Throughput Optimization for Admitting NFV-Enabled Requests in Cloud Networks

Zichuan Xu[1], Weifa Liang[2], Alex Galis[3], Yu Ma[2], Qiufen Xia[1,*], Wenzheng Xu[4]

Abstract

Network softwarization is emerging as a techno-economic transformation trend that impacts the way that network service providers deliver their network services significantly. As a key ingredient of such a trend, network function virtualization (NFV) is shown to enable elastic and inexpensive network services for next-generation networks, through deploying flexible virtualized network functions (VNFs) running in virtual computing platforms. Different VNFs can be chained together to form different service chains for different network services, to meet various user data routing demands. From the service provider point of view, such services are usually implemented by VNF instances in a cloudlet network consisting of a set of data centers and switches. In this paper we consider provisioning network services in a cloud network for implementing VNF instances of service chains, where the VNF instances in each data center are partitioned into K types with each hosting one type of service chain. We investigate 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 instantiated in data centers. We first show the problem is NP-Complete, and propose an optimal algorithm for a special case of the problem when all requests have identical packet rates; otherwise, we devise two approximation algorithms with approximation ratios, depending on whether the packet traffic of each request is splittable. If arrivals of future requests are not known in advance, we study the online throughput maximization problem by proposing an online algorithm with a competitive ratio. We finally conduct experiments to evaluate the performance of the proposed algorithms by simulations. Simulation results show that the performance of the proposed algorithms are promising.

Keywords: Throughput maximization; cost minimization; approximation algorithms; online algorithms; network function virtualization; algorithm analysis.

[*]Corresponding author. Tel.: +8641162274368. Email address: qiufenxia@dlut.edu.cn

Email addresses: `z.xu@dlut.edu.cn` (Zichuan Xu), `wliang@cs.anu.edu.au` (Weifa Liang), `a.galis@ucl.ac.uk` (Alex Galis), `u5108648@anu.edu.au` (Yu Ma), `qiufenxia@dlut.edu.cn` (Qiufen Xia), `wenzheng.xu@scu.edu.cn` (Wenzheng Xu)

[1]Dalian University of Technology, P. R. China

[2]the Australian National University, Australia

[3]University College London, UK

[4]Sichuan University, Chengdu, P. R. China

1. Introduction

Network services traditionally make use of dedicated devices and equipments 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 may continuously purchase, add and operate new physical equipments into their operational networks. This usually leads to a high capital expenditure (CAPEX) and operational expenditure (OPEX) to purchase and manage the deployed network equipments. For example, it is expected that the total CAPEX of worldwide network service providers reaches US$374 billion in 2019 [18] 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 softwarization enabled by the technique of Network Function Virtualization (NFV) is emerged as a paradigm to bring significant benefits in terms of reducing the CAPEX and OPEX of network service providers [5, 17, 22, 26], by deploying network services as software running in virtualized resources.

In this paper we consider a cloud network that is operated by a network service provider and consists of data centers and switches interconnected by links, providing various network services as virtualized network functions (VNFs) in its data centers. To enable different types of network services, a number of instances for different types of VNFs are instantiated in each data center, where it is assumed that there are K different types of service chains. Considering that network service providers aim to maximize their profits by fully utilizing the instantiated instances of service chains, one fundamental and challenging problem for them is how to efficiently allocate VNF instances such that the network throughput is maximized, while the cost of realizing user requests is minimized. Unlike conventional user requests, each *NFV-enabled request* has a service chain requirement that requires to steer its traffic along a sequence of VNFs in a specified orderprior to reaching its destination. Furthermore, different requests have different stringent end-to-end delay requirements. Meeting such stringent requirements is crucial to guarantee the quality of network services and user satisfactions.

There are several studies focusing on the provisioning of network services via the NFV technique [13, 17, 21, 23, 27, 32]. Most of them however assumed that the instances of service chains cannot be reused among different requests and for each newly arrived request it has to instantiate a new instance of its required service chain for its implementation. This significantly reduces the network throughput and increases the implementation cost due to new service instantiation, since some network services, e.g., anti-virus services, can be instantiated only once and then reused by later requests. On the other hand, some of them developed novel architectures and built systems for networks to support the NFV technique, by formulating Integer Linear Programming (ILP) solutions with the aim to optimize network performance, e.g., network throughput [5, 20, 32]. Such ILP solutions however suffer from poor scalability when the problem size is quite large. Others did not consider the end-to-end delay requirement of user requests or ignore the computing resource constraint [17]. This can significantly degrade the quality of network services.

To the best of our knowledge, we are the first to study the throughput maximization problem in a cloud network consisting of multiple data centers to provide different types of service chains, where a number of instances of service chains have been instantiated in each data center in advance. We aim to admit as many user requests as possible while minimizing their accumulative implementation costs and meeting their end-to-end delay requirements. We consider a dynamic setting with user requests arriving at the network in an online manner, and VNF instances in each data center can be reused by later requests to further maximize the throughput and reduce the implementation cost. In addition, unlike existing studies focusing on heuristic solutions without performance guarantees, we devise the very first approximation and online algorithms with performance guarantee for the problem, which describes the distances of the obtained solutions from the optimal ones via theoretical analysis. Specifically, we develop efficient approximation algorithms with approximation ratios for the problem. In addition, if there is no knowledge of future request arrivals, we study the online throughput maximization problem by devising an online algorithm with a provable competitive ratio.

The main contributions of this paper are as follows. We first formulate the throughput maximization problem in a cloud network and show that the problem is NP-Complete. We then devised an optimal solution if 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 also devised a primal-dual online algorithm with a competitive ratio, if the future arrivals of user requests are not known in advance. We finally evaluate the performance of the proposed algorithms.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces the system model and notations, and define the problem. Section 4 presents an exact solution to the problem. Section 5 proposes novel algorithms for the throughput maximization problem if the request arrivals are known in advance; otherwise a new online algorithm with a competitive ratio is devised for the online throughput maximization problem in Section 6. Section 7 evaluates the performance of the proposed algorithms through simulations, and Section 8 concludes the paper.

2. Related Work

The NFV technique is expected to enable network service providers to use software to set up, configure, and manage network services in their networks automatically and dynamically [3, 23, 28]. In particular, NFV enables elastic resource usage on a pay-as-you-go basis. This means that virtualized network functions of different network services can be instantiated once by a network service provider and reused among different user requests.

Much recent attention has been focusing on the placement of virtualized network functions (VNF) [23, 6, 31], traffic steering given placed network functions [26], joint traffic steering and VNF placement [17], availability of service chains [7, 36, 35], and dynamic network function chaining [32]. For example, Qazi *et al.* developed SIMPLE [26] that enforces high-level routing policies for middlebox-specific traffic, they however did not consider virtualization or dynamic network function placements. Fayazbakhsh *et al.* proposed FlowTags [8] for flow scheduling in a network in the presence of dynamic modifications performed by middleboxes. Martins *et al.* [23] 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.* [27] studied the problem of delay-aware scheduling and resource optimization with NFV in a virtual network. Wang *et al.* [32] studied the problem of dynamic network function composition, and proposed a distributed algorithm, using Markov approximation method for the problem. Huang *et al.* [17] studied the problem of jointly routing of user requests and placing their required network functions to some servers in a data center, with the aim to maximize the network throughput while meeting various capacity constraints of the network and the end-to-end delay requirement of each user requests. Vizarreta *et al.* [31] investigated the VNF placement with an aim to reduce the expenditures incurred by function placements, and the availability of VNF functions for offering network services is considered. An ILP based solution is proposed for small problem sizes, and then an efficient heuristic is adopted for large scale networks. Dynamic placement of VNFs and re-use of existing VNF instances however are not considered. Most of the mentioned studies that are designed for communication networks may not be suitable for a cloud 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) [24, 29], most of them do not jointly consider routing and VNF placement. Their solutions thus cannot be directly applied into networks that support the NFV technique.

There are several studies focusing on the provisioning of network services in cloud platforms [5, 13, 17, 20, 9]. Most of them focused on a single data center [13, 17, 20]. Li *et al.* [20] aim to provide real-time guarantees for user requests in a data center. Gu *et al.* [13] 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 a cloud network with geo-graphically distributed data centers. Cheng *et al.* [5] studied a similar problem in an SDN with some switches having instances of network functions, which can be shared among different service chains. They focused on developing Integer Linear Programming (ILP) solutions or simulated annealing algorithms that are not scalable or take prohibitively long time to converge. Feng *et al.* [9] consider the provisioning of service chains in a distributed cloud, by proposing approximation algorithms for the joint NFV placement and routing. However, the end-to-end delay requirement of each user request is not considered. In our earlier conference version [35], we performed preliminary study on the throughput maximization problem in a cloud network. Based on the study, we investigate a new throughput maximization problem with the objective to maximize the network throughput, while minimizing the implementation cost of all admitted requests, assuming that the requests dynamically arrive at the network one-by-one. We also propose an exact solution to the problem. More evaluation results on the performance of all algorithms on real network topologies are also conducted.

3. Preliminaries

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

3.1. 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 computing resource capacity to implement network functions as software, referred to as VNFs. Following existing studies [33, 34], we focus on inter-datacenter networking of the data centers in $\mathcal{DC}$. Provisioning VNFs at different data centers incurs different costs, as servers in different data centers have different amounts of energy consumptions [33, 34]. 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 cloud network.

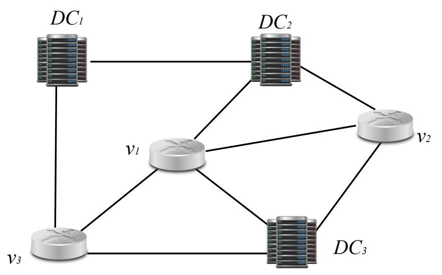


Figure 1: A cloud network G with a set $\mathcal{DC} = \{DC_1, DC_2, DC_3\}$ of data centers that are connected by a set $V = \{v_1, v_2, v_3\}$ of switches.

