Throughput Maximization and Resource Optimization in NFV-Enabled Networks

Zichuan Xu[†], Weifa Liang[‡], Alex Galis[†], and Yu Ma[‡]
[†] Department of Electronic and Electrical Engineering, University College London, London, UK
[‡] Research School of Computer Science, Australian National University, Canberra, ACT 2601, Australia
z.xu@ucl.ac.uk, wliang@cs.anu.edu.au, a.galis@ucl.ac.uk, u5108648@anu.edu.au

***Abstract*—Network function virtualization (NFV) has been emerging as a new paradigm to enable elastic and inexpensive network services in modern computer networks, through deploying flexible virtualized network functions (VNFs) running in virtual computing platforms. Different VNFs can be chained together to form different *service chains*, to meet various user data routing demands for different network services. In this paper we consider provisioning network services in an NFV-enabled network that consists of data centers for implementing VNF instances of service chains and switches. We study the throughput maximization problem with the aim to admit as many user requests as possible while minimizing the implementation cost of the requests, assuming that limited numbers of instances of each service chain have been stored in data centers. We first propose an optimal algorithm for the problem if all requests have identical packet rates; otherwise, we devise two approximation algorithms with probable approximation ratios, depending on whether the packet traffic of each request is splittable. We finally conduct experiments to evaluate the performance of the proposed algorithms by simulations. Experimental results show that the proposed algorithms achieve at least 15% more throughput than that of a greedy algorithm.**

I. Introduction

Network services provided by the telecommunications industry traditionally make use of dedicated devices and equipment to implement various network functions, such as network address translation (NAT), firewall, and intrusion detection, to name a few. To meet ever-growing traffic demands on network services, network service providers must continuously purchase, add and operate new physical equipment into their operational networks. This does require not only high and rapidly changing skills for technicians operating and managing the equipment, but also dense deployments of network equipment, leading to high CAPEX and OPEX. For example, it is expected that the total CAPEX of worldwide network service providers reaches US$374 billion in 2019 [10] and this cost still grows at a rate of 1.3 percent each year. Underpinned by the techniques of computing virtualization in cloud computing, network resource virtualization, Network Functions Virtualization (NFV) has been emerging as a technology to reduce the high CAPEX and OPEX of network providers [3], [9], [13], [17], by deploying networking services as software in Virtual Machines (VMs).

In this paper we consider an NFV-enabled network that consists of data centers and switches interconnected by links, providing various network services as virtualized network functions (VNFs) in the data centers. Since the primary goal of network service providers is to maximize their profits by fully utilizing their resources, one fundamental problem for them is to efficiently allocate VNFs such that the network throughput is maximized, while the cost of realizing requests is minimized. This is a fundamentally challenging problem. Unlike conventional service requests without service chain requirements, a user request for network services here requires that its traffic is steered along a sequence of VNFs in the orders of its service chain, prior to its destination. Furthermore, different requests have different stringent end-to-end delay requirements. Meeting such stringent user requirements is crucial to guarantee the quality of network services and user satisfication.

There are several studies focusing on the provisioning of network services via the NFV technique [8], [9], [14], [18], [20]. Some of them aim to developing novel architectures and building systems for NFV-enabled networks by formulating Integer Linear Programming (IPL) solutions to optimize network performance, e.g., network throughput [3], [12], [20]. Such ILP solutions however suffer from poor scalability when the problem size is quite large. Others either do not consider either the end-to-end delay requirement of user requests or ignore the computing resource constraint [9]. This can significantly degrade the quality of network services. Therefore, efficient and scalable algorithms with performance guarantees are urgently needed to enable the NFV technique in networks.

Unlike the mentioned studies, we study the throughput maximization problem in an NFV-enabled network under the assumption that limited numbers of instances of service chains have been instantiated in data centers in advance. We aim to admit as many user requests as possible while minimizing their implementation costs and meeting their end-to-end delay requirements. We will develop efficient and scalable optimal and approximation algorithms with performance guarantees for the problem.

The main contributions of this paper are as follows. We first formulate the throughput maximization problem in an NFV-enabled network consisting of multiple data centers and switch nodes, and show that the problem is NP-hard. We then devised an optimal solution when all requests have identical packet rates. Otherwise, we propose two approximation algorithms with provable approximation ratios, depending on whether the packet traffic of each request is splittable. We finally evaluate the performance of the proposed algorithms.

The rest of the paper is organized as follows. Section II reviews related work. Section III introduces the system model and notations, and define the problem. Section IV proposes an optimal algorithm for a special case of the problem when all requests have identical packet rates; otherwise approximation algorithms are proposed in Section V. Section VI evaluates the performance of the proposed algorithms through simulations, and Section VII concludes the paper.

978-1-4673-8999-0/17/$31.00 ©2017 IEEE

II. Related Work

Much recent attention has been focusing on the placement of virtualized network functions (VNF) [14], [4], traffic steering given placed network functions [17], joint traffic steering and VNF placement [9], and dynamic network function chaining [20]. For example, Qazi *et al.* developed SIMPLE [17] that enforces high-level routing policies for middlebox-specific traffic, they however did not consider virtualization or dynamic network function placements. Martins *et al.* [14] introduced a platform to improve network performance, by revising existing virtualization technologies to support the deployment of modular, virtual middleboxes on lightweight VMs. Qu *et al.* [18] studied the problem of delay-aware scheduling and resource optimization with NFV in a virtual network. Wang *et al.* [20] studied the problem of dynamic network function composition, and proposed a distributed algorithm, using Markov approximation method for the problem. However, most of the mentioned studies that are designed for communication networks may not be suitable for an NFV-enabled network consisting of multiple data centers, since they assumed that each network function is solely used by a user request. Although there are extensive studies on resource allocations for Virtual Machines (VMs) [15], [19], most of them do not jointly consider routing and VNF placement. Their solutions thus cannot be directly applied into NFV-enabled networks.

