of all VNF instances is bounded by the computing capacity of each data center. Constraint (4) enforces that each admitted request r_j is assigned to either pre-installed VNF instance or newly created VNF instance of its type among all data centers, and rejected requests will not be scheduled to any instance of VNFs. Constraint (5) guarantees that requests assigned to pre-installed VNF instances do not violate their quotas.

Although this ILP can deliver an exact solution, it is only applicable when the problem size is small. The ILP solution unfortunately is not scalable.

IV. AN APPROXIMATION ALGORITHM FOR THE PROFIT MAXIMIZATION PROBLEM

Since the profit maximization problem is NP-hard, in this section we devise an approximation algorithm for it. We also analyze the time complexity and approximation ratio of the proposed approximation algorithm.

A. Algorithm overview

Given $G = (\mathcal{SW} \cup V, E)$ and a set $\mathcal{R}$ of requests, we aim to maximize the profit by admitting as many requests in $\mathcal{R}$ as possible. Recall that we can use the pre-installed instances of its VNF or create new instances as long as there is sufficient computing resource in that data center.

The basic idea of the proposed algorithm is to reduce the profit maximization problem to the Generalized Assignment Problem (GAP). To this end, each data center $v \in V$ is treated as $K + 1$ bins, corresponding the K bins for the instances of K types of pre-installed VNFs and a bin consisting of the available computing resource for creating new VNF instances for all types of VNFs in the data center, where each bin for pre-installed type k VNF instances is referred to as *the type-k bin* while the bin consisting of available computing resource is referred to as *the common bin*. Denote by $\mathcal{B}$ the set of bins in G, i.e., $|\mathcal{B}| = |V| \cdot (K + 1)$, denote by $B_v^{(k)}$ the type-k bin and B_v^c the common bin in data center v, respectively. The capacity of a bin $B \in \mathcal{B}$ is $C(B) = n_v^{(k)} \cdot C(f^{(k)})$ if it is a type-k bin; otherwise, it is a common bin with size $C(B) = C_v - \sum_{1 \le k \le K} n_v^{(k)} \cdot C(f^{(k)})$, where $n_v^{(k)}$ is the number of instances of VNF $f^{(k)}$ in data center v. For the sake of convenience, in the rest of discussions, we assume that the bins are indexed, and each bin corresponds to a unique index in the range from 1 to $|\mathcal{B}|$.

For each request r_j of type k, its computing resource consumption in a data center is defined as $s(B, j) = C(f_j^{(k)}) = C(f^{(k)})$. Its profit is $p(B, j) = p(r_j, v)$ if it is assigned to a bin B which is either type-k bin $B_v^{(k)}$ or the common bin B_v^c in data center v; or its profit $p(B, j) = 0$ otherwise.

B. Approximation algorithm

The approximation algorithm proceeds iteratively. Within each iteration, one bin is packed. It essentially decomposes the profit function into two functions: one is used for the bin packing in the current iteration; and the other is used for the rest of bin packing. For each request r_j and each bin $B \in \mathcal{B}$, the initial profit matrix is set to $p_1(B, j) = p(B, j)$.

There are $|\mathcal{B}|$ iterations. In iteration l with $1 \le l \le |\mathcal{B}|$, it assigns requests in $\mathcal{R}$ into bin l and with a profit $p_l(B, j)$, subject to the bin's capacity constraint $C(l)$. Let $\overline{S_l}$ be the set of requests assigned to the bin l by an α-approximation algorithm for the knapsack problem, clearly $\overline{S_l} \subseteq \mathcal{R}$. Then, the profit function $p_l(B, j)$ is decomposed into two profit functions $p_l^{(1)}(B, j)$ and $p_l^{(2)}(B, j)$, where $p_l^{(1)}(B, j) = p_l(B, j)$ if $r_j \in \overline{S_l}$ or $B = l$; $p_l^{(1)}(B, j) = 0$ otherwise, and $p_l^{(2)}(B, j) = p_l(B, j) - p_l^{(1)}(B, j)$. The new profit function $p_{l+1}(B, j)$ for the next bin $(l+1)$ is $p_l^{(2)}(B, j)$,

i.e., $p_{l+1}(B,j) = p_l^{(2)}(B,j)$. Then remove the column corresponding to bin l from $p_{l+1}(B,j)$ as the entries of this column are all zeroes. This procedure continues until the last bin $|\mathcal{B}|$ is packed. An approximate solution to the profit maximization problem then is derived.