3.2. Service chains, user requests, and SLA requirements

Different users typically require different types of services. For example, some enterprise users may require a service chain consisting of a sequence of a firewall and a load balancer, while video content providers may require a sequence of a firewall, a transcoder, and a load balancer. Following existing studies [32], we consider an ordered sequence of VNFs as a *service chain*, as shown in Fig. 2.

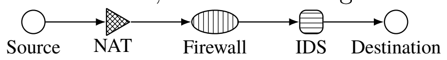


Figure 2: An example of a service chain.

Without loss of generality, we assume that the service chains of all user requests are classified into K types. There are a number of instances for each type of service chains that have been instantiated in each data center and 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 [5, 13], we assume that each instance SC_i^k of a type-k service chain in data center DC_i represents an atomic network service, and many of such atomic network services can be composed together to form a network service with a larger processing capability when necessary. 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. Notice that the number of instances of each type of service chains that can be instantiated in a data center may vary with the amount of available resource in the data center. Such numbers of instantiated service chains however do not affect the scale in/out of a specific network service; specifically, to scale out the number of instances allocated to a network service can be expanded as long as there are idle instances, while it can be reduced to scale in by releasing some instances.

Request r_j requires to route its 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. Request r_j 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. Assuming $d(r_j, DC_i)$ is the delay experienced by request r_j if its traffic is processed at data center DC_i, the end-to-end delay requirement D_j of r_j is

$$d(r_j, DC_i) = d(s_j, DC_i) + d(SC_i^k) + d(DC_i, t_j) \leq D_j, \tag{1}$$

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

3.3. Cost model

Cloud service providers provide network services on a pay-as-you-go basis [33, 34], 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. Without loss of generality, we assume that such costs are *monetary costs* that represent the amount of money for resource usages. 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 $c(r_j, DC_i)$ of implementing unit packet rate of r_j at data center DC_i is

$$c(r_j, DC_i) = \Big(c(SC_i^k) + \sum_{e \in p_{s_j,DC_i}} c(e) + \sum_{e \in p_{DC_i,t_j}} c(e)\Big), \tag{2}$$

where $p_{y,z}$ is the shortest path in G from node y to node z.

3.4. Problem definition

Given a network $G = (V \cup \mathcal{DC}, E)$, let $\mathcal{R}$ be a set of requests with each being represented by $r_j = (s_j, t_j; SC^k, \rho_j, D_j))$, and denote by $\mathcal{R}_{adm}$ the set of admitted requests delivered by an algorithm. For the sake of clarity, we use $\mathcal{R}_k$ to represent the set of requests that require instances of type-k service chain to process its traffic, and clearly $\cup_{k=1}^{K} \mathcal{R}_k = \mathcal{R}$. We define the following optimization problems.

Problem 1. *The throughput maximization problem in* G is to admit as many requests in $\mathcal{R}$ as possible, with the aim to maximize the network throughput that is defined as the total packet rates of requests that can be admitted by network G, i.e., $\sum_{r_j \in \mathcal{R}_{adm}} \rho_j$, while minimizing the accumulative implementation cost of all admitted requests, subject to the processing capacity of each type-k service chain instances in each data center, i.e., $\rho \cdot |\mathcal{SC}_i^k|$.

Problem 2. Assuming that requests in $\mathcal{R}$ arrive at the system one-by-one and the arrivals of future requests are not known in advance, *the online throughput maximization problem in* G is to admit as many requests as possible, with the objective to maximize the network throughput, while minimizing the accumulative implementation cost of all admitted requests, subject to the processing capacity of each type-k service chain instances in each data center, i.e., $|\mathcal{SC}_i^k| \cdot \rho$.

Lemma 1. *The decision version of the throughput maximization problem in a cloud network* $G = (V \cup \mathcal{DC}, E)$ *is NP-complete.*

Proof. The decision version of the throughput maximization problem is NP-Complete, by a polynomial time reduction from the partition problem [10]. 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. □

For the sake of convenience, symbols used in this paper are summarized in the following table.

Table 1: Symbols

Symbols	Meaning
$G = (V \cup \mathcal{DC}, E)$	a network with a set V of switches, a set DC of data centers, and a set E of edges
DC_i	a data center DC_i in DC
e and d_e	link $e \in E$ and the delay of implementing a unit packet along link e
K	the number of service chain types
$\mathcal{SC}_i^k$ and $\lvert\mathcal{SC}_i^k\rvert$	the set of instances of type-k service chains at data center DC_i, where $1 \leq k \leq K$ and its cardinality
SC^k	type-k service chains
SC_i^k	an instance of type-k service chains at data center DC_i
$\mathcal{R}$ and $\mathcal{R}_k$	the set of requests and the set of requests that require instances of type-k service chain
r_j, s_j, t_j	a request, its source and destination nodes in G
ρ_j and ρ $(= \rho_{min})$	packet rate of request r_j and the minimum packet rate of all requests in R
ρ_{max}	the maximum packet rate of all requests in $\mathcal{R}$
D_j	the end-to-end delay requirement of r_j
$d(r_j, DC_i)$	the delay experienced by r_j if its traffic is processed by an instance at data center DC_i
$d(s_j, DC_i)$ and $d(DC_i, t_j)$	the transfer delays of r_j from s_j to DC_i and from DC_i to t_j
$d(SC_i^k)$	the processing delay by instance SC_i^k at data center DC_i
$c(SC_i^k)$	the cost of implementing an instance of type-k service chain in data center DC_i
$c(e)$	the cost of transferring a unit packet rate for request r_j through link $e \in E$
p_{s_j, DC_i} and p_{DC_i, t_j}	the shortest paths from s_j to DC_i and from DC_i to t_j
$c(r_j, DC_i)$	the implementation cost of implementing unit packet rate of r_j at data center DC_i
x_{ij}	indicator variable indicates whether request r_j is assigned to an instance of type-k service chain in DC_i
$G' = (V', E')$	the auxiliary graph constructed in the optimal algorithm for a special case with identical packet rates
s_0 and t_0	virtual source and sink node in V' for all requests
$\langle x, y \rangle$	a directed edge in G' from node x to node y
$G'' = (V'', E'')$	the auxiliary graph constructed based on G', which is used in the approximation algorithm `Appro-Split`
γ	$= \frac{\rho_{max}}{\rho_{min}}$
$r'_{j1}, r'_{j2}, \cdots, r'_{j\gamma_j}$	virtual requests of request r_j, with each virtual request having an identical packet rate ρ $(= \rho_{min})$
L and DC_l	the number of data centers to which the virtual requests of r_j are assigned and the lth data center.
DC_{l_0}	The data center with the maximum number of virtual requests of r_j among the L data centers
ϵ	the accuracy parameter in Garg and K$\acute{}$onemman's algorithm
β_{kj}, λ_{ik}, μ_{ikj}, and θ	the dual variables for constraints (4), (5), and (6) in the **LP**.
P_i	the shadow price of each data center DC_i
I^*	the maximum resource usage of requests.
DC_{i^*}	$\arg\min_{DC_i \in \mathcal{DC}} P_i$
Δ	a constant which equals to $\frac{I^*}{\epsilon}$
$c_p(v_i)$	the processing cost at datacenter v_i
B	the budget of implementing requests

4. Integer Linear Programming

In this section we formulate the throughput maximization problem as an Integer Linear Program (ILP) when the problem size is small. This exact solution will serve as the benchmark for the online throughput maximization problem.

The objective of the throughput maximization problem is to admit as many requests in $\mathcal{R}$ as possible. Specifically, to admit a request r_j, a data center DC_i with at least one instance of the required type of service chain by the request need to be selected. Then, the traffic of r_j will be routed via the shortest path from s_j to DC_i and then from DC_i to t_j. Recall that $\mathcal{R}_k$ is the set of requests that require instances of type-k service chain to process its traffic. We thus use an indicator variable x_{ij} to indicate whether request $r_j \in \mathcal{R}_k$ is assigned to an instance of type-k service chain in DC_i to process its traffic. The objective of the throughput maximization problem is

$$\textbf{LP}: \qquad \max \sum_{DC_i \in \mathcal{DC}} \sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_k} x_{ij}\rho_j, \tag{3}$$

subject to the following constraints,

$$\sum_{i=1}^{|\mathcal{DC}|} x_{ij} \le 1, \qquad \forall r_j \in \mathcal{R}_k, 1 \le k \le K \tag{4}$$

$$\sum_{r_j \in \mathcal{R}_k} x_{ij} \cdot \rho_j \le \rho|\mathcal{SC}_i^k|, \quad \forall DC_i, 1 \le k \le K \tag{5}$$

$$x_{ij} \cdot d(r_j, DC_i) \le D_j, \quad \forall DC_i, r_j \in \mathcal{R}_k, 1 \le k \le K \tag{6}$$

$$\sum_{DC_i \in \mathcal{DC}} \sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_k} x_{ij} \cdot \rho_j \cdot c(r_j, DC_i) \le B \tag{7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall DC_i, r_j \in \mathcal{R}_k, 1 \le k \le K, \tag{8}$$

where constraints (5) indicate that the accumulative packet rates of the requests with type-k service chains should not exceed the capacity of all available instances in $\mathcal{SC}_i^k$, if the requests are assigned to DC_i for to process their traffic. Constraints (6) say that the end-to-end delay requirement D_j of each admitted request r_j should not be violated. In constraint (7), B is a pre-defined budget for the cost of implementing all requests in $\mathcal{R}$. In the proposed online algorithm (Section 6), budget B can be set to a large value, and easily tuned afterwards. Also, B is used in finding the dual of **LP** for online algorithm design. It determines the update step of the shadow price variable related to cost and represents the accuracy in updating the shadow price, and thus has an impact on the derived competitive ratio of the online algorithm. This will be elaborated in Section 6. Constraints (8) impose the integral constraint of indicator variable x_{ij}.

