Less is More: Service Profit Maximization in Geo-Distributed Clouds

Zhenjie Yang, Yong Cui, Xin Wang, Minming Li, and Yadong Liu

Abstract—Nowadays cloud providers purchase a good deal of bandwidth from Internet service providers to satisfy the growing requests from corporate customers for the exclusive use of inter-datacenter bandwidth. For exclusive bandwidth services, neither maximizing the revenue nor minimizing the cost can bring the maximal profit to cloud providers. The diversity of bandwidth prices and the random arrival time of user requests further increase the difficulty in economically scheduling the services to meet user requests from cloud providers. In this article, we propose to help cloud providers maximize their service profits by properly selecting user requests to serve rather than satisfying them all. We formulate the problem of service profit maximization and prove its NP-hardness. To handle offline request submission, we propose a solution that maximizes the service profit by alternately maximizing the service revenue and minimizing the service cost. To maximize service profit under online request submission, we propose an online scheduling algorithm that carefully handles the risk of not being able to pay off the incremental service cost and makes scheduling decisions in real time. Our extensive evaluations demonstrate that our solutions can achieve more than 1.6x the service profits of existing solutions.

Index Terms—Geo-distributed cloud, service profit, maximization, randomized rounding, simulated annealing

1 Introduction

In the past decade, cloud computing has achieved great success and transformed a large part of the IT industry [1]. As many companies move their services to clouds, to improve the efficiency of cloud computing and reduce operating costs in data centers, cloud providers such as Amazon, Microsoft and Alibaba take many approaches, including computational sprinting [2], adaptive green hosting [3], instance blending [4] and effective capacity modulation [5]. They have made large profits from the cloud computing market. With the proliferation of globally-distributed services and quick growth of user requests across data centers, cloud providers turn to geo-distributed clouds [6], [7].

Cloud providers build data centers in different geographical areas and use dedicated inter-datacenter wide-area networks (Inter-DC WANs) to connect them [6], [7], [8], [9], [10]. Except for some giant companies that build dedicated lines between their data centers, most cloud providers spend hundreds of millions of dollars per year to lease the bandwidth from Internet service providers (ISPs) to connect the data centers [6]. Typically ISPs sell bandwidth at a fixed price per *unit* (e.g., 10 Gbps or 80 Gbps) [6], [11], [12], with the price varied with links and regions [13]. The bandwidth usage is calculated over a fixed *billing cycle*, such as a month or a year [12]. As the growth of wide-area network bandwidth has been decelerating for many years and the volume of inter-datacenter traffic increases quickly [14], [15], wide-area network bandwidth becomes more and more expensive, and it is critical for cloud providers to use the bandwidth more economically to make higher profit [16].

On the cloud computing market, corporate customers usually request cloud providers to reserve a certain amount of exclusive bandwidth between given data centers over a specific time period, and negotiate bandwidth prices privately with the cloud providers. To satisfy the user requests for exclusive bandwidth, cloud providers purchase bandwidth from ISPs at high cost. It is important and challenging for a cloud provider to maximize its service profit, the net benefit with the bandwidth cost subtracted from the revenue generated by serving user requests.

Intensive efforts have been made to schedule user requests between data centers economically [12], [16], [17], [18], [19], [20], [21]. In order to make full use of the bandwidth purchased from ISPs and increase the cost efficiency of bandwidth expenditure, many solutions adopt store-and-forward approaches or transfer user data in multiple paths at the cost of extra storage or packet-level reordering. As another type of solutions, inter-datacenter transfers are scheduled with different pricing methods (e.g., dynamic pricing) to maximize the service revenue, which requires the cloud providers to modify the current pricing mechanism. As the existing solutions will either introduce additional costs (or performance losses) or require major changes to the current mechanisms, they are not good options to apply in practical geo-distributed clouds.

In the current service mode, cloud providers are obliged to satisfy user requests for exclusive bandwidth based on negotiated prices. If the expensive wide-area network bandwidth

- *Zhenjie Yang, Yong Cui, and Yadong Liu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: {yangzj15, liuyd17}@mails.tsinghua.edu.cn, cuiyong@tsinghua.edu.cn.*
- *Xin Wang is with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11790 USA. E-mail: x.wang@stonybrook.edu.*
- *Minming Li is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China, and also with Shenzhen Research Institute, City University of Hong Kong, Hong Kong, China. E-mail: minming.li@cityu.edu.hk.*

Manuscript received 26 March 2020; revised 22 August 2020; accepted 9 September 2020. Date of publication 18 September 2020; date of current version 6 September 2022.
(Corresponding author: Yong Cui.)
Recommended for acceptance by X. Wang.
Digital Object Identifier no. 10.1109/TCC.2020.3024616

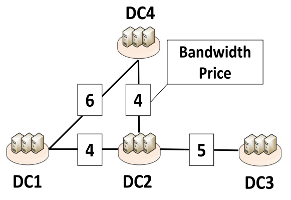
cannot be efficiently used, the profit of cloud providers will be compromised. To simplify the negotiation process between both sides and help cloud providers maximize service profit, we consider a cloud operational model where users independently submit their requests and bids, and the cloud provider just serves the ones that can maximize its service profit. For the declined ones, users may update and resubmit their requests to the cloud provider.