Specifically, let S_l be the set of requests assigned to bin l. If $l = |\mathcal{B}|$, i.e., l is the last bin, then $S_l = \overline{S_l}$; otherwise, the set of requests assigned to bin l is $S_l = \overline{S_l} \setminus \bigcup_{B=l+1}^{|\mathcal{B}|} S_B$ for all l with $1 \leq l < |\mathcal{B}|$. The detailed algorithm is presented in `Algorithm` 1.

Algorithm 1 Approximation algorithm

Input: $G = (\mathcal{SW} \cup V, E)$ with a set V of data centers, a set of requests $\mathcal{R}$.

Output: Admit or reject each request $r_j \in \mathcal{R}$.

1: Construct $|\mathcal{B}| = |V| \cdot (K+1)$ bins, and assign the capacity $C(B)$ to each bin $B \in \mathcal{B}$;
2: Calculate the size matrix $s(B,j)$ for all request $r_j \in \mathcal{R}$ and bin $B \in \mathcal{B}$;
3: Find a shortest path for each request $r_j \in \mathcal{R}$ via each data center $v \in V$, and obtain the profit matrix $p(B,j)$;
4: Initialize the profit matrix: $p_1(B,j) \leftarrow p(B,j)$;
5: **for** $l \leftarrow 1$ to $|\mathcal{B}|$ **do**
6: Assign requests in $\mathcal{R}$ to bin l by applying an α-approximation algorithm for the knapsack problem, subject to the capacity constraint $C(l)$ of l, using the profit function $p_l(B,j)$. Let $\overline{S_l}$ be the result delivered by the approximation algorithm to bin l, where $\overline{S_l} \subseteq \mathcal{R}$;
7: Decompose the profit function into two profit functions $p_l^{(1)}(B,j)$ and $p_l^{(2)}(B,j)$; $p_{l+1}(B,j) \leftarrow p_l^{(2)}(B,j)$, and remove the column of bin l from $p_{l+1}(B,j)$;
8: **end for**
9: $S_{|\mathcal{B}|} = \overline{S_{|\mathcal{B}|}}$;
10: **for** $l \leftarrow |\mathcal{B}| - 1$ to 1 **do**
11: $S_l \leftarrow \overline{S_l} \setminus \bigcup_{B=l+1}^{|\mathcal{B}|} S_B$;
12: **end for**
13: **return** S_l for each bin $l \in \mathcal{B}$.

C. Algorithm analysis

We now analyze the performance of `Algorithm` 1.

Theorem 1: Given an SDN $G = (\mathcal{SW} \cup V, E)$, a set V of data centers that contain a certain number of pre-installed VNF instances for all K types of VNFs, and a set of requests $\mathcal{R}$, there is an approximation algorithm, `Algorithm` 1, for the profit maximization problem with approximation ratio $(2+\epsilon)$. The time complexity of the proposed algorithm is $O(|V|(K+1)(|\mathcal{R}|\log\frac{1}{\epsilon}+\frac{1}{\epsilon^4})+|\mathcal{SW}\cup V|^2\cdot|\mathcal{R}|\cdot|V|)$, where ϵ is a constant with $0 < \epsilon \leq 1$.

Proof: Following Cohen *et al.*[3], the approximation ratio of the proposed algorithm for the profit maximization problem is $2+\epsilon$, where ϵ is a constant with $0 < \epsilon \leq 1$.