There are several studies focusing on the provisioning of network services in cloud platforms [3], [8], [9], [12]. Most of them focused on a single data center [8], [9], [12]. Li *et al.* [12] aim to provide real-time guarantees for user requests in a data center. Gu *et al.* [8] investigated dynamic service chaining in an NFV market of a single data center, by devising efficient and truthful auction mechanisms and assuming some of the instantiated network functions can be reused by later requests. Their solutions however may not be applicable to an NFV-enabled network with geo-graphically distributed data centers. They focused on developing Integer Linear Programming (ILP) solutions or simulated annealing algorithms that are not scalable or take prohibitively long time to converge.

III. Preliminaries

In this section, we first introduce the system model and notations, and then define the problem precisely.

A. System model

We consider a network $G = (V \cup \mathcal{DC}, E)$ operated by a cloud service provider, where V is the set of switches, $\mathcal{DC}$ is the set of data centers connected to some of the switches, and $|\mathcal{DC}| \ll |V|$. E is the set of links between switches and switches and data centers. Each data center $DC_i \in \mathcal{DC}$ has limited computing and storage resources to implement network functions in software that run in Virtual Machines (VMs), referred to as VNFs. An ordered sequence of VNFs is defined as a *service chain* [20], the service chains of all user requests are classified into K types. There are a given number of instances for each type of service chains in each data center. Provisioning such service chain instances at different data centers incurs different costs, as servers in different data centers have different amounts of energy consumptions [21], [22], [23], [24]. Furthermore, data transfers at each link $e \in E$ incur transmission delays. Let d_e be the delay of implementing a unit packet along link e. Figure 1 is an example of a software-defined network.

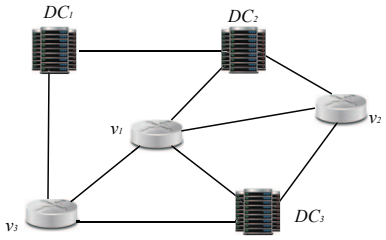


Fig. 1. An NFV-enabled network G with a set $\mathcal{DC} = \{DC_1, DC_2, DC_3\}$ of data centers that are connected by a set $V = \{v_1, v_2, v_3\}$ of switches.

B. Service chains, user requests, and SLA requirements

We consider K types of service chains with each type of service chains having a different sequence of VNFs. We assume that the number of instances for each type of service chains has been instantiated at each data center DC_i, and these instances can be reused by later requests. Let $\mathcal{SC}_i^k$ be the set of instances of type-k service chains at data center DC_i, and denote by $|\mathcal{SC}_i^k|$ the number of instances of a type-k service chain at DC_i with $1 \leq k \leq K$. Following existing studies [3], [8], we assume that each instance SC_i^k of a type-k service chain in data center DC_i represents an atomic network service. Therefore, each service chain instance is allocated with enough computing resources that can process a *minimum packet rate* ρ, and different instances of the same type in DC_i can be composed together to handle requests with higher packet rates.

We consider a request r_j that routes packets from a source node s_j to a destination node t_j with a given packet rate ρ_j, such that its traffic passes through one instance of a type-k of service chains. Furthermore, request r_j usually has an *end-to-end delay requirement* that specifies the maximum time experienced by its traffic from the source node to the destination node in terms of both the processing delay at a data center and the transfer delay at links. Let D_j be the end-to-end delay requirement of r_j. Assuming an instance SC_i^k of type-k service chains at data center DC_i is assigned to process the traffic of r_j, then the delay experienced by r_j from s_j to t_j consists of the transfer delay $d(s_j, DC_i)$ from s_j to DC_i, the processing delay $d(SC_i^k)$ by instance SC_i^k at data center DC_i, and the transfer delay $d(DC_i, t_j)$ from DC_i to t_j. The end-to-end delay requirement D_j of r_j is

$$d(s_j, DC_i) + d(SC_i^k) + d(DC_i, t_j) \leq D_j, \qquad (1)$$

For simplicity, r_j is represented by $r_j = (s_j, t_j; SC^k, \rho_j, D_j)$

C. Cost model

Cloud service providers provide network services on a pay-as-you-go basis [21], [22], [23], [24], and aim to maximize their profits through minimizing the cost of implementing requests. Specifically, *the implementation cost* of request $r_j = (s_j, t_j; SC^k, \rho_j, D_j)$ consists of the cost of computing resource consumption, i.e., the use of an instance of type-k service chains at a data center DC_i, and the communication cost of transferring its traffic from s_j to the data center DC_i for

processing then transferring the processed data from DC_i to its destination t_j. Let $c(SC_i^k)$ be the cost of implementing an instance of a type-k service chain of r_j in DC_i, and $c(e)$ be the cost of transferring a unit packet rate for request r_j through link $e \in E$. To utilize bandwidth resources in an economical way, we assume that the traffic of request r_j is routed via shortest paths from its source to the chosen data center DC_i and from DC_i to its destination t_j, i.e., p_{s_j,DC_i} and p_{DC_i,t_j}. Then, the implementation cost $c(r_j)$ of r_j in the network is

$$c(r_j) = \min_{1 \le i \le |\mathcal{DC}|} \{\rho_j \Big(c(SC_i^k) + \sum_{e \in p_{s_j,DC_i}} c(e) + \sum_{e \in p_{DC_i,t_j}} c(e) \Big) \}, \quad (2)$$

where $p_{y,z}$ is the shortest path in G from node y to node z, and ρ_j is the packet rate of request r_j.

D. Problem definition

Given a network $G = (V \cup \mathcal{DC}, E)$, let $\mathcal{R}$ be a set of requests with each being represented by $r_j = (s_j, t_j; SC^k, \rho_j, D_j)$). *The throughput maximization problem in* G is to admit as many requests in $\mathcal{R}$ as possible while minimizing the accumulative implementation cost of all admitted requests, subject to network resources capacity constraints.