5. Approximation Algorithms for the Throughput Maximization Problem

In this section we devise approximations algorithms for the throughput maximization problem, since the problem is NP-Complete. To this end, we first consider a special case of the problem where all requests have identical packet rates ρ, for which we devise an optimal algorithm. Based on the proposed optimal algorithm, we then consider the throughput maximization problem where different requests may have different packet rates, by devising an approximation algorithm for it, if the traffic of each request is splittable. Otherwise, we propose another approximation algorithm by extending the proposed approximation algorithm.

5.1. Optimal algorithm for a special case with identical packet rates

Assuming that each request r_j with the minimum packet rate ρ denotes that one instance of its required type-k service chain will be used to process its traffic, since each instance is assumed to be able to process a packet rate of ρ. This means that given a set $\mathcal{R}$ of requests, a number $|\mathcal{R}|$ of instances are needed to fulfill their demands. 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.

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., $V' = V' \cup \{\mathcal{SC}_i^k \mid 1 \le k \le K,\ \textit{and}\ 1 \le i \le |\mathcal{DC}|\}$. 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. 3 illustrates an example of G'.

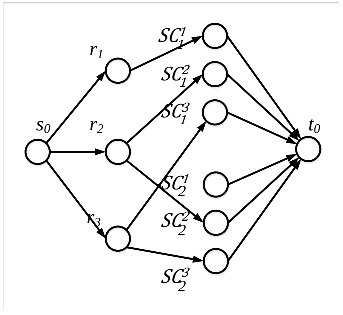


Figure 3: A constructed auxiliary graph G' based on a network with two data centers DC_1 and DC_2 and three requests to be admitted, i.e., r_1, r_2, and r_3, where requests r_1, r_2, and r_3 require instances of type 1, 2, and 3 service chains, respectively, and DC_2 is too far from the source of r_1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

Algorithm 1 An optimal algorithm for a special case of the throughput maximization problem where all requests have identical packet rates ρ

```
Input: A network $G(V \cup \mathcal{DC}, E)$, a set $\mathcal{R}$ of requests, an identical atomic packet rate $\rho$ 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 $G(V \cup \mathcal{DC}, E)$ as illustrated in Fig. 3;
2: Set edge costs and capacities for each edge in $E'$, by setting the capacity of edge $\langle r_j, \mathcal{SC}_i^k \rangle$, to 1, the cost of edge $\langle r_j, \mathcal{SC}_i^k \rangle$ to $\rho(c(SC_i^k) + \sum_{e \in p_{s_j, DC_i}} c(e) + \sum_{e \in p_{DC_i, t_j}} c(e))$, the cost of edge $\langle \mathcal{SC}_i^k, t_0 \rangle$ to 0, and the capacity of $\langle \mathcal{SC}_i^k, t_0 \rangle$ 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.
```

is given in `Algorithm` 1, which is referred to as algorithm `Optimal`.

5.2. Approximation algorithm with splittable traffic

The basic idea of the algorithm is to treat each request into a number of *virtual requests* with a minimum packet rate ρ, and then reduce 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 the previous section, with slightly different edge capacity settings.

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 ρ_{min}. We further assume that $\rho = \rho_{min}$. We 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 identical packet rate 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 treat each request r_j as a commodity with demand γ_j that need to be routed in G'' from s_0 to t_0. We then find a minimum-cost multicommodity flow f' in G'', by using the approximation algorithm due to Garg and Könemann [11]. The obtained flow f' corresponds to a splittable assignment of virtual requests of each request into data centers in network G. The details of the proposed algorithm are given in `Algorithm` 2.

5.3. 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 derived from 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 implements its service chain in one data center. Specifically, for each request r_j whose virtual requests are

Algorithm 2 Approximation algorithm `Appro-Split` for the throughput maximization problem with splittable traffic

```
Input: A network $G(V \cup \mathcal{DC}, E)$, a set $\mathcal{R}$ of requests with each request $r_j$ having a packet rate $\rho_j$, 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 $\rho$ 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 $\rho_{max}$ and $\rho_{min}$ be the maximum and minimum packet rates of all requests in $\mathcal{R}$, respectively, and assume that $\rho = \rho_{min}$;
2: Divide each request $r_j \in \mathcal{R}$ into $\gamma_j$ $(= \frac{\rho_j}{\rho})$ virtual requests with each virtual request having a packet rate of $\rho$.
3: Construct an auxiliary graph $G'' = (V'', E'')$ following the construction procedure of algorithm 1, i.e., steps 1 and 2 in algorithm 1.
4: Set the capacities for edges $\langle s_0, r_j \rangle$ and $\langle r_j, \mathcal{SC}_i^k \rangle$ to $\gamma_j$, and the capacities, costs of all other edges are the same as those in $G'$;
5: Find a minimum-cost multicommodity flow $f'$ from $s_0$ to $t_0$ in $G''$ by invoking the algorithm due to [11], by considering each $r_j$ as a commodity with demand $\gamma_j$ that needs to be routed from $s_0$ to $t_0$ in $G''$;
6: return The assigned service chain for each admitted request, and the requests that are rejected.
```

Algorithm 3 Approximation algorithm `Appro-Unsplit` for the throughput maximization problem with unsplittable traffic

```
Input: A cloud network $G(V \cup \mathcal{DC}, E)$, a set $\mathcal{R}$ of requests with each request $r_j$ having a packet rate $\rho_j$, 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 $\rho$ 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.
```

assigned to multiple data centers, we use DC_1, ..., DC_l, ..., DC_L to denote 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}. Notice that, such merging may violate the number of available instances of service chains in that data center. To avoid such violations, 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 of $|\mathcal{DC}|$, before applying `Algorithm` 2. That is, the number of instances in set $\mathcal{SC}_i^k$ is $\lfloor \frac{|\mathcal{SC}_i^k|}{|\mathcal{DC}|} \rfloor$. The proposed approximation algorithm is described in `Algorithm` 3.

5.4. Algorithm analysis

We now analyze the performance of algorithms 1, 2 and 3, in theorems 1, 2, and 3, respectively.

Theorem 1. *Given a cloud network $G(V \cup \mathcal{DC}, E)$ with a set V of switch nodes, a set $\mathcal{DC}$ of data centers that are data centers that are attached to some of the switches and there is an optical cable interconnecting the switch and the attached data center, 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. See Appendix. □

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 ratio to the minimum packet rate of all requests, i.e., ρ_{max}/ρ_{min}, is a given constant and $\rho = \rho_{min}$. There is an approximation algorithm for the throughput maximization problem,* `Algorithm` *2, which delivers an approximate solution with an approximation ratio of $(1-3\epsilon)$. The algorithm takes $O^*(K^2|\mathcal{DC}|^2|\mathcal{R}|^2 + (|V| + |\mathcal{DC}|)^2)$[5] time to deliver an approximate solution, where ϵ is the accuracy parameter in Garg and Könemman's algorithm with $0 < \epsilon \le 1/3$.*

Proof. See Appendix. □

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 to the minimum packet rate, i.e., ρ_{max}/ρ_{min}, is a given constant and $\rho = \rho_{min}$. There is an approximation algorithm for the throughput maximization problem,* `Algorithm` *3 that delivers an approximate solution with an approximation ratio of $\frac{1-3\epsilon}{|\mathcal{DC}|}$, where $|\mathcal{DC}|$ can be considered as a given constant with $|\mathcal{DC}| \ll |V|$.*

Proof. See Appendix. □

6. Online Algorithm for the Online Throughput Maximization Problem

We now consider the online throughput maximization problem where requests arrive in one-by-one without the knowledge of future arrivals. We propose an online algorithm with a competitive ratio for it.

6.1. Overview

The proposed online algorithm is based on the idea of primal-dual update. The rationale of this approach is to maintain *shadow price* variables for data centers in $\mathcal{DC}$, which parsimoniously abstract the state of resource usages in data centers [4, 15]. On the arrival of a request, the algorithm compares the price of the cheapest data center in terms of the shadow price to a given threshold that will be defined later, and the request is then either rejected or admitted. The shadow price variables are then updated accordingly. This procedure is triggered on the arrival of each request. Let β_{kj}, λ_{ik}, μ_{ikj}, and θ be the dual variables for constraints (4), (5), (6), and (7) in **LP** in Section 4, respectively.

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

6.2. Online algorithm

Considering that the future arrivals of requests are not known in advance, myopic admission of current requests may harm the admissions of future requests. We thus need an admission policy to regulate the admissions of current requests and a metric on how much resources should be reserved for future requests. To this end, we define the shadow price P_i of each data center DC_i, the maximum resource usage I^* of requests, and a constant Δ to adjust resource reservations for future requests. First, the shadow price P_i of each data center DC_i represents the marginal increase of strengthening the capacity and budget constraints (5) and (7) of the **LP** if request r_j is admitted by DC_i. It thus can be defined as

$$P_i = \Big(\lambda_{ik} + \theta \cdot c(r_j, DC_i)\Big). \tag{9}$$

Second, we give the definition of I^* to capture the maximum of the strengthenings of constraints (5) and (7) of the **LP** over all data centers for the requests with unit packet rate, i.e., the maximum of computing resource usage and costs of implementing requests in data centers. This constant regulates how much the resource and delay constraints in the **LP** can be approached in the worst case, when admitting a request with unit packet rate.

$$I^* = \max\Big\{1, \max_{DC_i \in \mathcal{DC}_j, r_j \in \mathcal{R}_k} \Big(\frac{d(r_j, DC_i)}{\rho_j}, c(r_j, DC_i)\Big)\Big\} \tag{10}$$

Third, to define the metric on how much resources should be reserved for future requests, we use a constant Δ that is a function of the defined constant I^* and a given parameter ϵ with $0 < \epsilon \le 1$, where

$$\Delta = \frac{I^*}{\epsilon}. \tag{11}$$

Parameter ϵ is also used to provide a tradeoff between the competitiveness of the proposed online algorithm and the degree of violating the constraints in **LP**. Namely, a larger ϵ will lead to less network throughput while a lower degree of violations on the resource capacities of G and the delay requirement of requests.