The time complexity of `Algorithm` 1 is analyzed as follows. Finding a shortest path for each request r_j via each data center v takes $O(|\mathcal{SW} \cup V|^2)$ time. It takes $O(|\mathcal{R}|\log\frac{1}{\epsilon}+\frac{1}{\epsilon^4})$ time [10] to find the set of requests $\overline{S_l} \subseteq \mathcal{R}$ assigned to bin l in each iteration. Updating profit matrices takes time $O(|V|(K+1)\cdot|\mathcal{R}|)$. Thus, the running time of `Algorithm` 1 is $O(|V|(K+1)(|\mathcal{R}|\log\frac{1}{\epsilon}+\frac{1}{\epsilon^4})+|\mathcal{SW}\cup V|^2\cdot|\mathcal{R}|\cdot|V|)$. ■

V. An online algorithm for the online profit maximization problem

In this section we deal with the dynamic request admissions. We reduce this online profit maximization problem for a time duration of T as a series of the profit maximization problems at each time slot $\tau \in T$. Denote by $\mathcal{R}_\tau$ the set of arrived requests in the beginning of time slot τ. We aim to admit as many requests in $\mathcal{R}_\tau$ as possible so that the profit collected from the admitted requests is maximized.

The basic idea of the proposed online algorithm is to treat all idle VNF instances as pre-installed VNF instances. We can either make use of them or create new VNF instances if there is sufficient available computing resource in that data center. We admit requests in $\mathcal{R}_\tau$ by invoking `Algorithm` 1 at each time slot τ. Denote by $A_\tau(\subseteq \mathcal{R}_\tau)$ the set of requests admitted at time slot τ. The union $\bigcup_{\tau \in T} A_\tau$ forms a feasible solution to the online profit maximization problem.

With more and more instances created at each data center, the available computing resource in that data center will become less and less if none of the created instances will be released back to the system. On the other hand, existing VNF instances of some types may have been idle for a while. Thus, new requests of specific types may not be admitted due to lack of available computing resource for them. This exhibits an extreme mismatching between VNF instance provisioning and request demands. To address this, we will opportunistically perform VNF instance adjustments by migrating some live VNF instances to the other non-expensive data centers and release some idle VNF instances back to the system. As the workload of each data center varies over time, there are opportunities to migrate some live VNF instances from one data center to another to reduce the implementation cost of requests.

Although live VNF migrations and idle VNF releases are the promising techniques in increasing the profit of network service providers, they also pose challenges. That is, how to determine at which time slots to perform such adjustments, since frequent adjustments will incur an overhead and jeopardize efficiency of SDN operations. In the following we propose an efficient online algorithm, `Algorithm` 2, for dynamic request admissions, which consists of two stages: dynamic request admissions at each time slot, followed by live VNF migrations and idle VNF releases after a certain of time slots via threshold triggering, in which `Procedure` 3 and `Procedure` 4 will be introduced later.

We assume that VNF instance adjustment is conducted in the beginning of some time slots and its duration only takes a small fraction of the entire time slot. We also assume that there is a given threshold to trigger the VNF instance migrations and idle VNF instance releases. The intuition behind is that when a VNF instance has been idle for a while, it should be given a high priority to be released. Define *the accumulated idle time slot* of each different type of VNF instance as the sum of idle time slots since performing the VNF instance adjustment last time. If the sum $\sigma_\tau^{(k)}$ of accumulated idle time slots of VNF instances of any type k with $1 \leq k \leq K$ is greater than a predefined threshold $\sigma(>0)$, the VNF instance adjustment will be conducted at that time slot.

Algorithm 2 Online request admissions with VNF instance adjustments

Input: An SDN $G = (\mathcal{SW} \cup V, E)$, a given period T, a set of requests $\mathcal{R}_\tau$ for each time slot $\tau \in T$.

Output: A solution to maximize the profit, by dynamic request admissions with VNF instance adjustments.