The decision version of the throughput maximization problem is NP-Complete, by a polynomial time reduction from the partition problem [5]. Specifically, given a set S of positive integers, the partition problem is to decide whether the integers in S can be partitioned into two subsets S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the numbers in S_2. Given any instance of the partition problem, we can construct a special case of the throughput maximization problem in a network G without considering implementation costs and end-to-end delay requirements of users, by adding a request for each integer in S with packet rate as the value of the integer and including two data centers with equal numbers of instances of a single type of service chains that can process a total packet rate that equal to the half of the sum of the integers in S. A solution to the partition problem will return a feasible solution to the throughput maximization problem, without taking into account the implementation costs and request end-to-end delay requirements.

IV. Algorithm with Identical Packet Rates

We consider the problem when all requests have identical packet rates ρ by devising an optimal algorithm.

A. Overview of the algorithm

Assuming that each request r_j with the minimum packet rate ρ denotes that one instance of its required type-k service chain is enough to process its traffic. Maximizing the throughput of network G thus is to admit as many requests as possible, by assigning each admitted request r_j to one instance of its type-k service chain, without violating the number $|\mathcal{SC}_i^k|$ of instances of a type-k service chain at data center DC_i. The basic idea of the proposed algorithm is to transfer the throughput maximization problem in G into the minimum-cost maximum flow problem in an auxiliary graph $G' = (V', E')$. The solution to the latter in turn will return a feasible solution to the former.

B. Algorithm

Given a set $\mathcal{R}$ of requests to be admitted by G, we now construct the auxiliary graph $G' = (V', E')$ as follows.

We first construct the node set V' of G'. For each data center DC_i, we add K *service chain nodes* into V' with each service chain node $\mathcal{SC}_i^k$ corresponding the set of instances of type-k service chains, i.e., $V' = V' \cup \{\mathcal{SC}_i^k \mid 1 \le k \le K, \ and \ 1 \le i \le |\mathcal{DC}|\}$. For each request $r_j \in \mathcal{R}$, a *request node* r_j is added into V' too, i.e., $V' = V' \cup \{r_j\}$. Furthermore, a *virtual source* s_0 and a *virtual sink* t_0 is added into V'.

We then add edges into set E' of G', and set edge capacities and costs. There is a directed edge from the virtual source s_0 to each request node r_j, i.e., $E' = E' \cup \{\langle s_0, r_j \rangle \mid 1 \le j \le |\mathcal{R}|\}$; its cost is set to zero, and capacity is set to 1. Also, there is a directed edge $\langle r_j, \mathcal{SC}_i^k \rangle$ in E' from each request node r_j to a service chain node $\mathcal{SC}_i^k$ if the sum of the transfer delay from its source s_j to DC_i, the process delay at DC_i, and the transfer delay from DC_i to t_j meets the delay requirement D_j of request r_j. The cost of edge $\langle r_j, \mathcal{SC}_i^k \rangle$ is set to the implementing cost of request r_j at DC_i, i.e., $c(\langle r_j, \mathcal{SC}_i^k \rangle) = \rho(\sum_{e \in p_{s_j,DC_i}} c(e) + \sum_{e \in p_{DC_i,t_j}} c(e) + c(SC_i^k))$. The capacity of edge $\langle r_j, \mathcal{SC}_i^k \rangle$ is set to 1. In addition, there is an edge $\langle \mathcal{SC}_i^k, t_0 \rangle$ from each type of service chain node $\mathcal{SC}_i^k$ to the virtual sink t_0. Its cost is zero, and its capacity is set to $|\mathcal{SC}_i^k|$, i.e., the number of available instances of a type-k service chain in DC_i. Fig. 2 illustrates an example of G'.

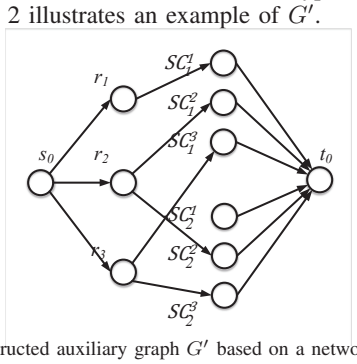


Fig. 2. A constructed auxiliary graph G' based on a network with two data centers DC_1 and DC_2 and three requests to be admitted, i.e., r_1, r_2, and r_3, where requests r_1, r_2, and r_3 require instances of type 1, 2, and 3 service chains, respectively, and DC_2 is too far from the source of r_1 to meet its delay requirement.

Having constructed the auxiliary graph G', the problem then is to find an integral minimum-cost maximum flow f in G' from s_0 to t_0 without violating the capacity constraints of edges in G'. The detailed algorithm is given in **Algorithm** 1.

C. Algorithm analysis

We now analyze the performance of the proposed algorithm.

Theorem 1. *Given a network $G(V \cup \mathcal{DC}, E)$ with a set V of switch nodes, a set $\mathcal{DC}$ of data centers that are interconnected by switches, a set $\mathcal{R}$ of requests that have identical packet rates ρ, and a set $\mathcal{SC}_i^k$ of instances of a type-k service chain at data center DC_i, there is an algorithm for this special case of the throughput maximization problem, i.e.,* **Algorithm** *1, which delivers an optimal solution.*

Proof. We first show that the algorithm delivers a feasible solution. This is to show a minimum cost maximum flow

Algorithm 1 `Optimal`

Input: A network $G(V \cup \mathcal{DC}, E)$, a set $\mathcal{R}$ of requests, an identical atomic packet rate ρ of all requests in $\mathcal{R}$, and a set $\mathcal{SC}_i^k$ of instances of type-k service chains in each data center $DC_i \in \mathcal{DC}$.

Output: Admit or reject each request in $\mathcal{R}$, and an assignment of admitted requests to instances of service chains in data centers in $\mathcal{DC}$.