Given the definitions of P_i, I^*, and Δ, we now describe the admission policy of the algorithm. Specifically, for request $r_j \in \mathcal{R}_k$, we find the data center with the minimum shadow price P_{i^*}, that is $DC_{i^*} \leftarrow \arg\min_{DC_i \in \mathcal{DC}_j} P_i$, where $\mathcal{DC}_j$ is the set of data centers that can meet the delay requirement of r_j. If

$$P_{i^*} \ge 1 - \frac{(d(r_j, DC_i))^2}{\Delta \cdot D_j \cdot \rho_j}, \tag{12}$$

request r_j will be rejected; otherwise, it will be accepted. The rationale behind this threshold setting of P_{i^*} is to keep the dual feasibility of the dual variables. Notice that the term $\frac{(d(r_j, DC_i))^2}{\Delta \cdot D_j \cdot \rho_j}$ in the right hand side of inequality (12) is to ensure only data centers that can meet the delay requirement of r_j to be considered.

Algorithm 4 An online algorithm for the online throughput maximization problem.

```
Input: A cloud network $G(V \cup \mathcal{DC}, E)$, and requests that arrive at the network one by one.
Output: Admit or reject each arrived request, and an assignment of admitted requests to instances of service chains in the
    data centers in $\mathcal{DC}$.
1: for each arrival of request $r_j$ do
2:     Let $\mathcal{DC}_j$ be the set of data centers that can met the delay requirement of $r_j$;
3:     $DC_{i^*} \leftarrow \arg\min_{DC_i \in \mathcal{DC}_j} P_i$;
4:     if $P_{i^*} \geq 1 - \frac{(d(r_j, DC_{i^*}))^2}{\Delta \cdot D_j \cdot \rho_j}$ then
5:         Reject request $r_j$;
6:     else
7:         Assign request $r_j$ to an instance of type-$k$ instances at data center $DC_{i^*}$;
8:         Update dual variables $\beta_{kj}$, $\lambda_{i^*k}$, $\mu_{i^*kj}$, and $\theta$, following rules in (13), (14), (15), and (16);
```

Having defined the admission policy, we then describe how to regulate the resource preservation for future requests, by giving the rules of updating dual variables. Each defined dual variable is set to zero initially. If a request $r_j \in \mathcal{R}_k$ is assigned to data center $DC_{i^*} \leftarrow \arg\min_{DC_i \in \mathcal{DC}_j} P_i$, the values of dual variables β_{kj}, λ_{i^*k}, μ_{i^*kj}, and θ are updated according to the following rules

$$\beta_{kj} \leftarrow \rho_j\big(1 - P_{i^*}\big) \tag{13}$$

$$\lambda_{i^*k} \leftarrow \lambda_{i^*k}\Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}_{i^*}^k|}\Big) + \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_{i^*}^k|} \tag{14}$$

$$\mu_{i^*kj} \leftarrow \frac{d(r_j, DC_{i^*})}{\Delta \cdot D_j} \tag{15}$$

$$\theta \leftarrow \theta\Big(1 + \frac{\rho_j \cdot c(r_j, DC_{i^*})}{B}\Big) + \frac{\rho_j \cdot c(r_j, DC_{i^*})}{\Delta \cdot B}. \tag{16}$$

The details of the proposed algorithm are given in Algorithm 4, which is referred to as algorithm `Online`.

6.3. Algorithm analysis

In the following we analyze the competitive ratio of the proposed online algorithm as follows.

We first give the dual of the **LP** in Section 4, i.e., its objective is to

$$\min \sum_{DC_i \in \mathcal{DC}} \sum_{k=1}^{K} \rho \cdot \lambda_{ik} |\mathcal{SC}_i^k| + \sum_{DC_i \in \mathcal{DC}} \sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_k} \mu_{ikj} \cdot D_j + \theta \cdot B + \sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_k} \beta_{kj}.$$

subject to

$$\sum_{DC_i \in \mathcal{DC}} \sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_k} \Big(\lambda_{ik} \cdot \rho_j \cdot |\mathcal{SC}_i^k| + \mu_{ikj} \cdot d(r_j, DC_i) + \theta \cdot \rho_j \cdot c(r_j, DC_i)\Big) + \sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_j} \beta_{kj} \geq \rho_j. \tag{17}$$

For the sake of simplicity, we re-write inequality (17) as

$$\sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_j} \beta_{kj} \geq \rho_j - \sum_{DC_i \in \mathcal{DC}} \sum_{k=1}^{K} \sum_{r_j \in \mathcal{R}_k} \Big(\lambda_{ik} \cdot \rho_j \cdot |\mathcal{SC}_i^k| + \mu_{ikj} \cdot d(r_j, DC_i) + \theta \cdot \rho_j \cdot c(r_j, DC_i)\Big) \tag{18}$$

Given the dual of the **LP**, we show the dual feasibility of the defined updating rules defined in inequalities (13), (14), (15), and (16), in the following lemma.

Lemma 2. *The updating rules in (13), (14), (15), and (16) preserve the dual feasibility of the dual variables.*

Proof. See Appendix. □

We then show the upper bound on the increase of the dual objective in Lemma 3, if a request is admitted.

Lemma 3. *Whenever a request r_j is admitted, the increase of the objective function value of the dual program is no more than $(1+\epsilon)\rho_j$.*

Proof. See Appendix. □

We finally show how much the computing capacity, delay requirement, and budget constraint in the **LP** can be violated. Let $L_{ik}(j)$ and $\lambda_{ik}(j)$ be the total packet rate of requests that are admitted by type-k instances in data center DC_i and the value of dual variable λ_{ik}, after the admission of request r_j. We have

Lemma 4. *The admission policy of* `Algorithm` *4 can make the end-to-end delay requirement of each admitted request met, and the violation of the computing capacity and budget constraints in the* **LP** *is at most by a multiplicative $O(\log N + \log(1/\epsilon))$, where $N = \max_{i,k,j}\{|\mathcal{SC}_i^k|, D_j, B\}$.*

Proof. See Appendix. □

Theorem 4. *Given a network $G(V \cup \mathcal{DC}, E)$ with requests arriving in the network one-by-one without the knowledge of future arrivals, 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 that have been instantiated in data center DC_i. There is an online algorithm for the online throughput maximization problem,* `Algorithm` *4, which delivers a solution with a competitive ratio of $(1-3\epsilon, O(\log N + \log(1/\epsilon)))$, where the throughput achieved by it is at least $(1-3\epsilon)$ times of the optimal one while the violation of constraints (5), (6), and (7) is no more than a multiplicative factor $O(\log N + \log(1/\epsilon))$, where $N = \max_{i,k,j}\{|\mathcal{SC}_i^k|, D_j, B\}$, $0 < \epsilon \leq 1/3$.*

Proof. See Appendix. □

Remarks: The budget B, the number $|\mathcal{SC}_i^k|$ of instances of type-k service chain in data center DC_i , and the delay requirement D_j of r_j determine the violation of constraints (5), (6), and (7) . For example, if $N = \max_{i,k,j}\{|\mathcal{SC}_i^k|, D_j, B\} = B$, budget B has an impact on the violation ratio, which means a larger budget will incur a higher violation of resource constraints because more requests may be admitted. This gives a conservative of setting the value of B. Specifically, given the number of available instances for each type of service chain, the budget can be set as the total cost of using all the instances and bandwidth resources along the longest path in the network G. Due to such conservative way of setting budget, the resource capacities can be violated. Such resource violations however can be avoided by scaling down the number of available instances of each type of service chain by the ratio of $O(\log N + \log(1/\epsilon))$, and then invoke `Algorithm` 4.

7. Simulations

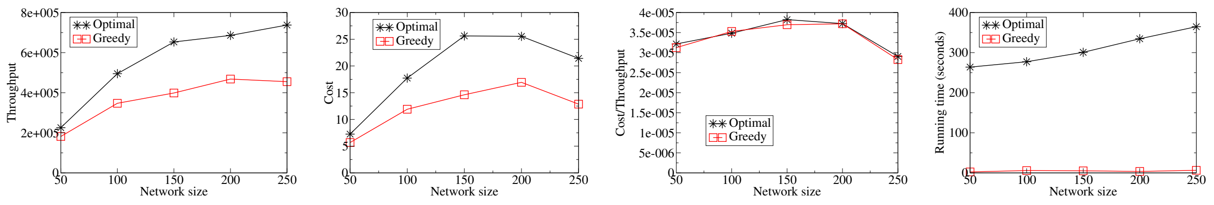
In this section, we evaluate the performance of the proposed algorithms through experimental simulations.