Maximizing service profit by declining part of user requests requires cloud providers to address two major challenges: (1) *Request acceptance.* Although accepting all the requests of customers may not maximize the service profit, it is still hard for cloud providers to find the right set of requests to turn down. If there are N $(N \geq 0)$ requests in a billing cycle, there will be 2^N available combinations of user requests for cloud providers to choose from. As there is no way for cloud providers to describe the service profit changes over different combinations, it would be very time-consuming to evaluate these combinations one by one. (2) *Request scheduling.* In a typical inter-datacenter network, there are several routing paths between two data centers, and they have different unit-bandwidth prices. For a set of user requests, adopting different routing solutions will lead to different bandwidth costs and greatly affect the service profit. Properly scheduling these requests is essential but it is difficult for cloud providers to find the right schedule and paths for the maximum service profit. If users submit their requests whenever they want in a billing cycle, cloud providers will face a more serious situation to make good scheduling decisions when the "future" requests are unknown.

The above challenges are intertwined with each other, and online scheduling increases the difficulty for cloud providers to make good decisions. We propose different solutions to handle the offline submission scheme with user requests submitted only at the beginning of a billing cycle and the online submission scheme that requests can be submitted at any time. When users submit their requests at the beginning of a billing cycle, we take an easy-to-control mechanism *Metis* to optimize the service profit. We alternately maximize the service revenue under given bandwidth and minimize the bandwidth cost under given requests, and cloud providers could dynamically adjust the bandwidth to purchase and the requests to accept. In the case that users submit requests in an online manner, we propose an algorithm *OSA* to evaluate the impact of scheduling the requests in different ways and make decisions in real time. The main contributions of our work are summarized as follows.

First, we formally define the problem of **S**ervice **P**rofit **M**aximization (SPM) in geo-distributed clouds, where we take into account the variation of bandwidth prices in inter-datacenter networks and the differences of delivering a user request on different paths or declining it. SPM is proved to be NP-hard. By solving it properly, cloud providers are expected to maximize their service profits.

Second, we propose an offline solution, Metis, to solve SPM in polynomial time without incurring too high loss. Instead of solving SPM directly, Metis obtains high service profits by alternately solving two subproblems of SPM. To further improve the performance of Metis, we propose approximation algorithms to solve the subproblems of SPM and also prove the approximation ratios.



(a) Inter-DC WAN

Req.	s	d	ts	td	r	v
R1	1	2	0	1	1	3
R2	1	3	1	2	1	6
R3	1	4	1	2	1	4
R4	2	4	0	1	1	3

(b) Requests

Solution	Accepted Requests	Routing Path				Cloud Providers'		
		R1	R2	R3	R4	Revenue	Cost	Profit
S1	R1, R2, R3, R4	1-2	1-2-3	1-2-4	2-4	**16**	**17**	**-1**
S2	R1, R3, R4	1-2	-	1-4	2-4	**10**	**14**	**-4**
S3		1-2	-	1-2-4	2-4	**10**	**8**	**2**

(c) Representative solutions

Fig. 1. Comparison of different solutions.

Third, to handle user requests in real time, we propose a simulated annealing-based online scheduling algorithm, OSA, as a complement to Metis. In order to maximize the service profits of cloud providers, OSA evaluates the performance of different schedules by our proposed metric and make decisions immediately. Extensive evaluations demonstrate that our solutions can achieve higher than 1.6x the service profit of the existing solutions.

The remainder of this paper is organized as follows: we formulate SPM in Section 2. We present the design of Metis in Section 3. We propose two algorithms to solve the subproblems of SPM in Sections 4 and 5. In Section 6, we propose an online algorithm to handle user requests in real time. We show the evaluation results in Section 7 and present deployment discussion in Section 8. Finally, we introduce the related work in Section 9, and conclude our work in Section 10.

2 Service Profit Maximization

In this section, we first give a simple example to show the advantage of declining some user requests for cloud providers to make more profit, then we present the system model and notations used in this paper. Next, we formulate the problem of service profit maximization and prove its NP-hardness.

2.1 Motivation

Given the diversity of user requests and the pricing method used by ISPs to charge the bandwidth, satisfying all user requests may prevent cloud providers from obtaining the maximum service profit. Rather than accepting *all* of the customer requests, *declining* part of the requests may help cloud providers make more profit. We give an example to show how cloud providers can make more service profit by declining some user requests and adopting proper scheduling solution. In Fig. 1a, the Inter-DC WAN connects four data centers (DC1, DC2, DC3 and DC4) with four bi-directional links, whose bandwidth price (i.e., the cost of per unit bandwidth) is different with each other. The number on each link represents its bandwidth price. In Fig. 1b, four requests (R) in a billing cycle from time 0 to 2 are associated with different sources (s), destinations (d), start times (ts), end times (td), required bandwidths (r) and values (v). For each request R, the customer expects the cloud provider to reserve r units of bandwidth between data center s and data center d during the period $[ts, td)$. She is willing to pay v units of money as cloud provider's return. In Fig. 1c, we present 3 representative solutions. In the solution $S1$, the cloud provider accepts all four requests and gets a total service revenue of 16. Fig. 1c shows the bandwidth the cloud provider reserves for users on different routing paths, and the total bandwidth usages of DC1-DC2, DC2-DC3 and DC2-DC4 are 2, 1 and 1, respectively. The total cost is 17, i.e., $2*4+1*5+1*4$. By scheduling user requests with $S1$, cloud providers can obtain a service profit of -1. Rather than accepting all requests as $S1$, the solutions $S2$ and $S3$ decline the request $R2$ and schedule the remaining requests with different routing solutions, and get the service profits of -4 and 2, respectively. Accepting all requests will use more bandwidth and result in higher bandwidth cost at the same time. Comparing $S1$ with $S3$, declining request $R2$ helps cloud providers get more service profit by scheduling the accepted requests ($R1$, $R3$, and $R4$) properly to efficiently use the purchased bandwidth. In $S2$ and $S3$, cloud providers decline $R2$ and adopt different routing solutions to schedule the accepted requests. The difference between the service profit is up to 6 units of money, which shows the significance of request scheduling on maximizing the service profit. From this example, we can see that, a good request admission and routing solution is critical for cloud providers to make the maximum service profit.