1: **for** $\tau \in T$ **do**
2: Calculate $\sigma_\tau^{(k)}$ for each type k VNF instances in data centers with $1 \leq k \leq K$;
3: **if** $\exists\ \sigma_\tau^{(k)} > \sigma$ **then**
4: Perform live VNF migrations by invoking `Procedure 3`;
5: Perform idle VNF releases by invoking `Procedure 4`;
6: Update $\sigma_\tau^{(k)},\ \forall\ 1 \leq k \leq K$;
7: **end if**
8: Admit requests in $\mathcal{R}_\tau$ by invoking `Algorithm 1`;
9: **end for**

A. Live VNF migration

Given a set $\mathcal{F}$ of live VNF instances running in data centers in G, we aim to maximize the profit by migrating some live VNF instances from their current data centers to others. To ensure the feasibility of VNF instance migrations, we assume a live VNF instance can use the available computing resource or replace an idle VNF instance of the same type in its target data center. The occupied computing resource by the VNF instances in their original data centers will be released back to the system after their migration. The objective is to minimize the implementation cost of live VNF instances thus to maximize the profit of the network service provider.

The live VNF migration problem is NP-hard. Its detailed proof is omitted due to space limitation. In the following we propose an approximation algorithm through reducing it to the GAP problem as follows.

Different from `Algorithm 1`, each type-k bin consists of idle VNF instances, with size $C(B) = n_v^{(k),idle} \cdot C(f^{(k)})$, and each common bin consists of the available resource for creating new VNF instances with capacity $C(B) = C_v - \sum_{1 \leq k \leq K}(n_v^{(k),live} + n_v^{(k),idle}) \cdot C(f^{(k)})$, where $n_v^{(k),live}$ and $n_v^{(k),idle}$ are numbers of live and idle VNF instances of type k in data center v, respectively. Each live VNF instance $F_{j,v}^{(k)} \in \mathcal{F}$ is processing the data flow of a request r_j of type k at data center v. Its size $s(B,j)$ in a bin B then is $s(B,j) = C(f_j^{(k)}) = C(f^{(k)})$.

When a live VNF instance $F_{j,v}^{(k)} \in \mathcal{F}$ is assigned to a bin B, it is associated with a profit $p(B,j)$. The profit is modelled by its profit $p(r_j, v')$ in the target data center v' of r_j subtracting the VNF instance migration cost $c_j^{mig}(v,v')$ from its current data center v to the target data center v', if it is assigned to the type k bin $B_{v'}^{(k)}$ or the common bin $B_{v'}^{c}$ in any data center $v' \in V$, i.e., $p(B,j) = p(r_j,v') - c_j^{mig}(v,v')$. Otherwise, the profit is 0. Notice that, if it is assigned to its current data center, i.e., $v' = v$, its migration cost is 0. The detailed migration procedure is given in `Procedure 3`.

B. Idle VNF release

We now determine how many idle VNF instances in data centers should be released. A naive solution is to release all idle VNF instances. However, by doing so is not cost-effective since new VNF instances of this type may need to be created to accommodate new requests in the rest of time slots. Thus, a certain number if not all idle VNF instances should be kept. We apply the linear regression method based on historical request traces to predict the number of idle VNF instances to be kept.

Procedure 3 Live VNF migration in G

Input: An SDN $G = (\mathcal{SW} \cup V, E)$ with a set $\mathcal{F}$ of live VNF instances in data centers.

Output: A solution to maximize the profit, by migrating some live VNF instances from their current data centers to other data centers.

1: Construct $|\mathcal{B}|$ $(= |V| \cdot (K+1))$ bins, and obtain the capacity $C(B)$ of each bin $B \in \mathcal{B}$;
2: Calculate the size matrix $s(B,j)$ and the profit matrix $p(B,j)$ for all live VNF instances $F_{j,v}^{(k)} \in \mathcal{F}$ and bin $B \in \mathcal{B}$;
3: Find an approximate solution by applying the algorithm due to Cohen *et al.* [3];
4: Migrate each live VNF instance from its current data center to its target data center by the solution.