7.1. Experiment Settings

We consider networks that are generated by the tool GT-ITM [12] and two real network topologies including an European network GÉANT [14] and an ISP network AS1755 [30] in our simulations. There are nine data centers for the GÉANT topology as set in [14] and the number of data centers in ISP networks are provided in [26]. The transmission delay of a link varies between 2 milliseconds (ms) and 5 ms [19]. 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). Each service chain instance has at most five network functions. The processing delay of a packet for each network function is randomly drawn from 0.045 ms to 0.3 ms [23], and the processing delay of a service chain instance is the sum of processing delays of its network functions. The number of service chain types K is 5. The number of instances 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 [23]. 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 [20], the delay requirement varies from 10 ms to 200 ms [25], and its type of service chain is randomly assigned from one of the five types. Parameter ϵ in the online algorithm is set to 0.2. 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. Table 2 summarizes the above-mentioned settings of main parameters used in the experiments.

Table 2: Parameter Settings

Parameters	Range	Distributions	References
Transmission delay of a link	$[2, 5]$ ms	Uniform	[16, 27, 35]
D_j	[10, 200] ms	Uniform and Zipf	[16, 25]
The costs of transmitting 1 GB	[\$0.05, \$0.12]	Uniform	[2]
The costs of processing 1 GB	[\$0.15, \$0.22]	Uniform	[2]
The processing delay of a packet	[0.045 to 0.3] ms	Uniform	[23]
The number of service chain types K	5	-	[23, 20, 35]
The number of instances of each VNF type	$[10, 50]$	-	-
ρ	400 packets/second	Uniform	[23]
ρ_j	[400, 4,000] packets/second	Uniform	[20]



(a) Throughput (b) Cost (c) Cost/Throughput (d) Running time (seconds)

Figure 4: The performance of algorithms `Optimal` and `Greedy` [35].

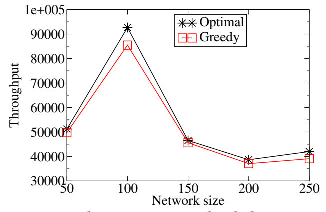
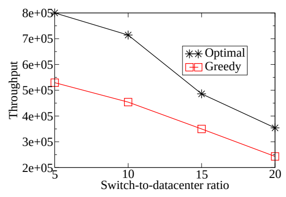


Figure 5: The performance of algorithms `Optimal` and `Greedy` with delay requirements of requests following Zipf distribution.

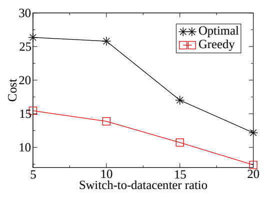
We compare the proposed approximation algorithms with a greedy algorithm that aims to maximize the throughput by always admitting requests with the smallest packet rates first, by implementing the request in a data center that not only meets its delay requirement but also has the maximum number of available service chain instances. For simplicity, we refer to this greedy algorithm as algorithm `Greedy`. The proposed algorithms 1, 2, and 3 as algorithms `Optimal`, `Appro-Split`, and `Appro-Unsplit`, respectively. We also evaluate the performance of the proposed online algorithm, i.e., `Online`, with an online greedy algorithm that always assigns an arrived request to a data center that has the maximum ratio of its available computing resource to the demand of computing resource of the request. We refer to this greedy algorithm for the online throughput maximization problem as algorithm `Online-Greedy`. In addition, since the exact solution in Section 4 is not scalable for large network sizes, we use its relaxed version that considers each variable x_{ij} as a real value in the range of $[0, 1]$. The obtained solution is an upper bound on the optimal throughput of the exact solution. We refer to this upper bound as `OPT-UB`.

7.2. Performance evaluation of the optimal algorithm with identical packet rates

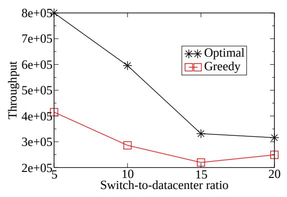
We first study the performance of algorithms `Optimal` and `Greedy` by varying the number of switches $|V|$ from 50 to 250, while fixing the *switch-to-datacenter* ratio $\frac{|V|}{|\mathcal{DC}|}$ at 10. Fig. 4 shows the results, and from Fig. 4 (a) and (b) we can see that algorithm `Optimal` achieves a throughput at least 30% more than that by algorithm `Greedy`, while `Optimal` has a higher implementation cost for 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 for the requests. Furthermore, as shown in Fig. 4 (a), the throughput of all algorithms increases when the network size grows from 50 to 250, because a larger network means more data centers with more available service chain instances. From Fig. 4 (c), we can see that algorithms `Optimal` and `Greedy` have similar performance on the cost of



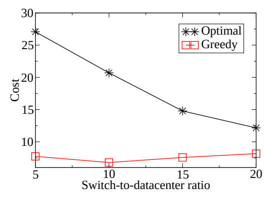
(a) Throughputs of algorithms `Optimal` and `Greedy` in network GÉANT



(b) Costs of algorithms `Optimal` and `Greedy` network GÉANT



(c) Throughputs of algorithms `Optimal` and `Greedy` in network AS1755



(d) Costs of algorithms `Optimal` and `Greedy` network AS1755

Figure 6: The performance of algorithms `Optimal` and `Greedy` in networks GÉANT and AS1755.

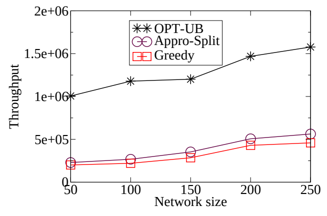
implementing one unit of packet rate, as both algorithms aim to maximize throughput. In addition, it can be seen from Fig. 4 (d) that algorithm `Optimal` takes more time than that by algorithm `Greedy` to deliver a solution.

Fig. 5 shows the performance of algorithms `Optimal` and `Greedy` when the delay requirements of requests follow Zipf distribution, with most requests have low delay requirements that account for only a small portion of the range $[10, 200]$ ms. From Fig. 5, it can be seen that with the increase of network sizes, the throughputs of algorithms increase first from 50 to 100 and then drops afterwards. The reason is that most requests have low delay requirements and each request with higher probabilities of being implemented in a longer path with more links when the network size keeps growing, thereby increasing the probability of being rejected due to the violation of its delay requirement.

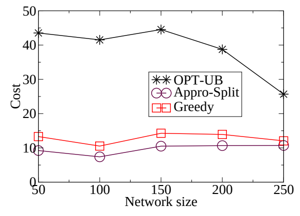
We then evaluate the performance of algorithm `Optimal` against that of algorithm `Greedy` by varying the switch-to-datacenter ratio from 4 to 10 in networks GÉANT and AS1755, respectively. It can be seen from Fig. 6 (a) and Fig. 6 (c) that algorithm `Optimal` consistently achieves much more throughput than that by algorithm `Greedy` in networks GÉANT and AS1755, when the switch-to-datacenter ratio is 10, although it delivers higher costs (to admit more requests). Also, with the increase of $\frac{|V|}{|\mathcal{DC}|}$, the algorithms admit less requests, since larger value of $\frac{|V|}{|\mathcal{DC}|}$ means smaller number of data centers and less available number of service chain instances, if the size $|V|$ of a network is fixed.

7.3. Performance evaluation of the approximation algorithm with different packet rates

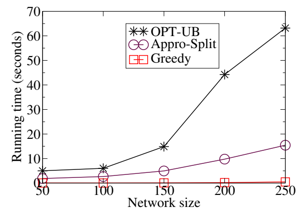
We first study the performance of algorithms `Appro-Split`, `Greedy`, and `OPT-UB` by varying the network size from 50 to 250 while fixing the *switch-to-datacenter* ratio at 10. We can see from Fig. 7 (a) that the throughput achieved by algorithm `Appro-Split` is better than that by algorithm `Greedy`, whereas the cost of algorithm `Appro-Split` is far less than algorithm `Greedy`. Also, algorithm `OPT-UB` has the highest throughput among the three algorithms. For example, when the network size is 100, algorithm `Appro-Split` admits around $30,000$ higher throughput than that by algorithm `Greedy`. Fig. 8 (a) depicts the results of algorithms `Appro-Split`, `Greedy`, and `OPT-UP` when delay requirements of requests following Zipf distributions, from which it can be seen that the throughputs of the algorithms increases first when the network size increases from 50 to 150, and decreases afterwards. Similar performance of algorithms `Appro-Split`, `Greedy`, and `OPT-UB` in networks GÉANT and AS1755 can be seen in Fig. 9.



(a) Throughputs

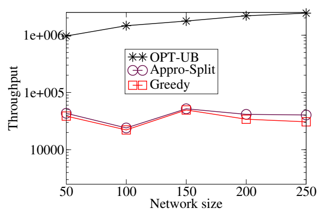


(b) Costs

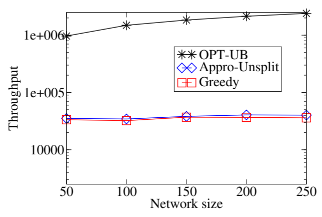


(c) Running times (seconds)

Figure 7: The performance of algorithms `Appro-Split`, `Greedy`, and `OPT-UB` [35].

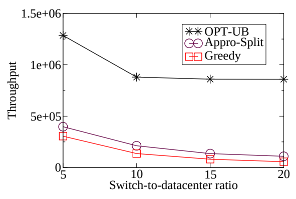


(a) Throughputs of algorithms `Appro-Split`, `Greedy`, and `OPT-UB`

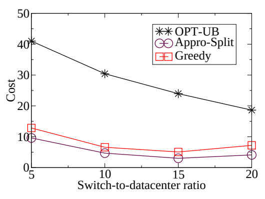


(b) Throughputs of algorithms `Appro-Unsplit`, `Greedy`, and `OPT-UB`

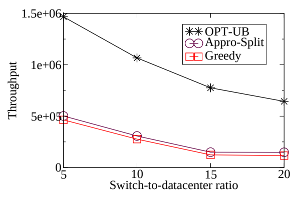
Figure 8: The performance of algorithms `Appro-Split`, `Appro-Unsplit`, `Greedy`, and `OPT-UB` with delay requirements of requests following Zipf distribution.