2.2 Model

In cloud market, a cloud provider controls an Inter-DC network, which is denoted as $G(V, E)$. Each edge e in G represents a directed link from a data center to another. We use V and E to denote the sets of data centers and edges in the network G. Typically, ISPs sell bandwidth by unit, for example, 10 Gbps per unit. They also charge bandwidth in a fixed billing cycle [12], which consists of T independent time slots. For simplicity, we use t to index a time slot. With u_e denoting the cost of per unit of bandwidth on edge e and c_e the number of bandwidth units to be charged, the bandwidth cost of e is the product of the two [18], [22]. In a billing cycle, users submit a set of K requests in total. A request i ($1 \leq i \leq K$) is denoted by a six-tuple $\{s_i, d_i, ts_i, td_i, r_i, v_i\}$, where the transmission is carried from the source data center s_i to the destination data center d_i between the time ts_i and td_i. The bandwidth required is r_i, and the value is v_i. For each pair of data centers (s_i, d_i), there are multiple available paths in G. We use $P_i = \{P_{i,j} : 1 \leq j \leq L_i\}$ to denote the set of paths for request i, where L_i denotes the number of paths. With the above definitions, we define service revenue and service cost as follows.

Service Revenue. For a request i, we use a binary variable $x_{i,j}$ to indicate whether it takes the path $P_{i,j}$. Specifically, $x_{i,j}$ is 1 if it takes the jth path from its source (s_i) to its destination (d_i). Otherwise, $x_{i,j} = 0$. If its requirement is satisfied, i.e., $\sum_{j=1}^{L_i} x_{i,j} = 1$, the cloud provider receives a service revenue of v_i, and the revenue is 0 otherwise. The service revenue $\mathcal{I}$ can be represented as

$$\mathcal{I} = \sum_{i=1}^{K} v_i \left(\sum_{j=1}^{L_i} x_{i,j} \right).$$

TABLE 1
Summary of Notations

Notation	Definition
G	(V, E): the topology of Inter-DC WAN
K	the number of requests in a billing cycle
T	the number of time slots of a billing cycle
i	$\{s_i, d_i, ts_i, td_i, r_i, v_i\}$: the i-th request; s_i: source; d_i: destination; ts_i: start time; td_i: end time; r_i: required rate; v_i: value;
L_i	the number of directed path from s_i to t_i
$P_{i,j}$	$\{i, j \mid 1 \leq i \leq K, 1 \leq j \leq L_i\}$: the j-th directed path from s_i to t_i
P_i	$\{P_{i,j} : 1 \leq j \leq L_i\}$: the set of directed paths from s_i to t_i
u_e	bandwidth price of e, i.e., the cost of per unit of bandwidth on e
c_e	the charging bandwidth of e
$x_{i,j}$	$\{0, 1\}$: whether request i flows through $P_{i,j}$
$I_{i,j,e}$	$\{0, 1\}$: whether edge e is in the path $P_{i,j}$

Service Cost. In this work, we focus on the cost of inter-datacenter networks and take the bandwidth cost of inter-datacenter links as service cost, and

$$\mathcal{C} = \sum_{e \in E} u_e c_e.$$

For clarity, we summarize the notations we use in Table 1.

2.3 Formulation

For any pair of data centers in the network G, there is usually more than one routing path. As splitting a user request into multiple sub-requests and reserving bandwidth for one user on many routing paths will inevitably result in reordering problem and affect the interest of users, in this work, we solve the problem with user requests unsplittable. For request i, the cloud provider should select at most one path to reserve bandwidth. Thus the following inequality should be satisfied:

$$\forall i : \sum_{j=1}^{L_i} x_{i,j} \leq 1. \tag{1}$$

Our goal is to find the match of requests and routing paths, which we denote as "schedule". A schedule is said to satisfy the request i if $\sum_{j=1}^{L_i} x_{i,j} = 1$ holds.

A schedule is said to satisfy the link capacities if the link capacity constraint is satisfied for every edge e at any time slot t. First, we describe the required bandwidth r_i of request i as

$$\forall i, \forall t \quad : \quad r_{i,t} = \begin{cases} r_i & , \quad t \in [ts_i, td_i] \\ 0 & , \quad \text{otherwise} \end{cases}.$$

Based on this, we present the link capacity constraint as

$$\forall t, \forall e : \sum_{i=1}^{K} \sum_{j=1}^{L_i} r_{i,t} x_{i,j} I_{i,j,e} \leq c_e, \tag{2}$$

where $I_{i,j,e}$ is a predefined binary number and it indicates whether edge e is on path $P_{i,j}$.