Let W be the width of a time window that we use for prediction. The number $n_{v,\tau-t}^{(k),live}$ of live VNF instances of type k in data center v at time slot $\tau - t$ is tracked, $1 \leq t \leq W$. Then, we apply the linear regression method [5] to predict the number $\hat{n}_{v,\tau}^{(k),live}$ of live VNF instances of type k at time slot τ in data center v. On the other hand, there are $n_{v,\tau}^{(k),live}$ numbers of live VNF instances of type k that currently are processing data traffic of requests in data center v. Thus, at least $n_{v,\tau}^{(k),live}$ of live VNF instances of type k should be kept in data center v. Thus, the actual number of VNF instances of type k should be kept in data center v is $\max\{\hat{n}_{v,\tau}^{(k),live}, n_{v,\tau}^{(k),live}\}$.

Due to the computing capacity constraint on each data center, it may not be feasible to provide $\max\{\hat{n}_{v,\tau}^{(k),live}, n_{v,\tau}^{(k),live}\}$ VNF instances of type k in data center v for all types of VNFs. We instead introduce *a fair resource allocation strategy* such that the number of VNF instances of each type of VNF should be proportional to its computing resource demand in each data center. The details of idle VNF releases are described in `Procedure 4`.

Procedure 4 Idle VNF instance release in G

Input: An SDN $G = (\mathcal{SW} \cup V, E)$, a prediction time window W, the number of live VNF instances of each type in each data center v within the prediction window W from time slot $\tau - W$ to $\tau - 1$.

Output: A solution to release idle VNF instances in data centers.

1: **for** $v \in V$ **do**
2: Predict the number $\hat{n}_{v,\tau}^{(k),live}$ of live VNF instances of each type k in v in the following time slot by applying the linear regression method [5];
3: Keep the number $\max\{\hat{n}_{v,\tau}^{(k),live}, n_{v,\tau}^{(k),live}\}$ of live VNF instances of type k in data center v;
4: Check whether there is sufficient computing resource to meet the resource demands of different types of VNF instances at each data center. If not, the number of VNF instances for each type VNF is proportionally scaling down by a factor of the actual residual computing resource at that data center to the total computing resource demand of all VNF instances;
5: **end for**

We finally analyze the performance of online algorithm

Algorithm 2 as follows.

Theorem 2: Given an SDN network $G = (\mathcal{SW} \cup V, E)$ and a time period T, a set of requests $\mathcal{R}_\tau$ arriving at time slot $\tau \in T$, there is an online algorithm Algorithm 2 for the online profit maximization problem, which takes $O(|V|(K+1)(|\mathcal{R}_\tau| \log \frac{1}{\epsilon} + \frac{1}{\epsilon^4}) + |\mathcal{SW} \cup V|^2 \cdot |\mathcal{R}_\tau| \cdot |V|)$ time for dynamic request admissions and VNF instance adjustments at each time slot τ, where ϵ is a constant with $0 < \epsilon \leq 1$.

The proof of Theorem 2 is omitted, due to space limitation.

VI. Performance Evaluation

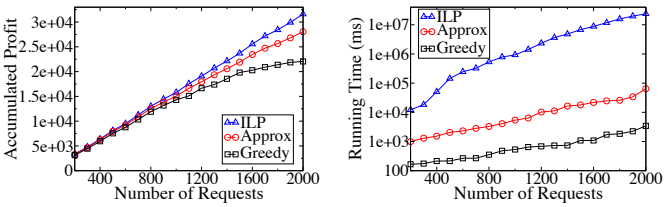
In this section, we evaluate the performance of the proposed algorithms through experimental simulations. We also investigate the impacts of important parameters on the performance of the proposed algorithms.