1: Construct an auxiliary graph $G' = (V', E')$ from network $G(V \cup \mathcal{DC}, E)$, by adding a virtual source s_0, a virtual sink t_0, a request node r_j for each request in $\mathcal{R}$, and a service chain node $\mathcal{SC}_i^k$ for instances of a type-k service chain in each data center DC_i, and adding an edge from s_0 to each request node r_j, an edge from each request node r_j to a type-k service chain node $\mathcal{SC}_i^k$ if the delay requirement of r_j can be met, an edge from each service chain node $\mathcal{SC}_i^k$ to virtual sink t_0;
2: Set edge costs and capacities for each edge in E', by setting the capacity of the edge from each request node r_j to each service chain node $\mathcal{SC}_i^k$, i.e., $\langle r_j, \mathcal{SC}_i^k \rangle$, to 1, the capacity of edge from each service chain node $\mathcal{SC}_i^k$ to its corresponding data center node DC_i to $|\mathcal{SC}_i^k|$, and the capacity of edge from each data center node DC_i to t_0 to $|\mathcal{R}|$. The cost of edge $\langle r_j, \mathcal{SC}_i^k \rangle$ is set to $\rho(c(SC_i^k) + \sum_{e \in p_{s_j, DC_i}} c(e) + \sum_{e \in p_{DC_i, t_j}} c(e))$, and the cost of edge $\langle \mathcal{SC}_i^k, t_0 \rangle$ is set to 0, and its capacity is set to $|\mathcal{SC}_i^k|$;
3: Find a minimum cost maximum flow f in the auxiliary graph G' by applying the algorithm in [1];
4: The requests that are assigned into service chain node $\mathcal{SC}_i^k$ in the flow f will be processed by an instance of a type-k service chain in data center DC_i;
5: All other requests that are not assigned in flow f will be rejected;
6: **return** The assigned service chain for each admitted request, and requests that are rejected.

f from s_0 to t_0 in G' corresponds a feasible assignment of requests to data centers in $\mathcal{DC}$, and the delay requirement of each admitted request is met. Since the capacity of edge $\langle \mathcal{SC}_i^k, DC_i \rangle$ is set to $|\mathcal{SC}_i^k|$ that is the number of available instances of type-k service chains in DC_i, each of the requests that are assigned to $\mathcal{SC}_i^k$ in flow f will have an instance to process its traffic. In addition, it is clear that the delay requirement of an admitted request will be met, because there will not be an edge from a request node r_j to a data center that cannot meet its delay requirement in G'.

We then show that the algorithm delivers an optimal solution in polynomial time. The edge capacities of the constructed auxiliary graph G' are integral values. Following the well-known integrality property for the minimum-cost maximum flow problem [1], there is an integral minimum-cost maximum flow f, which can be found in polynomial time. That is, for each request node r_j and each service chain node $\mathcal{SC}_i^k$ at data center, the flow $f_{r_j, \mathcal{SC}_j^k}$ from r_j to $\mathcal{SC}_j^k$ is either 0 or 1, as the capacity of edge $\langle r_j, \mathcal{SC}_j^k \rangle$ is 1. □

V. Approximation Algorithms with Different Packet Rates

We now consider the throughput maximization problem where different requests may have different packet rates. We first devise an approximation algorithm for the problem by assuming that the total packet rate of the requests is no larger than the one that can be processed by the available instances of service chains of all data centers, that is $\sum_{r_j \in \mathcal{R}} \rho_j \leq \sum_{DC_i \in \mathcal{DC}, 1 \leq k \leq K} |\mathcal{SC}_i^k| \cdot \rho$, if the traffic of each request is splittable. Otherwise, we propose another approximation algorithm by extending the proposed approximation algorithm.

A. Approximation algorithm with splittable traffic

The basic idea of the algorithm is that we first treat each request into a number of *virtual requests* with each having a minimum packet rate ρ, and then transfer the problem into a minimum-cost multicommodity problem in an auxiliary graph $G'' = (V'', E'')$. The construction of G'' is similar to the auxiliary graph $G' = (V', E')$ in Section IV, with slightly different edge capacity settings.

Algorithm 2 `Appro-Split`

Input: A network $G(V \cup \mathcal{DC}, E)$, a set $\mathcal{R}$ of requests with each request r_j having a packet rate ρ_j, a set $\mathcal{SC}_i^k$ of instances of type-k service chains at each data center $DC_i \in \mathcal{DC}$, and the minimum packet rate ρ that can be processed by each service chain instance.

Output: Admit or reject each request in $\mathcal{R}$, and an assignment of admitted requests to instances of service chains in the data centers in $\mathcal{DC}$.

1: Let ρ_{max} and ρ_{min} be the maximum and minimum packet rates of all requests in $\mathcal{R}$, respectively;
2: Assume that $\rho = \rho_{min}$;
3: Divide each request $r_j \in \mathcal{R}$ into γ_j ($= \frac{\rho_j}{\rho}$) virtual requests r'_{j1}, r'_{j2}, ..., $r'_{j\gamma_j}$ with each virtual request having a packet rate of ρ, assuming that ρ_j is dividable by ρ;
4: Construct an auxiliary graph $G'' = (V'', E'')$ following the construction procedure of algorithm 1, i.e., steps 1 and 2 in algorithm 1.
5: Set the capacities for edges $\langle s_0, r_j \rangle$ and $\langle r_j, \mathcal{SC}_i^k \rangle$ to γ_j, and the capacities, costs of all other edges are the same as those in G';
6: Find a minimum-cost multicommodity flow f' from s_0 to t_0 in G'' by invoking the algorithm due to [6], by considering each r_j as a commodity with demand γ_j that needs to be routed from s_0 to t_0 in G'';
7: For each request r_j, its traffic may be processed by its required instances in multiple data centers, if its demand is routed into multiple data centers in flow f';
8: **return** The assigned service chain for each admitted request, and the requests that are rejected.