To maximize service profit for a cloud provider, our objective is to maximize the value of service revenue minus service cost in Inter-DC WANs. We call it **S**ervice **P**rofit **M**aximization, and formulate it as follows:

$$\max\ (\mathcal{I} - \mathcal{C}),$$

s.t.

$$\forall e : c_e \in \mathbb{N} \tag{3}$$

$$\forall i, \forall j : x_{i,j} \in \{0,1\} \text{Constraints} \quad (1)(2). \tag{4}$$

By reducing the subset sum problem [23] (denoted as SUBSET-SUM) to SPM, we have the following theorem:

Theorem 1. *SPM is NP-hard.*

Proof. SPM is NP-hard, as it contains SUBSET-SUM as a special case, which is NP-complete. SUBSET-SUM is defined as follows: Given a set of integers $\mathcal{S}$, is there a non-empty subset whose sum is a given integer $\mathcal{N}$?

We first construct an instance $\mathcal{A}$ of SUBSET-SUM. We consider setting the integer set $\mathcal{S} = \{a_1, a_2, \ldots, a_n\}$ and $\sum_{i=1}^{n} a_i = \mathcal{M}$. The goal of SUBSET-SUM is to find a subset of sum $\mathcal{N}$ in $\mathcal{S}$. Without loss of generality, let $\mathcal{N} < \mathcal{M} < 2\mathcal{N}$. Next, we construct a special case $\mathcal{A}'$ of SPM. Suppose there is one edge e in the network G and one time slot in a scheduling period, and there are n requests. The value of i is linearly related to the bandwidth it requires. For each request i, it requires $r_i\ (= a_i/\mathcal{N})$ units of bandwidth and we set the value v_i of it as r_i. Then we have that the sum of v_i is greater than 1 but less than 2. We consider setting the bandwidth price of edge e as σ where σ is less than 1 but it is very close to 1. By far we have transformed $\mathcal{A}$ to $\mathcal{A}'$ in polynomial time.

For the instance $\mathcal{A}'$, the cloud provider would obtain the maximal service profit $1 - \sigma$ by satisfying a subset of sum 1 in the set of $\{v_i\}$. If we could solve SPM with a polynomial-time algorithm, we would obtain the optimal solution of SUBSET-SUM, which means that SUBSET-SUM could be solved in polynomial time. Therefore, SPM is at least as hard as SUBSET-SUM. As SUBSET-SUM is NP-hard, SPM is NP-hard, too. This completes the proof. □

Depending on request submission scheme taken by users, offline or online, cloud providers need to schedule the service based on the whole or partial request information to maximize the service profit. In the following sections, we will introduce the solutions for offline and online submission respectively.

3 Handling User Requests in Batch

In the previous section, we have proved that SPM is NP-hard. Since the service revenue and the service cost are closely-coupled, directly solving SPM is complex and difficult. The subproblems of SPM are similar to the classical flow problems [24], [25]. Thus we first decompose SPM into two subproblems, and then propose a framework to obtain a good schedule for SPM by alternately solving the two subproblems.

3.1 Decomposition of SPM

We decompose the original problem into two subproblems, which are defined as follows:

• *Request-Limited SPM.* Given a set of accepted requests, solving SPM is equivalent to minimizing the service cost. We call it **R**equest-**L**imited SPM (RL-SPM), and formulate it as follows:

$$\min \sum_{e \in E} u_e c_e,$$

s.t.

$$\forall i : \sum_{j=1}^{L_i} x_{i,j} = 1 \qquad \text{Constraints} \quad (2)(3)(4). \tag{5}$$

Like the minimum-cost flow problem [24], the goal of RL-SPM is to find the cheapest possible way of reserving bandwidth for a group of requests accepted in G.

• *Bandwidth-Limited SPM.* Suppose the link capacity in the network is fixed, i.e., c_e is given for each $e \in E$, service profit maximization can be transformed to maximizing service revenue, which is called **B**andwidth-**L**imited SPM (BL-SPM) and formulated as follows:

$$\max \sum_{i=1}^{K} v_i \left(\sum_{j=1}^{L_i} x_{i,j} \right),$$

s.t.

$$\text{Constraints} \quad (1)(2)(4).$$

Like the unsplittable flow problem [25], the goal of BL-SPM is to find the subset of requests that can maximize the revenue, so that the entire demand for each such request can be satisfied on its path while respecting the given link capacity. We prove BL-SPM is NP-hard in Section 5.

In the following, we design a framework, Metis, to handle user requests in batch by alternately solving these two subproblems, RL-SPM and BL-SPM.

3.2 Design of Metis

Despite that the two subproblems of SPM are relatively easier to solve, solving either of the two is not sufficient for solving SPM. Considering the conflicts between RL-SPM and BL-SPM, where the given condition of one problem (i.e., the requests accepted in RL-SPM and the bandwidth limitation in BL-SPM) is the objective of the other (i.e., minimizing the bandwidth cost and maximizing the service revenue). By dynamically setting the accepted requests and the bandwidth limitation as required by the two subproblems, alternately solving them can potentially generate a good solution for the original problem. Following this strategy, we propose a framework, *Metis*, for cloud providers to specify the alternate mode, as shown in Fig. 2.

There are six modules in Metis: Input collects the necessary information for solving SPM, such as the requests, the network topology, the bandwidth prices and so on. Output will output the final decisions, i.e., acceptance decision and scheduling decision, based on the calculation of SP Updater.