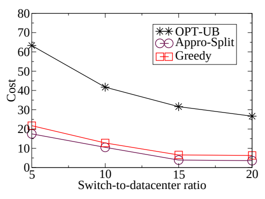
(a) Throughputs of algorithms `Appro-Split` and `Greedy` in network GÉANT



(b) Costs of algorithms `Appro-Split` and `Greedy` network GÉANT

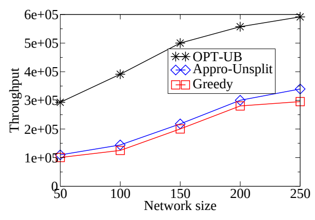


(c) Throughputs of algorithms `Appro-Split` and `Greedy` in network AS1755

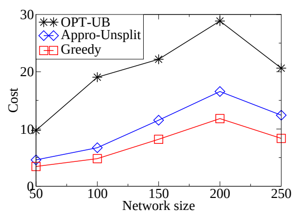


(d) Costs of algorithms `Appro-Split` and `Greedy` network AS1755

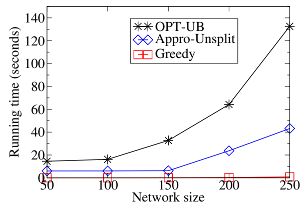
Figure 9: The performance of algorithms `Appro-Split` and `Greedy` in networks GÉANT and AS1755.



(a) Throughputs



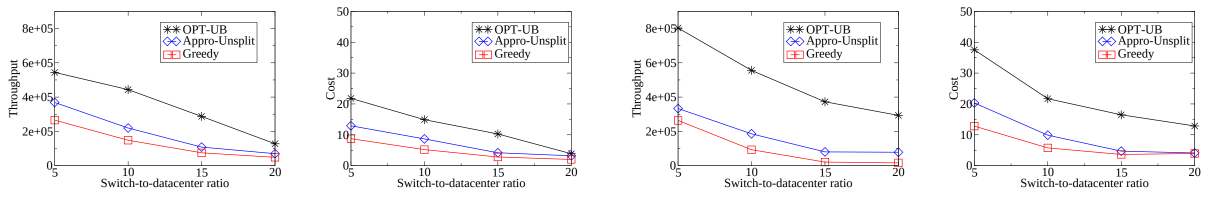
(b) Costs



(c) Running times (seconds)

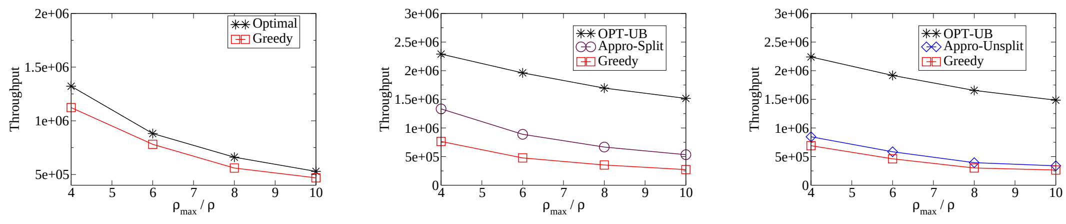
Figure 10: The performance of algorithms `Appro-Unsplit` and `Greedy`.

We then compare the performance of algorithm `Appro-Unsplit` with these of algorithms `Greedy` and `OPT-UB` by varying network size from 50 to 250 while fixing the *switch-to-datacenter* ratio at 10. It can be seen from Fig. 10 that algorithm `Appro-Unsplit` achieves a 6% more throughput than that by algorithm `Greedy`, when the network size is 250. Furthermore, by comparing the performance of algorithm `Appro-Split` and algorithm `Appro-Unsplit` in Fig. 7 and Fig. 10, it can be seen that algorithm `Appro-Split` has a higher



(a) Throughputs of algorithms `Appro-Unsplit`, `Greedy`, and `OPT-UB` in network GÉANT (b) Costs of algorithms `Appro-Unsplit`, `Greedy`, and `OPT-UB` network GÉANT (c) Throughputs of algorithms `Appro-Unsplit`, `Greedy`, and `OPT-UB` in network AS1755 (d) Costs of algorithms `Appro-Unsplit`, `Greedy`, and `OPT-UB` network AS1755

Figure 11: The performance of algorithms `Appro-Unsplit`, `Greedy` and `OPT-UB` in networks GÉANT and AS1755.



(a) Throughputs of algorithms `Optimal` and `Greedy`. (b) Throughputs of of algorithms `Appro-Split`, `Greedy`, and `OPT-UB`. (c) Throughputs of of algorithms `Appro-Unsplit`, `Greedy`, and `OPT-UB`.

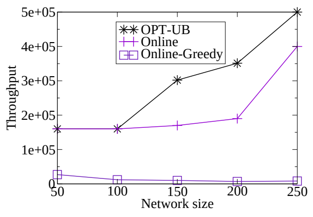
Figure 12: The performance of algorithms `Appro-Unsplit`, `Appro-Split`, `Greedy`, and `OPT-UB`.

throughput than algorithm `Appro-Unsplit`. Similar results for such performance can also be found in networks GÉANT and AS1755, as shown in Fig. 11. In addition, Fig. 8 (b) shows the results if delay requirements of requests follow Zipf distributions.

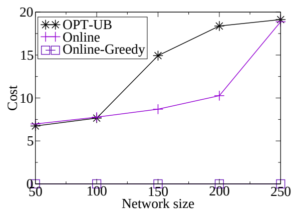
We finally study the impact of the minimum packet rate ρ on the performance of algorithms `Appro-Split` and `Appro-Unsplit`, by varying the ratio $\frac{\rho_{max}}{\rho}$ from 4 to 10 while fixing ρ_{max} at 4,000 and the network size at 100. We can see from Fig. 12 that algorithms `Optimal`, `Appro-Split`, and `Appro-Unsplit` perform much better than algorithm `Greedy` for different values of $\frac{\rho_{max}}{\rho}$. Also, the throughput of algorithms `Optimal`, `Appro-Split`, and `Appro-Unsplit` decrease with the increase the value of $\frac{\rho_{max}}{\rho}$. The reason is that a higher $\frac{\rho_{max}}{\rho}$ means a lower ρ, making each instance of service chains being able to process a lower packet rate. This reduces the total packet rate that can be processed by the data centers, which reduces the number of requests that can be admitted.

7.4. Performance evaluation of the online algorithm

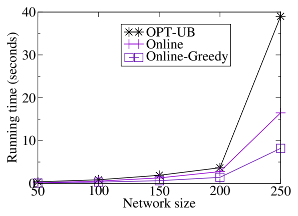
We here study the performance of algorithm `Online` against that of algorithms `Online-Greedy` and `OPT-UB`, by varying network size from 50 to 250 while fixing the number of requests at 2,000. The results are shown in Fig. 13, and we can see from Fig. 13 (a) that algorithm `Online` delivers a near-optimal throughput which is around 90% of the optimal one by algorithm `OPT-UB`. Also, algorithm `Online` outperforms algorithm `Online-Greedy` and delivers a solution that is at least 10 times better than that of algorithm `Online-Greedy`, while it has a higher cost to implement the admitted request. It must be mentioned that algorithm `OPT-UB`



(a) Throughputs of algorithms `Online`, `Online-Greedy`, and `OPT-UB`.



(b) Costs of of algorithms `Online`, `Online-Greedy`, and `OPT-UB`.



(c) Running times (seconds) of algorithms `Online`, `Online-Greedy`, and `OPT-UB`.

Figure 13: The performance of algorithms `Online`, `Online-Greedy`, and `OPT-UB`.

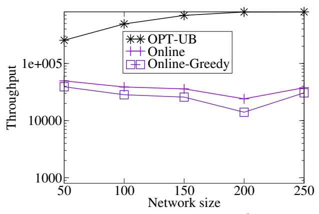
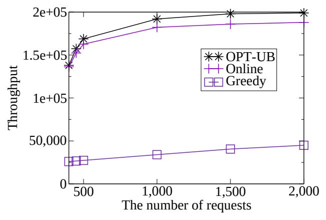


Figure 14: The performance of algorithms `Online`, `Online-Greedy`, and `OPT-UB` with delay requirements of requests following Zipf distribution.

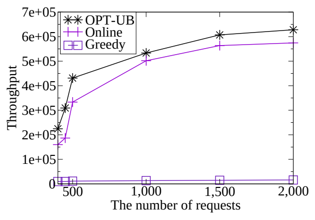
has the longest running time whereas algorithm `Online-Greedy` has the shortest running time. Similar performance of algorithms `Online`, `Online-Greedy` and `OPT-UB` in networks GÉANT and AS1755 can be seen in Fig. 15. In addition, Fig. 14 shows the results if the delay requirements of requests follow Zipf distributions. It can be seen that the throughput of the algorithms decrease with the growth of network sizes, since requests have higher probability of being assigned to longer paths in the network thereby violating their delay requirements.

8. Conclusion

In this paper we investigated the throughput maximization problem in a cloud network, in which limited numbers of instances of each type of service chains are instantiated in data centers. We proposed an optimal algorithm for a special case of the problem when all requests have identical packet rates. Otherwise, we devised two approximation algorithms with provable approximation ratios for the problem, depending on whether the traffic of each request is splittable. If arrivals of future requests are not known in advance, we studied the online throughput maximization problem by proposing an online algorithm with a competitive ratio. Simulation results demonstrated that the performance of the proposed algorithms are promising. It must be mentioned that the developed algorithms have wide applications for network operators to optimize the performance of their networks. For example, the algorithms for throughput maximization algorithms can be used for one-shot optimizations periodically, or in each time slot. Specifically, the time can be divided into equal time slots, and requests are available in the beginning of each time slot. Such time slots can be set in the length of several minutes, depending on the network environment and request characteristics. The algorithms can be directly adopted by deciding the admissions of each request once it arrives into the system.