We now detail the approximation algorithm. Let ρ_{max} and ρ_{min} be the maximum and minimum packet rates of requests in $\mathcal{R}$, respectively. Without loss of generality, we assume that $\gamma = \frac{\rho_{max}}{\rho_{min}}$ is a given constant and the packet rate ρ_j of request r_j is dividable by r_{min}. We further assume that $\rho = \rho_{min}$. We then treat each request into multiple virtual requests with each having a minimum packet rate ρ, by treating each request r_j as γ_j ($= \frac{\rho_j}{\rho}$) virtual requests r'_{j1}, r'_{j2}, ..., $r'_{j\gamma_j}$ with each virtual request having a packet rate of ρ.

We then construct the auxiliary graph $G''(V'', E'')$, by letting $V'' = V'$ and $E'' = E'$. The only difference between G'' and G' is the capacities for edges $\langle s_0, r_j \rangle$ and $\langle r_j, \mathcal{SC}_i^k \rangle$, which are both set to γ_j. The capacity and cost settings for all other edges are the same as those in G'.

Given the constructed auxiliary graph G'', we consider each request r_j as a commodity with demand γ_j that need to be routed from s_0 to t_0 in G''. We then find a minimum-cost multicommodity flow f' in G'', by using the fast approximation algorithm due to Garg and Könemann's algorithm [6]. The obtained flow f' corresponds to a splittable assignment of the virtual requests of each request into the data centers in network G. The details of the proposed algorithm is shown in **Algorithm** 2.

B. Approximation algorithm with unsplittable traffic

If the traffic of each request is not splittable, the solution delivered by **Algorithm** 2 is infeasible, since the virtual requests of each request may be assigned to different data centers for processing. To modify the solution to make it feasible, we perform adjustments such that the traffic of each admitted request is implemented by its service chain in a single data center. This however may violate the number of instances of service chains in that data center. To avoid such violations,

Algorithm 3 `Appro-Unsplit`

Input: A network $G(V \cup \mathcal{DC}, E)$, a set $\mathcal{R}$ of requests with each request r_j having a packet rate ρ_j, a set $\mathcal{SC}_i^k$ of instances of type-k service chains at each data center $DC_i \in \mathcal{DC}$, and the minimum packet rate ρ that can be processed by each service chain instance.

Output: Admit or reject each request in $\mathcal{R}$, and an assignment of admitted requests to instances of service chains in the data centers in $\mathcal{DC}$.

1: Invoke **Algorithm** 2 to obtain a solution that may assign the virtual requests of each request r_j into multiple data centers for processing;
2: For each request r_j, let DC_1, ..., DC_l, ..., DC_L be the L data centers to which its virtual requests are assigned. Denote by DC_{l_0} be the data center that is assigned with the highest number of virtual requests of r_j;
3: Move the virtual requests of r_j that are assigned to other data centers to data center DC_{l_0};
4: **return** The assigned service chain for each admitted request, and the requests that are rejected.

we can scale down the number of available instances of a type of service chain in each data center $DC_i \in \mathcal{DC}$ by a factor, and then apply **Algorithm** 2 and later adjustment. The detailed algorithm description is given below.

We first scale down the number of instances of type-k service chains at each data center DC_i by $|\mathcal{DC}|$, that is the number of instances in set $\mathcal{SC}_i^k$ is $\lfloor \frac{|\mathcal{SC}_i^k|}{|\mathcal{DC}|} \rfloor$. We then apply **Algorithm** 2. To obtain a feasible solution from the one obtained by **Algorithm** 2, we modify the solution by merging the virtual requests derived from a single request to one of their assigned data centers. Specifically, for each request r_j whose virtual requests are assigned to multiple data centers. Let DC_1, ..., DC_l, ..., DC_L be the L data centers to which the virtual requests of r_j are assigned, where $2 \leq l \leq |\mathcal{DC}|$. Denote by DC_{l_0} be the data center with the maximum number of virtual requests of r_j. We merge the virtual requests assigned to other data centers to the ones in data center DC_{l_0}. The proposed approximation algorithm is described in **Algorithm** 3.

C. Algorithm analysis

We now analyze the performance of algorithms 2 and 3.

Theorem 2. *Given a network $G(V \cup \mathcal{DC}, E)$ with a set $\mathcal{R}$ of requests with each having a packet rate of ρ_j, and a set $\mathcal{SC}_i^k$ of instances of type-k service chains with each being able to process a minimum packet rate ρ, assume that the ratio of the maximum packet rate ρ_{max} and minimum packet rate ρ_{min} of all requests is a given constant and $\rho = \rho_{min}$. There is an approximation algorithm for the throughput maximization problem, i.e.,* **Algorithm** *2, that delivers a feasible solution with a throughput being $(1 - 3\epsilon)$ times of the optimal while the implementation cost is optimal, in time $O^*(K^2|\mathcal{DC}|^2|\mathcal{R}|^2 + (|V| + |\mathcal{DC}|)^2)$[1], where ϵ is the accuracy parameter in Garg and Könemman's algorithm with $0 < \epsilon \leq 1/3$.*

Proof. Clearly, the solution obtained is feasible if the traffic of each request r_j is allowed to be split into different data centers, following Theorem 1. In terms of the approximation ratio, since the solution obtained by Garg and Könemman's algorithm directly translates into a feasible solution to the throughput maximization problem, the approximation ratio thus is the same as the Garg and Könemman's algorithm, i.e., at least $1 - 3\epsilon$ times of the optimal [6].