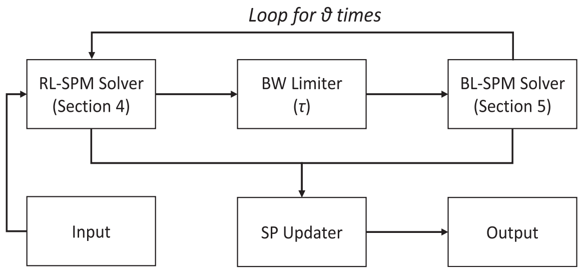


Fig. 2. Overview of metis.

SP Updater works with other three modules: RL-SPM Solver, BW Limiter and BL-SPM Solver. Before running them, SP Updater initializes the service profit as zero, where the cloud provider accepts no request and uses no bandwidth. Both service revenue and service cost are zero. We introduce the functions and operations of these modules in the following:

• *RL-SPM Solver.* Given a set of accepted requests, it aims to present a scheduling solution that needs as little bandwidth as possible. We run it to output the bandwidth requirements and the scheduling of requests. If it can obtain more service profit than the recorded schedule by SP Updater, SP Updater deletes the original schedule and records the current schedule. In the initialization phase, we set all the user requests as "accepted".

• *BW Limiter.* Based on the link bandwidth output by RL-SPM Solver, BW Limiter is used to reset link bandwidth according to the rule (τ) predetermined by the cloud provider. As a low link utilization means a high bandwidth cost, BW Limiter reduces the bandwidth set by RL-SPM Solver for the link whose average utilization is the minimum. The BL-SPM Solver is then further called to continue the process until the maximum service revenue is achieved.

• *BL-SPM Solver.* With the given bandwidth, BL-SPM Solver can only satisfy part of (not all) user requests. Its goal is to achieve the maximal service revenue without violating link capacity constraints. As the given bandwidth can not satisfy all user requests, BL-SPM Solver declines some requests in each loop and update the set of user requests accepted accordingly.

• *SP Updater.* It is used to record the optimal schedule, including scheduling decisions and bandwidth usage. It checks whether RL-SPM Solver or BL-SPM Solver outputs a better schedule and decides whether to record it as the (tentatively) optimal one.

In a scheduling period, the above modules will run for θ ($\theta \geq 1$) times before Metis outputs the final schedule. Through Metis, we alternately reduce the cost and improve the revenue to optimize the final service profit. Cloud providers can set τ and θ based on their actual needs (e.g., low computing time) and historical data [10], [26]. As BL-SPM Solver declines user requests in each loop, the number of accepted user requests decreases gradually. Metis loops at most K (the number of requests in a billing cycle) times before finishing scheduling all requests.

In Metis, RL-SPM Solver and BL-SPM Solver can greatly affect the final service profit. In the following sections, we propose two algorithms to solve RL-SPM and BL-SPM with guaranteed approximation ratios.

4 Algorithm for RL-SPM Solver

In Metis, RL-SPM Solver is applied to select routing paths for the accepted requests and minimize the total bandwidth cost. However, there are two challenges derived from integer constraints: any request is unsplittable for the interest of users [27] and the bandwidth charging is calculated by integer unit [12]. The constraints make it hard to solve the problem directly.

To address the challenges, we propose to decompose RL-SPM into two subproblems: a subproblem $\mathcal{P}_1$ with the relaxing of the integer requirement of bandwidth, and a subproblem $\mathcal{P}_2$ with the relaxing of the integral routing constraint of request. $\mathcal{P}_1$ becomes the well-known multicommodity unsplittable flow problem [23]. Then we design a multistage approximation algorithm (*MAA*) to independently solve $\mathcal{P}_1$ and $\mathcal{P}_2$ based on the relax-and-round method, as follows:

• *Relaxation.* We first solve the relaxed linear program of $\mathcal{P}_1$ by allowing $x_{i,j}$ to be a fractional number, with $x_{i,j} \in [0,1]$. An advanced LP solver can yield the optimal fractional results, denoted as $\{\hat{x}_{i,j}\}$ and $\{\hat{c}_e\}$.

• *Randomized Rounding.* With the randomized rounding scheme, a request i is scheduled to path $P_{i,j}$ with a probability $\hat{x}_{i,j}$. For each request i, the overall path selection factors over all paths should meet $\sum_{j=1}^{L_i} \hat{x}_{i,j} = 1$. We select exactly one path for each request i with a probability $\hat{x}_{i,j}$ and update $\{\hat{x}_{i,j}\}$ and $\{\hat{c}_e\}$ accordingly. Let n denote the number of nodes in the network G. According to [28], [29], it achieves a $O(\frac{\log n}{\log \log n})$-approximation solution for $\mathcal{P}_1$ with high probability.[1]

• *Ceiling.* As the start times and end times of requests are different, for cloud providers, the required bandwidth varies with time slots and edges. Considering that bandwidth is charged in integer units, for each edge e, we round up $\hat{c}_e$ as the bandwidth for charging and denote it as $\lceil \hat{c}_e \rceil$, and calculate the total bandwidth cost. We denote the set of edges whose fractional charging bandwidth $\hat{c}_e$ is positive as E' ($E' \subseteq E$).

In the case that any fractional bandwidth $\hat{c}_e$ is greater than a positive number α, i.e., $\alpha = \min_{e \in E'}\{\hat{c}_e\}$, we have a theorem about the approximation algorithm for $\mathcal{P}_2$. Before presenting it, we denote the ρ-approximation algorithm of a minimization problem as the one that can achieve a solution within ρ ($\rho \geq 1$) times the optimal solution. Similar to the definition of ρ-approximation algorithm, the ρ-relaxed algorithm for an integer programming minimization problem is the one that can achieve a solution within ρ times the optimal solution of the relaxed original problem. Based on our definitions, we have the following theorem:

Theorem 2. *There exists a* $(\frac{\alpha+1}{\alpha})$*-relaxed algorithm for* $\mathcal{P}_2$*, where* $\alpha = \min_{e \in E'}\{\hat{c}_e\}$.