(a) Throughputs of algorithms `Online`, `Online-Greedy`, and `OPT-UB` in network GÉANT.



(b) Throughputs of algorithms `Online`, `Online-Greedy`, and `OPT-UB` in network AS1755.

Figure 15: The performance of algorithms `Online`, `Online-Greedy`, and `OPT-UB` in networks GÉANT and AS1755.

Acknowledgement

This work was partially funded by the EU 5GEx (671636), NECOS projects (777067), and the fundamental research funds for the central universities (DUT17RC(3)061).

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] J. W. Anderson *et al.* xOMB: extensible open middleboxes with commodity servers. *Proc. of ANCS*, ACM/IEEE, 2012.

[4] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trend in Theoretical Computer Science*, Vol. 3, Nos. 2-3, pp. 93-263, Now Publishers Inc., 2007.

[5] 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.

[6] 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.

[7] J. Fan, C. Guan, Y. Zhao, and C. Qiao. Availability-aware mapping of service function chains. *Proc. of INFOCOM'17*, IEEE, 2017.

[8] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. *Proc. of NSDI '14*, USENIX, 2014.

[9] H. Feng, J. Llorca, A. M. Tulino, D. Raz, A. F. Molisch. Approximation algorithms for the NFV service distribution problem. *Proc. of INFOCOM'17*, IEEE, 2017.

[10] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-Completeness.* W.H. Freeman, 1997.

[11] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *Proc. of FOCS'98*, IEEE, 1998.

[12] GT-ITM. `http://www.cc.gatech.edu/projects/gtitm/`.

[13] 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.

[14] A. Gushchin, A. Walid, and A. Tang. Scalable Routing in SDN-enabled Networks with Consolidated Middleboxes. *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization* (HotMiddlebox), 2015.

[15] L. Guo, J. Pang, and A. Walid. Dynamic service function chaining in SDN-enabled networks with middleboxes. *Proc. of ICNP'16*, IEEE, 2016.

[16] http://www.huawei.com/minisite/5g/en/defining-5g.html, 2018.

[17] 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.

[18] IHS. *http://www.infonetics.com/pr/2015/1H15-Service-Provider-Capex.asp.*

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

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

[21] Y. Ma, W. Liang, and Z. Xu. Online revenue maximization in NFV-enabled SDNs. To appear in Proc of IEEE ICC'18, May, 2018.

[22] 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.

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

[24] 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.

[25] Microsoft. Plan network requirements for Skype for business. `https://technet.microsoft.com/en-us/library/gg425841.aspx`, 2015.

[26] Z. A. Qazi *et al.*SIMPLE-fying middlebox policy enforcement using SDN. *Proc. SIGCOMM '13*, ACM, 2013.

[27] 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.

[28] V. Sekar *et al.* Design and implementation of a consolidated middlebox architecture. *Proc. of NSDI*, USENIX, 2012.

[29] V. Shrivastava *et al.*. Application-aware virtual machine migration in data centers. *Proc. of INFOCOM'11*, IEEE, 2011.

[30] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. *Proc. of SIGCOMM*, ACM, 2002.

[31] P. Vizarreta *et al.*. QoS-driven function placement reducing expenditures in NFV deployments. *Proc. of ICC'17*, IEEE, 2017.

[32] P. Wang *et al.*. Dynamic function composition for network service chain: Model and optimization. *Computer Networks*, Vol. 92, pp. 408–418, Elsevier, 2015.

[33] 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.

[34] 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.

[35] Z. Xu, W. Liang, A. Galis, and Y. Ma. Throughput maximization and resource optimization in NFV-enabled networks. *Proc. of ICC'17*, IEEE, 2017.

[36] R. Yu, G. Xue, and X. Zhang. QoS-aware and reliable traffic steering for service function chaining in mobile networks. *IEEE Journal on Selected Areas in Communications*, Vol. 35, No.11, pp.2522 – 2531, IEEE, 2017.

Appendix

Proof of Theorem 1

Proof. We first show that the algorithm delivers a feasible solution. This is to show a minimum cost maximum flow 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. We now show the delay requirement of each admitted request is met. Since a data center is attached to its switch via an optical cable, the delay incurred between the data center and its attach switch can be ignored. Also, considering that there will not be an edge from a request node r_j to a switch that is attached with data center DC_i that cannot meet its delay requirement in G', the delay requirement of an admitted request will be met.

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. □

Proof of Theorem 2

Proof. 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 [11].

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 [11], 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 $O^*(K^2|\mathcal{DC}|^2|\mathcal{R}|^2 + (|V| + |\mathcal{DC}|)^2)$. □

Proof of Theorem 3

Proof. 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 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'^k_l be the implementation cost of the virtual requests assigned to service chain SC^k_l at data center DC_l. Clearly, $\sum_{l=1}^{L} c'^k_l > c^k_{l_0}$. 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^k_l will depend on how many virtual requests are moved from DC_l to DC_{l_0}. Notice that both the cost of computing resource consumption and the communication cost of a unit packet rate of the moved virtual requests will be the same as the ones in DC_{l_0}. This is because the moved virtual requests will be processed by the same type of service chain and their traffic will be forwarded via the same paths as the ones in in 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, $c^k_l \leq c^k_{l_0}$. This means that the implementation cost c_j of r_j at DC_{l_0}, can be maximally $L \cdot c^k_{l_0} < |\mathcal{DC}| \cdot c^k_{l_0}$, i.e., $c_j < |\mathcal{DC}| \cdot c^k_{l_0}$. We then have

$$\begin{aligned} c' &= \sum_{r_j \in \mathcal{R}} \sum_{l=1}^{L} c'^k_l \geq \sum_{r_j \in \mathcal{R}} c^k_{l_0}, \quad \text{since} \sum_{l=1}^{\gamma_j} c'^k_l > c^k_{l_0}, \\ &> \sum_{r_j \in \mathcal{R}} \frac{c_j}{|\mathcal{DC}|}, \quad \text{since } c_j < |DC| \cdot c^k_{l_0}, \quad \geq \frac{c}{|\mathcal{DC}|}. \end{aligned} \tag{19}$$

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. □

Proof of Lemma 2

Proof. To show the dual feasibility, we need to show that the dual variables are always positive, and constraint (18) is always met. Clearly, for the values of variables λ_{i^*k}, μ_{i^*kj}, and θ, the updating rules always keep them to be nonnegative, as they are set to zero initially and their values can only increase in each update. The value of variable β_{kj} is positive due to the admission policy in inequality (12). Thus, the dual variables are positive. For constraint (18), we consider two cases: (1) request r_j is rejected; and (2) r_j is admitted.

Case (1): if request r_j is rejected, it is clear that the right hand side of inequality (18) is nonpositive. Considering β_{kj} is set to zero initially, constraint (18) holds.

Case (2): if request r_j is admitted, we have

$$P_{i^*} < 1 - \frac{(d(r_j, DC_i))^2}{\Delta \cdot D_j \cdot \rho_j}. \tag{20}$$

Due to Eq. (13), we then have

$$\begin{aligned}
&\beta_{kj} = \rho_j \cdot (1 - P_{i^*}) \\
&\geq \rho_j \cdot (1 - P_{i^*} - \frac{(d(r_j, DC_i))^2}{\Delta \cdot D_j \cdot \rho_j}), \\
&\qquad\qquad \text{since } 0 < \frac{(d(r_j, DC_i))^2}{\Delta \cdot D_j \cdot \rho_j} < 1 - P_{i^*} \\
&= \rho_j \cdot (1 - \lambda_{ik} - \theta \cdot c(r_j, DC_i) - \frac{\mu_{i^*kj} \cdot d(r_j, DC_i)}{\rho_j}).,
\end{aligned} \tag{21}$$

where the derivation of inequality (21) is due to the definition of P_i and the updating rule for μ_{i^*kj} (i.e., equality (15)). Therefore, constraint (18) holds in this case. For each of other data centers in $\mathcal{DC} \setminus \{DC_{i^*}\}$, constraint (18) also holds, since dual variables are only updated if a request is assigned to data center DC_i. Also, further updates for future requests only make the right hand side of (18) smaller, thereby preserving its feasibility. □

Proof of Lemma 3

Proof. Following the rules in (13), (14), (15), if a request r_j is admitted, the objective function value of the dual program increases by

$$\begin{aligned}
\Gamma_{r_j} \quad &= \Big(\lambda_{ik}\frac{\rho_j}{\rho \cdot |\mathcal{SC}_i^k|} + \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_i^k|}\Big)\rho|\mathcal{SC}_i^k| + \Big(\frac{d(r_j, DC_i)}{\Delta \cdot D_j}\Big)D_j \\
&+ \Big(\theta\frac{\rho_j \cdot c(r_j, DC_i)}{B} + \frac{\rho_j \cdot c(r_j, DC_i)}{\Delta \cdot B}\Big)B + \rho_j\big(1 - P_i\big) \\
&= \Big(\lambda_{ik}\rho_j + \frac{\rho_j}{\Delta}\Big) + \frac{d(r_j, DC_i)}{\Delta} + \Big(\theta \cdot \rho_j \cdot c(r_j, DC_i) + \frac{\rho_j \cdot c(r_j, DC_i)}{\Delta}\Big) \\
&+ \rho_j - \rho_j \cdot \lambda_{ik} - \rho_j \cdot \theta \cdot c(r_j, DC_i) \\
&= \rho_j + \frac{\rho_j}{\Delta}\Big(1 + \frac{d(r_j, DC_i)}{\rho_j} + c(r_j, DC_i)\Big) \leq \rho_j + \frac{\rho_j \cdot 3I^*}{\Delta} = \rho_j(1 + 3\epsilon), \text{since } \Delta = \frac{I^*}{\epsilon}.
\end{aligned} \tag{22}$$

□

Proof of Lemma 4

Proof. As shown in Step 2 of `Algorithm` 4, each admitted request r_j will only be assigned to one of the data centers that can met its delay requirement.