A. Experiment settings

We consider SDNs consisting of from 10 to 250 nodes. We adopt commonly used GT-ITM to generate network topologies. The number of data centers in each network is set to 10% of the network size. The computing capacity of each data center in the generated networks varies from 3,000 GHz to 5,000 GHz [16]. The revenue accumulation factors α and β are set as two constants drawn within $[1.5, 2.2]$ and $[0.5, 1.2]$, respectively, following typical charges in Amazon EC2 [1], [14]. The number K of different types of VNF is set as 6. The computing demand of each type of VNF instance is set between 3 GHz to 6 GHz [7], [13]. The processing cost of each network function in a data center is randomly drawn within $[0.5, 5]$, and the data transmission cost of transmitting a unit of data in each link is set within $[0.08, 0.24]$ [13]. The number of each type of VNF instances in a data center is randomly drawn from 10 to 20. The threshold σ for performing VNF adjustment is set as 300. The length of prediction window W is set as 5 time slots. Each NFV-enabled request r_j is generated as follows. Two nodes from $\mathcal{SW}$ are randomly generated as its source s_j and destination t_j. Its bandwidth requirement b_j is randomly drawn from 10 $Mbps$ to 20 $Mbps$ [7], its maximum duration is set as 8 time slots, and its type of VNF is randomly assigned one from six types. The running time is obtained based on a machine with 3.4 GHz Intel i7 Quad-core CPU and 16GB RAM. Unless otherwise specified, these parameters will be adopted in the default setting.

We first evaluate the performance of the proposed approximation algorithm Algorithm 1 against the ILP solution and a heuristic Greedy, where algorithm Greedy selects a request with the highest profit in a data center. It then updates the status of resource availability in that data center. This process continues until all requests are assigned or no requests can be admitted by any data center. In addition, we compare the performance of the online algorithm, Algorithm 2 against three baseline heuristics Non-Mig, Non-Rel and Non-Mig/Rel, where algorithm Non-Mig does not consider VNF instance migrations, algorithm Non-Rel does not consider VNF instance releases, while algorithm Non-Mig/Rel considers neither VNF instance migrations nor idle VNF instance releases. For simplicity, we refer to approximation algorithm Algorithm 1 and online algorithm Algorithm 2 as algorithms Approx and Online, respectively.

B. Performance evaluation

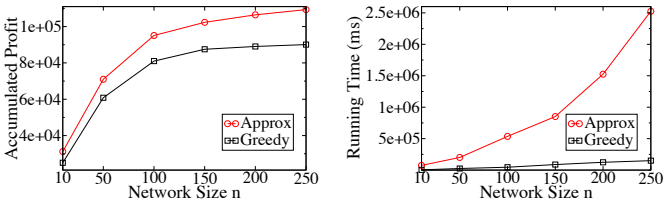


(a) The accumulated profit delivered by different algorithms in networks

(b) Running time of different algorithms in networks

Fig. 1. Performance of algorithms Approx, Greedy, and ILP in an SDN consisting of 50 nodes

We first evaluate the performance of algorithm Approx against that of algorithms ILP and Greedy, by varying the number of requests from 200 to 2,000, and fixing the number of nodes in G to 50. Fig. 1 depicts the total profits and running times of different algorithms. From Fig. 1 (a) we can see that algorithm Approx achieves a near optimal profit with at least 88.7% of the optimal one by algorithm ILP. Also, the accumulated profit of Approx is superior to Greedy in all cases and their performance gap enlarges with the increase of the number of requests. In addition, from Fig. 1 (b), it can be seen that algorithm ILP takes a much longer time to deliver an optimal solution, while algorithm Approx delivers a near optimal solution in a much shorter time. It must also be mentioned that the running time of algorithm ILP becomes prohibitive high with the increase of the number of requests, and the solution is no longer achievable when the number of requests reaches 2,000. Although algorithm Greedy has the least running time, its solution results in the least profit.