Proof. With the paths selected for requests in randomized rounding procedure, we denote the minimum service cost of the relaxed $\mathcal{P}_2$ as $\dot{C}_2$ and the output objective of

1. Consider a certain probabilistic algorithm on a graph with n nodes. If the probability that the algorithm returns the correct answer is $1 - \frac{1}{n}$, then when the number of nodes is very large, the algorithm is correct with a probability that is very close to 1. This fact is expressed shortly by saying that the algorithm is correct with high probability.

ceiling procedure as $\hat{\mathcal{C}}_2$. For each edge $e \in E'$, the bandwidth for charging is denoted by $\hat{c}_e$ and $\lceil \hat{c}_e \rceil$ respectively. Then we have

$$\ddot{\mathcal{C}}_2 = \sum_{e \in E'} u_e \hat{c}_e$$
$$\hat{\mathcal{C}}_2 = \sum_{e \in E'} u_e \lceil \hat{c}_e \rceil.$$

According to the operation in ceiling procedure, it is evident that

$$\forall e \in E' : \ \hat{c}_e \leq \lceil \hat{c}_e \rceil \ < \ \hat{c}_e + 1.$$

As $\hat{c}_e \geq \alpha$ holds for any edge e in E', it is easy to prove that

$$\forall e \in E' : \ \lceil \hat{c}_e \rceil \ < \ \frac{1+\alpha}{\alpha} \cdot \hat{c}_e.$$

Then we have

$$\hat{\mathcal{C}}_2 \ < \ \frac{\alpha+1}{\alpha} \cdot \ddot{\mathcal{C}}_2.$$

This completes the proof. □

Algorithm 1. Multistage Approximation Algorithm

```
Input: Requests: ${1, 2, ..., K}$; Paths: ${P_1, P_2, ..., P_K}$;
       Bandwidth prices: ${u_e}$;
Output: Transmission paths for requests: ${x_{i,j}}$; Charging
        bandwidth: ${c_e}$;
1: Relax RL-SPM and solve its relaxed linear program
2: for $1 \leq i \leq K$ do
3:     Select exactly one path $P_{i,j}$ with probability $\hat{x}_{i,j}$
4: end
5: Initialize each $c_e$ as 0
6: for $e \in E$ do
7:     $c_e = \lceil \max_t \{\sum_{i=1}^{K} r_{i,t}\hat{x}_{i,j}I_{i,j,e}\} \rceil$
8: end
9: return ${x_{i,j}}$ and ${c_e}$
```

As one unit of bandwidth is large and α is a small integer in practice, the approximation ratio of ceiling procedure is very close to 1. We present MAA in Algorithm 1. On line 1, we solve the relaxation of $\mathcal{P}_1$. From line 2 to line 4, we select a path for each request according to the probability. On line 5, we initialize the bandwidth to be charged to zero. For each link $e \in E$, at each time slot t, we find the total bandwidth required by the requests that are scheduled to transmit through e and we round up the maximal fractional bandwidth to obtain the bandwidth for charging of e (from line 6 to line 8). On line 9, our algorithm returns the schedule and the charging bandwidth. We solve RL-SPM with MAA in polynomial time, which has a proved approximation ratio. The time complexity of MAA is $O(K \cdot T \cdot |E|)$, where K denotes the number of requests, T denotes the number of time slots in a billing cycle and $|E|$ denotes the number of edges in G. For MAA, we have the following theorem:

Theorem 3. *Suppose there exist a ρ_1-approximation algorithm ($\rho_1 \geq 1$) for $\mathcal{P}_1$ and a ρ_2-relaxed algorithm ($\rho_2 \geq 1$) for $\mathcal{P}_2$. Then there exists a ($\rho_1\rho_2$)-approximation algorithm for RL-SPM.*

Proof. We denote RL-SPM as $\mathcal{P}_0$ and the optimal objective of $\mathcal{P}_0$ as $\mathcal{C}_0^*$. First, we use ρ_1-approximation algorithm to solve $\mathcal{P}_1$. Denoting the output objective as $\hat{\mathcal{C}}_1$ and $\mathcal{C}_1^*$ the optimal objective value of $\mathcal{P}_1$. With the selected paths $\hat{x}_{i,j}$ for each request i, we have

$$\hat{\mathcal{C}}_1 \leq \rho_1 \mathcal{C}_1^* \leq \rho_1 \mathcal{C}_0^*.$$

Let $\ddot{\mathcal{C}}_2$ be the optimal solution of relaxed $\mathcal{P}_2$, then we have

$$\ddot{\mathcal{C}}_2 \leq \hat{\mathcal{C}}_1.$$

Second, we use ρ_2-relaxed algorithm to solve $\mathcal{P}_2$, whose objective is denoted by $\hat{\mathcal{C}}_2$. Hence we have

$$\hat{\mathcal{C}}_2 \leq \rho_2 \ddot{\mathcal{C}}_2.$$

Therefore, the following inequality

$$\hat{\mathcal{C}}_2 \leq \rho_2 \ddot{\mathcal{C}}_2 \leq \rho_2 \hat{\mathcal{C}}_1 \leq \rho_2\rho_1 \mathcal{C}_1^* \leq \rho_2\rho_1 \mathcal{C}_0^*,$$

holds. This completes the proof. □

Combining Theorems 2 and 3, we prove the approximation ratio of MAA in the following:

Theorem 4. *MAA is a $O(\frac{\alpha+1}{\alpha} \cdot \frac{\log n}{\log\log n})$-approximation algorithm for RL-SPM with high probability.*

Proof. Let $\hat{\mathcal{C}}_0$ denote the total bandwidth cost of RL-SPM obtained with MAA and $\mathcal{C}_0^*$ denotes the minimal bandwidth cost. Our goal is to prove that

$$\hat{\mathcal{C}}_0 \leq O\left(\frac{\alpha+1}{\alpha} \cdot \frac{\log n}{\log\log n}\right) \cdot \mathcal{C}_0^*. \tag{6}$$

Since randomized rounding procedure achieves a $O(\frac{\log n}{\log\log n})$-approximation ratio for $\mathcal{P}_1$ with high probability and ceiling procedure achieves a $(\frac{\alpha+1}{\alpha})$-approximation ratio for the relaxed $\mathcal{P}_2$, according to Theorem 3, we can prove that the above inequality (6) holds. This completes the proof. □

5 Algorithm for BL-SPM Solver

BL-SPM Solver aims to maximize the service revenue with given bandwidth. Since BL-SPM contains the unsplittable flow problem (denoted as UFP) as a special case, which is known to be NP-hard [25], [29], we have the following theorem:

Theorem 5. *BL-SPM is NP-hard.*

Proof. UFP is defined as follows: Given an n-vertex graph $G^* = (V^*, E^*)$ with edge capacity $\{c_e^*\}$, and a set of k vertex pairs $\{(s_i^*, d_i^*) : i = 1, \ldots, k\}$. Each pair (s_i^*, d_i^*) has a demand z_i^* and a weight w_i^*. The goal is to find the maximal weight subset of pairs, along with a path for each chosen pair, so that the entire demand for each such pair can be routed on its path while respecting the capacity constraints [25].

We construct a special input of BL-SPM based on the definition of UFP. Consider that we have a network $G = G^*$ and k user requests $\{(s_i, d_i, ts_i, td_i, r_i, v_i) : s_i = s_i^*, d_i = d_i^*, \ r_i = z_i^*, \ v_i = w_i^*, \ ts_i = 0, \ td_i = 1, \ 1 \leq i \leq k\}$ in a billing cycle. For each $e \in E$, its capacity c_e is equal to c_e^*

in the network G^*. The goal of BL-SPM is to find a subset of user requests that can be satisfied without violating any bandwidth constraints while maximizing the service revenue of cloud providers. If we can solve BL-SPM with a polynomial-time algorithm, we can obtain the routing solution for UFP. Thus, BL-SPM is at least as hard as UFP. As UFP is NP-hard, BL-SPM is NP-hard, too. This completes the proof. □

To avoid the violation of link capacity constraint due to randomized rounding, we propose an approximation algorithm to solve BL-SPM with a performance guarantee. We present the details in the following.

First, we relax BL-SPM that binary variables $x_{i,j}$ can be fractional. We denote the optimal scheduling solution of the relaxed problem as $\{\hat{x}_{i,j}\}$. The corresponding revenue is denoted by $\hat{\mathcal{I}}$. Similarly, we denote the optimal objective of BL-SPM as $\mathcal{I}^*$.

Since requests are independent from each other, we select a path $P_{i,j}$ for request i independently with a probability $x_{i,j}$ and denote by $\mathcal{I}_i$ the service revenue generated by request i. The expectation of $\mathcal{I}_i$ is

$$E[\mathcal{I}_i] = v_i \sum_{j=1}^{L_i} x_{i,j} = m_i,$$

where v_i denotes the value of request i. Then we have

$$E[\mathcal{I}] = \sum_{i=1}^{K} E[\mathcal{I}_i] = m,$$

where K denotes the number of requests. Based on the following Chernoff-Hoeffding bounds [28], [30]:

Theorem 6 (Chernoff-Hoeffding bounds). *Suppose $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_K$ are K independent random variables in [0,1] and $\mathcal{I} = \sum_{i=1}^{K} \mathcal{I}_i$. For $\delta > 0$, and $m = E[\mathcal{I}] \geq 0$,*

$$Pr[\mathcal{I} > (1+\delta)m] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^m.$$

For $0 < \gamma \leq 1$,

$$Pr[\mathcal{I} < (1-\gamma)m] < \left[\frac{e^{\gamma}}{(1+\gamma)^{(1+\gamma)}}\right]^m,$$

we normalize the expectation of $\mathcal{I}_i$ to [0,1] to adapt to the above theorem, then we define $B(m, \delta)$ as

$$B(m,\delta) = \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^m,$$

to be the probability that $\mathcal{I}$ is within a given δ ratio around m, i.e., $(1-\delta)m < \mathcal{I} < (1+\delta)m$, with the random selection of paths for requests. We also define $D(m,x)$ to replace δ and make the following equality hold:

$$B(m, D(m,x)) = x.$$

Next, we use both $B(\cdot)$ and $D(\cdot)$ to guide the finding of feasible solutions of BL-SPM.