[1] $O^*(f(n)) = O(f(n) \cdot \log^{O(1)} n)$

We now analyze the running time of the proposed algorithm. It can be seen that the algorithm consists of two phases: (1) the construction of auxiliary graph G'', and (2) invoking Garg and Könemman's algorithm. Phase (1) takes $O(|V \cup \mathcal{DC}|^2)$ time, while phase (2) takes $O^*(\epsilon^{-2}m(n+m))$ time [6], where $m = |E''|$ and $n = |V''|$, while $|V''| = |\mathcal{R}| + (K+1)|\mathcal{DC}| + 2$ and $|E''| = O(|\mathcal{R}|(1+K|\mathcal{DC}|)+(K+1)|\mathcal{DC}|) = O(K \cdot |\mathcal{R}| \cdot |\mathcal{DC}|)$, the running time of **Algorithm** 2 thus is $O^*(K^2|\mathcal{DC}|^2|\mathcal{R}|^2 + (|V| + |\mathcal{DC}|)^2)$. □

Theorem 3. *Given a network $G(V \cup \mathcal{DC}, E)$ with a set $\mathcal{R}$ of requests with each having a packet rate of ρ_j, and a set $\mathcal{SC}_i^k$ of instances of type-k service chains with each being able to process a minimum packet rate ρ, assume that the ratio of the maximum packet rate ρ_{max} and minimum packet rate ρ_{min} of all requests is a given constant and $\rho = \rho_{min}$. There is an approximation algorithm for the throughput maximization problem, i.e.,* **Algorithm** *3 that delivers a solution that achieves a throughput of at least $\frac{1-3\epsilon}{|\mathcal{DC}|}$ times of the optimal with the implementation cost being no more than $|\mathcal{DC}|$ times of the optimal, where $|\mathcal{DC}|$ can be considered as a given constant with $|\mathcal{DC}| \ll |V|$.*

Proof. Clearly, the solution is feasible, as (1) each admitted request is assigned to an instance of its type of service chains, and (2) its end-to-end delay requirement is met, following similar derivation in Theorem 1.

We now show the approximation ratio on the throughput of **Algorithm** 3. Since the available number of each type of service chain at a data center is scaled down by a factor of $|\mathcal{DC}|$, it is clear that the throughput achieved by **Algorithm** 3 is no less than $\frac{(1-3\epsilon)}{|\mathcal{DC}|}$ times of the optimal.

We finally analyze the approximation ratio on the implementation cost of the solution by **Algorithm** 3. Denote by c the implementation cost by **Algorithm** 3 of all admitted requests, and c_j the implementation cost of request r_j. Let c' be the implementation cost due to **Algorithm** 2 by treating r_j as $\lceil \frac{\rho_j}{\rho_{min}} \rceil$ number of virtual requests. Note that c is achieved by merging the virtual requests of r_j that are assigned to different data centers. Specifically, if the virtual requests of r_j are assigned to data centers DC_1, ..., DC_l, ..., DC_L, all other virtual requests are assigned to data center DC_{l_0} that is assigned with the highest number of virtual requests. Let $c_l'^k$ be the implementation cost of the virtual requests assigned to service chain SC_l^k at data center DC_l. Clearly, $\sum_{l=1}^{L} c_l'^k > c_{l_0}^k$. After moving all virtual requests to DC_{l_0}, the implementation cost of a unit packet rate will be the same as that of DC_{l_0}, while the cost c_l^k will depend on how many virtual requests are moved from DC_l to DC_{l_0}. In the worst case there are $\frac{\gamma}{L} < \frac{\gamma}{2}$ virtual requests in data center DC_l that are moved to DC_{l_0}, since $2 \leq L \leq |\mathcal{DC}|$. Therefore, $c_l^k \leq c_{l_0}^k$. This means that the implementation cost c_j of r_j at DC_{l_0}, can be maximally $L \cdot c_{l_0}^k < |\mathcal{DC}| \cdot c_{l_0}^k$, i.e., $c_j < |\mathcal{DC}| \cdot c_{l_0}^k$. We then have

$$\begin{aligned} c' &= \sum_{r_j \in \mathcal{R}} \sum_{l=1}^{L} c_l'^k \geq \sum_{r_j \in \mathcal{R}} c_{l_0}^k, \quad \text{since } \sum_{l=1}^{\gamma_j} c_l'^k > c_{l_0}^k, \\ &> \sum_{r_j \in \mathcal{R}} \frac{c_j}{|\mathcal{DC}|}, \quad \text{since } c_j < |DC| \cdot c_{l_0}^k, \quad \geq \frac{c}{|\mathcal{DC}|}. \end{aligned} \tag{3}$$

In other words, we have $c \leq |\mathcal{DC}|c'$, meaning that the implementation cost of admitted requests will be no greater than $|\mathcal{DC}|$ times of the cost by the solution achieved through treating each request r_j as γ_j virtual requests. Let c^* be the optimal cost of the optimal solution to the throughput maximization problem. Denote by c'^* the optimal cost of the solution by treating each request as a number of virtual requests. According to Theorem 2, we have $c' = c'^*$. Clearly, $c'^* < c^*$ as each virtual request of a request is moved to the service chain with the maximum implementation cost of all the virtual requests. We thus have $c \leq |\mathcal{DC}|c' = |\mathcal{DC}|c'^* < |\mathcal{DC}|c^*$. This means that the implementation cost of the solution by **Algorithm** 2 is no more than γ times of the cost of the optimal solution. □

VI. Simulations

In this section, we evaluate the performance of the proposed algorithms through experimental simulations.