The upper bound on the violation of computing capacity as well as the budget constraint is analyzed as follows.

Initially, we have $\lambda_{ik}(j) = L_{ik}(j) = 0$ for all data centers $DC_i \in \mathcal{DC}$ and all k with $1 \leq k \leq K$. Notice that only when a request r_j is admitted, the values of $\lambda_{ik}(j)$ and $L_{ik}(j)$ will be updated,

$$L_{ik}(j) = L_{ik}(j-1) + \rho_j, \tag{23}$$

and

$$\begin{aligned}
&\lambda_{ik}(j) = \lambda_{ik}(j-1)\Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}_i^k|}\Big) + \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_i^k|} \\
&= \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_i^k|} + \Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}_i^k|}\Big) \cdot \frac{1}{\Delta}\Big(\frac{\rho_{j-1}}{\rho \cdot |\mathcal{SC}_i^k|} \\
&+ (1 + \frac{\rho_{j-1}}{\rho \cdot |\mathcal{SC}_i^k|})\frac{\rho_{j-2}}{\rho \cdot |\mathcal{SC}_i^k|} \\
&+ (1 + \frac{\rho_{j-2}}{\rho \cdot |\mathcal{SC}_i^k|})(1 + \frac{\rho_{j-1}}{\rho \cdot |\mathcal{SC}_i^k|})\frac{\rho_{j-3}}{\rho \cdot |\mathcal{SC}_i^k|} + \dots \\
&+ (1 + \frac{\rho_1}{\rho \cdot |\mathcal{SC}_i^k|}) \cdot \dots \cdot (1 + \frac{\rho_{j-1}}{\rho \cdot |\mathcal{SC}_i^k|})\frac{\rho_{j-3}}{\rho \cdot |\mathcal{SC}_i^k|}\Big) \\
&\geq \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_i^k|} + \Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}_i^k|}\Big) \cdot \frac{1}{\Delta}\Big(\\
&(\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})\frac{\rho_{j-2}}{\rho \cdot |\mathcal{SC}_i^k|} + (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})^2\frac{\rho_{j-3}}{\rho \cdot |\mathcal{SC}_i^k|} \\
&\dots + (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})^{j-1}\frac{\rho_1}{\rho \cdot |\mathcal{SC}_i^k|}\Big), \quad \text{since } \frac{\rho \cdot |\mathcal{SC}_i^k| + \rho_{j-1}}{\rho \cdot |\mathcal{SC}_i^k|} > \frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|} \qquad (24) \\
&\geq \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_i^k|} + \Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}_i^k|}\Big) \cdot \frac{1}{\Delta}\Big(1 - 1 + (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})\frac{1}{|\mathcal{SC}_i^k|} \\
&+ (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})^2\frac{1}{|\mathcal{SC}_i^k|}, \text{since } \rho_{min} = \rho \\
&\dots + (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})^{j-1}\frac{1}{|\mathcal{SC}_i^k|}\Big) \\
&\geq \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_i^k|} + \Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}_i^k|}\Big) \cdot \frac{1}{\Delta \cdot |\mathcal{SC}_i^k|}\Big(1 - 1 + (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|}) \\
&+ (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})^2 + \dots + (\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|})^{j-1}\Big) \qquad (25) \\
&\geq \frac{\rho_j}{\Delta \cdot \rho \cdot |\mathcal{SC}_i^k|} + \Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}_i^k|}\Big) \cdot \frac{1}{\Delta \cdot |\mathcal{SC}_i^k|}\Big(exp\Big(\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|}\Big) - 1\Big) \qquad (26) \\
&= \frac{1}{\Delta \cdot |\mathcal{SC}_i^k|}\Big(exp\Big(\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|}\Big) - 1\Big)\Big(1 + \frac{\rho_j}{\rho \cdot |\mathcal{SC}|_i^k}\Big) + \frac{\rho_j}{\rho}\Big) \\
&\approx \frac{1}{\Delta|\mathcal{SC}_i^k|}\Big(exp\Big(\frac{L_{ik}(j-1)}{\rho \cdot |\mathcal{SC}_i^k|}\Big)exp\Big(1 + \frac{\rho_j}{\rho|\mathcal{SC}_i^k|}\Big) - 1\Big) \geq \frac{1}{\Delta N}\Big(exp\Big(\frac{L_{ik}(j)}{\rho \cdot |\mathcal{SC}_i^k|}\Big) - 1\Big), \qquad (27)
\end{aligned}$$

where the derivation from inequality (25) to inequality (26) is due to the fact that $1 + x + x^2 + x^3 + \cdots > 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = exp(x)$. Notice that we used the first order approximation $exp(x) = 1 + x$. Let

$$1/Z_* = \min\Big\{1, \min_{DC_i \in \mathcal{DC}, r_j \in \mathcal{R}_k} \Big(\frac{d(r_j, DC_i)}{\rho_j}, c(r_j, DC_i)\Big)\Big\}.$$

If $\lambda_{ik} > Z_*$, then for any data center DC_i, we have

$$\lambda_{ik} + \frac{\mu_{ikj} d(r_j, DC_i)}{\rho_j} + \theta \cdot c(r_j, DC_i) > 1, \tag{28}$$

which means DC_i will not be selected for request r_j. Since `Algorithm` 4 admits requests one by one, we have $\lambda_{ik} < Z_*$ in the last update of λ_{ik} for the processing of request r_{j-1}. Since one more update increases λ_{ik} at most I^*, we have

$$\lambda_{ik} \leq I^* + Z_*. \tag{29}$$

This implies

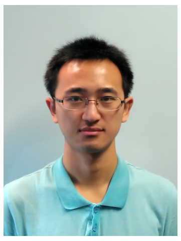
$$\frac{L_{ik}(j)}{\rho_j |\mathcal{SC}_i^k|} \leq \log((I^* + Z_*)N\Delta + 1) = O(\log N + \log(1/\epsilon)). \tag{30}$$

In other words, the violation of the capacity constraint of data center DC_i is at most by $O(\log N + \log(1/\epsilon))$. Similar results can be obtained for the budget constraint. □

Proof of Theorem 4

Proof. According to Lemma 4, each constraint of the problem can be violated at most by a multiplicative $O(\log N + \log(1/\epsilon))$.

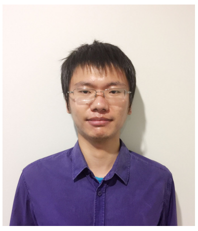
The rest is to show the competitive ratio of $(1-3\epsilon)$. By Lemma 3, the objective value of the dual program increases by at most $(1 + 3\epsilon)\rho_j$ if request r_j is admitted. As a result, the overall objective of the **LP** is at least $1/(1 + 3\epsilon) \geq 1 - 3\epsilon$ times of the objective of the dual program, when $0 < \epsilon \leq 1/3$. Therefore, the throughput obtained by the solution due to the fact that `Algorithm` 4 is at least $(1 - 3\epsilon)OPT$. □



Zichuan Xu (M'17) received his PhD degree from the Australian National University in 2016, ME degree and BSc degree from Dalian University of Technology in China in 2011 and 2008, all in Computer Science. He was a Research Associate at Department of Electronic and Electrical Engineering, University College London, UK. He currently is an Associate Professor at School of Software, Dalian University of Technology, China. His research interests include cloud computing, software-defined networking, network function virtualization, wireless sensor networks, routing protocol design for wireless networks, algorithmic game theory, and optimization problems.

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

Alex Galis is a Professor in Networked and Service Systems at University College London. He has co-authored 10 research books and more that 250 publications in the Future Internet areas: system management, networks and services, networking clouds, 5G virtualisation and programmability. He was a member of the Steering Group of the Future Internet Assembly (FIA) and he led the Management and Service-aware Networking Architecture (MANA) working group. He acted as TPC chair of 14 IEEE conferences. He is also a co-editor of the IEEE Communications Magazine feature topic on Advances In Networking Software. He acted as a Vice Chair of the ITU-T SG13 Group on Future Networking. He is involved in IETF and ITU-T SG13 network slicing activities and he is also involved in IEEE SDN initiative.



Yu Ma received BSc degree in Computer Science at the Australian National University in 2014. He currently is a PhD candidate in Computer Science at the Australian National University. His research interests include software defined networking, Internet of Things (IoT), and Wireless sensor networks.

Qiufen Xia received her PhD degree from the Australian National University in 2017, ME degree and BSc degree from Dalian University of Technology in China in 2012 and 2009, all in Computer Science. She currently is a Lecturer at International School of Information Science & Engineering, Dalian University of Technology, China. Her research interests include big data processing, mobile cloud computing, distributed clouds, cloud computing, and optimization problems.

Wenzheng Xu (M'15) received the BSc, ME, and PhD degrees in computer science from Sun Yat-Sen University, Guangzhou, P.R. China, in 2008, 2010, and 2015, respectively. He currently is a Special Associate Professor at the Sichuan University and was a visitor at the Australian National University. His research interests include wireless ad hoc and sensor networks, mobile computing, approximation algorithms, combinatorial optimization, online social networks, and graph theory. He is a member of the IEEE.