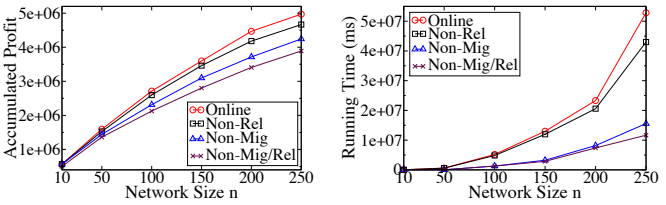
(a) The amount of profit accumulated by Approx, and Greedy

(b) Running time of Approx, and Greedy in milliseconds

Fig. 2. Performance of Approx, and Greedy on an SDN when the number of nodes varies from 10 to 250 while the number of requests is fixed at 10,000

We then investigate the performance of algorithm Approx against heuristic Greedy for large-scale networks by varying the number of nodes from 10 to 250, while fixing the number of requests at 10,000. The results are depicted in Fig. 2. It can be seen from Fig. 2 (a) that algorithm Approx delivers a larger accumulated profit than that of algorithm Greedy, and the larger the network size, the larger the performance gap between them. Furthermore, it can be seen from Fig. 2 (b) that algorithm Greedy runs much faster than algorithm Approx as algorithm Approx seeks to pack a bin by taking into account all requests within each iteration.

We then study the performance of algorithm Online against three baseline heuristics Non-Mig, Non-Rel and

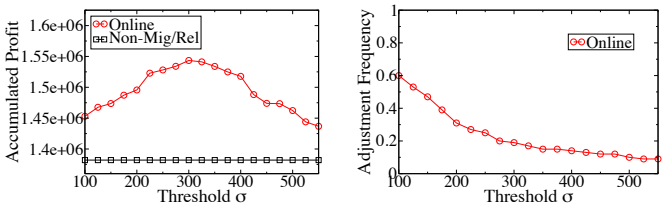


(a) The accumulated profit delivered by `Online`, `Non-Mig`, `Non-Rel`, and `Non-Mig/Rel` in networks

(b) Running time of algorithms `Online`, `Non-Mig`, `Non-Rel` and `Non-Mig/Rel` in networks

Fig. 3. Performance of different online algorithms in an SDN network where the number of nodes varies from 10 to 250 for a time horizon consisting of 100 time slots

`Non-Mig/Rel`, respectively, by varying the network size from 10 to 250 for a time horizon consisting of 100 time slots with 1,000 to 2,000 requests arriving in each time slot. Fig. 3 shows the performance curves of different algorithms. From Fig. 3 (a), we can see that algorithm `Online` outperforms all other heuristics, and the performance gap between it and the rest becomes larger and larger with the increase of the network size. In particular, when the network size reaches 250, algorithm `Online` delivers 6.53% more profit than that of algorithm `Non-Mig`, 17.13% more profit than that of algorithm `Non-Rel`, and 27.80% more profit than that of algorithm `Non-Mig/Rel`. The rationale behinds this is that algorithm `Online` opportunistically takes the advantage of VNF instance migrations to reduce processing and data transferring costs. The idle VNF instance releases aim at releasing VNF instances that have been idled for a while in order to make room for creating new VNF instances to admit more requests. Fig. 3 (b) depicts the running time curves of algorithm `Online` and the other three heuristics, where algorithm `Non-Mig/Rel` takes less time than algorithm `Online` in all cases as neither live VNF instance migrations nor idle VNF releases is performed.



(a) The accumulated profit by different online algorithms

(b) The VNF adjustment frequency by algorithm `Online`

Fig. 4. The accumulated profit and VNF adjustment frequency delivered by algorithm `Online` by varying threshold σ between 100 and 550 with network size 50.