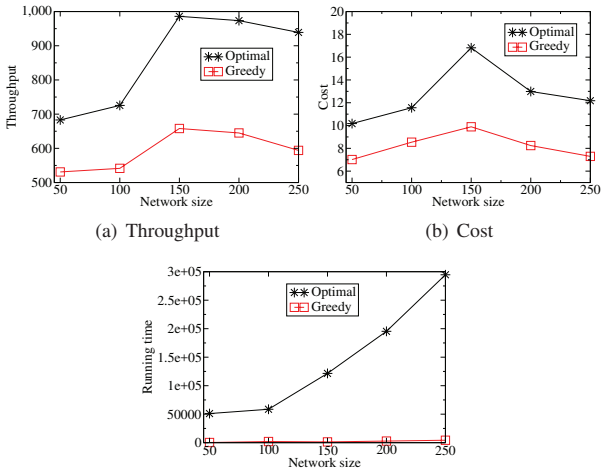
A. Experiment Settings

We consider networks that are generated by the tool GT-ITM [7]. The transmission delay of a link varies between 2 milliseconds (ms) and 5 ms [11]. The costs of transmitting and processing 1 GB (approximately 16,384 packets with each having size of 64 KB) of data are set within [$0.05, $0.12] and [$0.15, $0.22], respectively, following typical charges in Amazon EC2 with small variations [2]. We consider five categories of network functions: Firewall, Proxy, NAT, IDS, and Load Balancing (LB). The processing delay of a packet for each network function is randomly drawn from 0.045 ms to 0.3 ms [14], and the processing delay of a service chain instance is the total processing delay of its network functions, where each service chain instance has at most five network functions. The number of service chain types K is 5. The number of each type of service chains in a data center is randomly drawn from $[10, 50]$. The minimum packet rate of a service chain instance is set to 400 packets/second [14]. Each request $r_j \in \mathcal{R}$ is generated as follows, given a network $G = (V \cup \mathcal{DC}, E)$, two nodes from V are randomly drawn as its source s_j and destination t_j. Its packet rate ρ_j is randomly drawn from 400 to $4,000$ packets/second [12], the delay requirement varies from 10 ms to 100 ms [16], and its type of service chains is randomly assigned from one of the five types. The running time is obtained based on a machine with a 3.40GHz Intel i7 Quad-core CPU and 16 GiB RAM. Unless otherwise specified, these parameters will be adopted in the default setting.

We compare the proposed algorithms with a greedy algorithm that aims to maximize the throughput by admitting requests with small packet rates first. Specifically, the greedy algorithm first sorts the requests in increasing order of their packet rates. It then assigns the requests one by one, through implementing each request in a data center that not only meets its delay requirement but also has the maximum number of available service chain instances. For simplicity, we refer to this greedy algorithm as algorithm `Greedy`.

B. Performance evaluation with identical packet rates

We first study the performance of algorithms `Optimal` and `Greedy` by varying the number of switches $|V|$ of network G from 50 to 250, fixing the *switch-to-datacenter* ratio $\frac{|V|}{|\mathcal{DC}|}$ at 10. Fig. 3 shows the results, and from Fig. 3 (a) and 3 (b) we can see that algorithm `Optimal` achieves a throughput at least 30% percent more than that by algorithm `Greedy`, while `Optimal` has a higher implementation cost for all the admitted requests. The reason is that algorithm `Greedy` selects the data center with the most available number of service chain instances, leading to some requests being rejected due to that their nearby data centers (that can meet delay requirements) do not have enough available number of service chain instances to implement the requests. Furthermore, as shown in Fig. 3 (a), the throughputs by all algorithms increase when the network size varies from 50 to 150 and then decrease when the network size is larger than 150. The rationale behind is that, with the increase of network size, more service chain instances will be available to admit more requests as more data centers will be available; however, when the network size keeps growing, each request with higher probabilities of being implemented in longer paths with more links, thereby increasing the probability of being rejected due to the violation of its delay requirement. In addition, it can be seen from Fig. 3 (c) that algorithm `Optimal` takes more time than that by algorithm `Greedy` to deliver a much better solution.



(a) Throughput (b) Cost

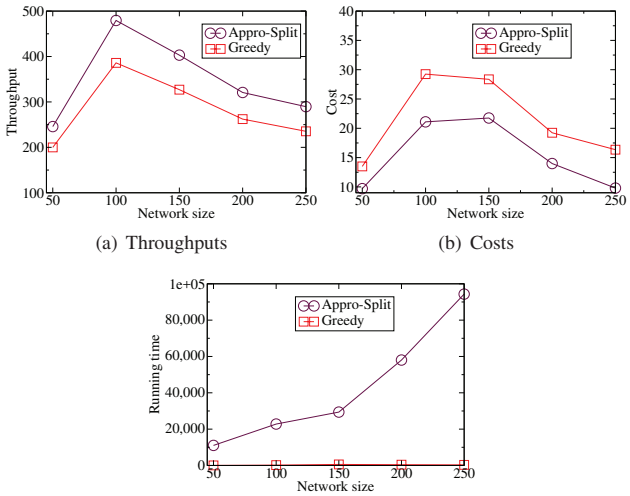
(c) Running time

Fig. 3. The performance of algorithms `Optimal` and `Greedy`.

C. Performance evaluation with different packet rates

We first study the performance of algorithms `Appro-Split` and `Greedy` by varying the network size from 50 to 250 while fixing the *switch-to-datacenter* ratio at 10. We can see from Fig. 4 (a) that the number of requests admitted by algorithm `Greedy` is around 80% of that by algorithm `Appro-Split` while more than 20% cost spent by algorithm `Appro-Split`. For example, when the network size is 100, algorithm `Appro-Split` admits around 200 more requests than that by algorithm `Greedy`.