We finally investigate the impact of threshold σ in VNF instance adjustment on the performance of algorithm `Online`, by varying its value from 100 to 550 while keeping the network size at 50. Fig. 4 plots the performance curves of algorithm `Online` and algorithm `Non-Mig/Rel`, by varying the value of σ. It can be seen from Fig. 4 (a) that algorithm `Online` delivers the maximum profit when $\sigma = 300$. When σ increases or decreases from 300, the profit decreases in both cases. This is due to the fact that the larger the value of σ, the more idle VNF instances in the network, which leads to less efficient resource usage. However, if the threshold is too small, VNF instance adjustments are performed frequently, a lot of VNF instances that can be used to admit new requests in later time slots are released and much cost will be incurred for VNF instance instantiation. However, in all cases, algorithm `Online` delivers more profit than the benchmark algorithm `Non-Mig/Rel` in which neither live VNF migrations nor idle VNF releases has ever been considered. Fig. 4 (b) plots the frequencies of VNF instance adjustments, where the frequency is defined as the ratio of the number of times VNF instance adjustment is performed to the total 100 time slots considered in this time horizon. It can be seen from Fig. 4 (b) that algorithm `Online` performs VNF instance adjustment less frequently with the increase of σ, and vice versa. This is due to the fact that when σ is small, VNF instance adjustment is performed more frequently, which causes substantial overheads.

VII. Conclusion

In this paper, we studied the profit maximization problem for NFV-enabled request admissions in a software-defined network under both a snapshot and a finite time horizon scenarios, assuming that some instances of VNFs have been installed in data centers in the SDN. We first formulated an ILP solution and devised an approximation algorithm with a provable approximation ratio for the snapshot case. We then proposed an online algorithm for dynamic request admissions. To reduce the implementation cost of dynamic request admissions, we opportunistically performed live VNF instance migrations and idle VNF instance releases. Experimental results demonstrate that the proposed algorithms are very promising.

References

[1] Amazon Web Services, Inc. Amazon ec2 instance configuration. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf.

[2] N. Chowdhury, and R. Boutaba. Network virtualization: state of the art and research challenges. *IEEE Comm. Magazine*, pp. 20 – 26, 2009.

[3] R. Cohen, *et al.* An efficient approximation for the generalized assignment problem. *Information Proc. Lett.*, vol. 100, pp. 162 – 166, 2006.

[4] A. Fischer, *et al.* Virtual network embedding: a survey. *IEEE Communications Surveys and Tutorials*, vol. 15, pp. 1888 – 1906, 2013.

[5] D. A. Freedman. Statistical Models: Theory and Practice. Cambridge University Press, pp. 26, 2009.

[6] I. Farris, *et al.* Optimizing service replication for mobile delay-sensitive applications in 5G edge network. *Proc. of ICC*, IEEE, May, 2017.

[7] M. Ghaznavi, N. Shahriar, R. Ahmed, and R. Boutaba. Service function chaining simplified. *CoRR*, vol. abs/1601.00751, Feb., 2016.

[8] M. Huang, *et al.* Throughput maximization in software-defined networks with consolidated middleboxes. *Proc. of LCN*, IEEE, 2016.

[9] T-W. Kuo, *et al.* Deploying chains of virtual network functions: on the relation between link and server usage. *Proc. of INFOCOM*, IEEE, 2016.

[10] E. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, vol. 4, pp. 339 – 356, 1979.

[11] Y. Li, L. T. X. Phan, and B. T. Loo. Network functions virtualization with soft real-time guarantees. *Proc. of INFOCOM*, IEEE, 2016.

[12] T. Lukovszki, and S. Schmid. Online admission control and embedding of service chains, *Proc. of ICSICC*, Springer, pp. 14 – 18, 2014.

[13] J. Martins, *et al.* ClickOS and the art of network function virtualization. *Proc. of NSDI'14, USENIX*, 2014.

[14] W. Racheg, N. Ghrada, and M. F. Zhani. Profit-driven resource provisioning in NFV-based environments. *Proc. of ICC*, IEEE, 2017.

[15] S. V. Rossem, *et al.* Deploying elastic routing capability in an SDN/NFV-enabled environment. *Proc. of NFV-SDN*, IEEE, 2015.

[16] Z. Xu, *et al.* Throughput maximization and resource optimization in NFV-enabled networks. *Proc. of ICC*, IEEE, 2017.