We finally compare the performance of algorithm `Appro-Unsplit` with that of algorithm `Greedy` by varying network size from 50 to 250 while fixing the *switch-to-datacenter* ratio at 10. We can see from From Fig. 5 it can be seen that algorithm `Appro-Unsplit` achieves a 15% more throughput than that by algorithm `Greedy`. Furthermore, by

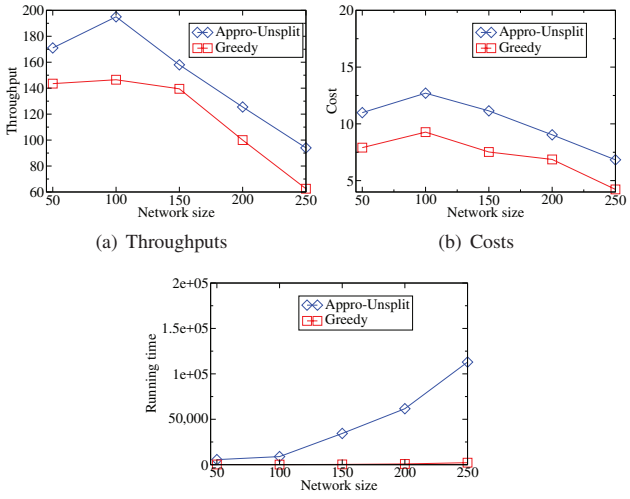


(a) Throughputs (b) Costs

(c) Running times

Fig. 4. The performance of algorithms `Appro-Split` and `Greedy`.

comparing the performance of algorithm `Appro-Split` and algorithm `Appro-Unsplit` in Fig. 4 and Fig. 5, it can be seen that algorithm `Appro-Split` admits more requests than algorithm `Appro-Unsplit`.



(a) Throughputs (b) Costs

(c) Running times

Fig. 5. The performance of algorithms `Appro-Unsplit` and `Greedy`.

VII. Conclusion

In this paper we investigated the throughput maximization problem in an NFV-enabled network, where limited numbers of instances of each type of service chains are instantiated at its data centers. We first proposed an optimal algorithm for a special case of the problem where all requests have identical packet rates. We also devised two approximation algorithms with provable approximation ratios for the problem when different requests different packet rates, depending on whether the traffic of each request is splittable. Experimental results demonstrated that the performance of the proposed algorithms outperform a greedy algorithm by achieving at least 15% more throughput. It must be mentioned that the proposed algorithms can be easily integrated into network resource orchestrators as they achieve near-optimal throughput in a scalable and efficient way.

Acknowledgement

The work done by Zichuan Xu and Alex Galis in this paper was partially supported by 5G Exchange innovation project (https://5gex.tmit.bme.hu) and by SONATA innovation project (http://sonata-nfv.eu) co-funded by the European Union under the Horizon 2020 EU Framework Programme.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.
[2] Amazon Web Services, Inc. Amazon ec2 instance configuration. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-ec2-config.html.
[3] G. Cheng, H. Chen, H. Hu, Z. Wang, and J. Lan. Enabling network function combination via service chain instantiation. *Computer Networks*, Vol. 92, pp.396–407, Elsevier, 2015.
[4] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca. The dynamic placement of virtual network functions. *Proc. of the Network Operations and Management Symposium* (NOMS), IEEE, 2014.
[5] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-Completeness*. W.H. Freeman, 1997.
[6] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *Proc. of FOCS'98*, IEEE, 1998.
[7] http://www.cc.gatech.edu/projects/gtitm/.
[8] S. Gu, Z. Li, C. Wu, and C. Huang. An efficient auction mechanism for service chains in the NFV market. *Proc. of INFOCOM'16*, IEEE, 2016.
[9] M. Huang, W. Liang, Z. Xu, M. Jia, and S. Guo. Throughput maximization in software-defined networks with consolidated middleboxes. *Proc. of LCN'16*, IEEE, 2016.
[10] IHS. *http://www.infonetics.com/pr/2015/1H15-Service-Provider-Capex.asp*.
[11] S. Knight *et al.* The internet topology zoo. *J. Selected Areas in Communications*, Volume 29, pp. 1765–1775, IEEE, 2011.
[12] Y. Li, L. T. X. Phan, and B. T. Loo. Network functions virtualization with soft real-time guarantees. *Proc. of INFOCOM*, IEEE, 2016.
[13] L. Mamatas, S. Clayman, and A. Galis. Information exchange management as a service for network function virtualization environments. *IEEE Transactions on Network and Service Management*, Vol. 13, No. 3, pp. 564-577, IEEE, 2016.
[14] J. Martins *et. al.* ClickOS and the art of network function virtualization. *Proc. of NSDI '14*, USENIX, 2014.
[15] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. *Proc. of INFOCOM'10*, IEEE, 2010.
[16] Microsoft. Plan network requirements for Skype for business. https://technet.microsoft.com/en-us/library/gg425841.aspx, 2015.
[17] Z. A. Qazi, C. C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu. SIMPLE-fying middlebox policy enforcement using SDN. *Proc. SIGCOMM '13*, ACM, 2013.
[18] L. Qu, C. Assi, and K. Shaban. Delay-aware scheduling and resource optimization with network function virtualization. To appear in *IEEE Transactions on Communications*, IEEE, 2016.
[19] V. Shrivastava, P. Zerfos, K. Lee, H. Jamjoom, Y. Liu, and S. Banerjee. Application-aware virtual machine migration in data centers. *Proc. of INFOCOM'11*, IEEE, 2011.
[20] P. Wang, J. Lan, X. Zhang, Y. Hu, and S. Chen. Dynamic function composition for network service chain: Model and optimization. *Computer Networks*, Vol. 92, pp. 408–418, Elsevier, 2015.
[21] Z. Xu and W. Liang. Minimizing the operational cost of data centers via geographical electricity price diversity, *Proc. of 6th IEEE International Conference on Cloud Computing*, IEEE, 2013.
[22] Z. Xu and W. Liang. Operational cost minimization for distributed data centers through exploring electricity price diversity. *Computer Networks*, Vol. 83, pp.59-75, Elsevier, 2015.
[23] Z. Xu, W. Liang, and Q. Xia. Electricity cost minimization in distributed clouds by exploring heterogeneities of cloud resources and user demands. *Proc. of ICPADS'15*, IEEE, 2015.
[24] Z. Xu, W. Liang, and Q. Xia. Efficient embedding of virtual networks to distributed clouds via exploring periodic resource demands. To appear in *IEEE Transactions on Cloud Computing*, Vol.XX, IEEE, 2016.