Since link capacity constraints may be violated if a path $P_{i,j}$ is selected for request i with the probability $\hat{x}_{i,j}$, we introduce a *scaling factor* μ ($0 < \mu < 1$), and take $\mu\hat{x}_{i,j}$ as the probability that we select the path $P_{i,j}$. To ensure that the probability of violating the link capacity constraint of BL-SPM is less than $\frac{1}{T\cdot(N+1)}$, we set μ to satisfy the following inequality:\

$$B\left(\mu c, \frac{1-\mu}{\mu}\right) < \frac{1}{T\cdot(N+1)}, \tag{7}$$

where c denotes the minimum bandwidth of edges in the WAN (except the edges whose bandwidth is zero); T and N denote the number of time slots in a billing cycle and the number of edges in G, respectively.

By our choice of μ, the probability that a constraint is violated will be less than $\frac{1}{T\cdot(N+1)}$. Since there are $T \cdot N$ link capacity constraint in BL-SPM, the probability that there exists at least one violated constraint is less than $\frac{N\cdot T}{T\cdot(N+1)}$. The expectation of the service revenue, $\mathcal{I}_S$, will be scaled down, i.e., $\mathcal{I}_S = \mu\hat{\mathcal{I}}$. Based on it, we have the following theorem:

Theorem 7. *Suppose $\mathcal{I}_B = \mathcal{I}_S \cdot [1 - D(\mathcal{I}_S, \frac{1}{N+1})]$, there exist solutions that make the service revenue of BL-SPM, $\widetilde{\mathcal{I}}$, satisfy*

$$\widetilde{\mathcal{I}} \geq \mathcal{I}_B. \tag{8}$$

Proof. If a solution makes the theorem hold, we say the corresponding schedule is "good", otherwise it is "bad". To prove the theorem, we will show that there exists non-zero probability that inequality (8) holds while no constraint in BL-SPM is violated (denoted as Y) in the following, i.e.,

$$Pr[\widetilde{\mathcal{I}} \geq \mathcal{I}_B \cap Y] > 0. \tag{9}$$

By inequality (8), we have

$$Pr[\widetilde{\mathcal{I}} < \mathcal{I}_B] < \frac{1}{N+1}. \tag{10}$$

Since there are $N \cdot T$ link capacity constraints in BL-SPM, and the probability that any of them is violated is less than $\frac{1}{T\cdot(N+1)}$, the probability that there exists at least one violated link capacity constraint (denoted by $\bar{Y}$) is less than $\frac{N\cdot T}{T\cdot(N+1)}$. Then we have

$$\begin{aligned} &Pr[\widetilde{\mathcal{I}} < \mathcal{I}_B \cup \bar{Y}] \\ &\leq Pr[\widetilde{\mathcal{I}} < \mathcal{I}_B] + Pr[\bar{Y}] \\ &< \frac{1}{N+1} + T\cdot N \cdot \frac{1}{T\cdot(N+1)} = 1. \end{aligned} \tag{11}$$

Since the sum of $Pr[\widetilde{\mathcal{I}} < \mathcal{I}_B \cup \bar{Y}]$ and $Pr[\widetilde{\mathcal{I}} \geq \mathcal{I}_B \cap Y]$ equals 1 and $Pr[\widetilde{\mathcal{I}} < \mathcal{I}_B \cup \bar{Y}] < 1$, we have that $Pr[\widetilde{\mathcal{I}} \geq \mathcal{I}_B \cap Y] > 0$, thus inequality (9) holds, implying the solutions achieve objectives higher than $\mathcal{I}_B$. This completes the proof. □

As proven in Theorem 7, there exist good solutions for BL-SPM. We propose an algorithm to construct a good solution for BL-SPM based on conditional probabilities.

Considering that there are K user requests in a billing cycle, we construct a K-level tree (denoted as $\mathcal{T}$). The nodes at the ith level represent the possible scheduling choices for

i. For any request i, there are $L_i + 1$ choices, either scheduling it on a path $P_{i,j}$ $(1 \leq j \leq L_i)$ or declining it. For the convenience of presentation of algorithm process, we consider declining request i as scheduling it on the path P_{i,L_i+1}, thus a request i has $L_i + 1$ possible routing paths. For K requests, there are totally $\prod_{i=1}^{K}(L_i + 1)$ possible schedules, each represented as a leaf node of $\mathcal{T}$. Theorem 7 shows there exist good schedules for BL-SPM. We can get that there exist "good" leaf nodes in $\mathcal{T}$. We aim to walk down $\mathcal{T}$ from the root node to a good leaf node in polynomial time so that the resulted service revenue $\widetilde{\mathcal{I}}$ is no less than $\mathcal{I}_B$.

Suppose that, at a typical stage of the computation, we have walked down the first $i-1$ levels of $\mathcal{T}$ and we are at the ith level $(1 \leq i \leq K)$. Now, we wish to select one of the $L_i + 1$ nodes to minimize the probability that we reach a "bad" leaf node that represents a bad schedule. We denote by $p_i(q_1, \ldots, q_{i-1})$ the conditional probability that we will reach a bad leaf node given the choices of the first $i-1$ requests, where q_k denotes the selected path of request k. Then

$$p_i(q_1, \ldots, q_{i-1}) = \sum_{j=1}^{L_{i+1}+1} \{x_{i,j} \cdot p_{i+1}(q_1, \ldots, q_{i-1}, P_{i,j})\},$$

where $L_{i+1} + 1$ denotes the number of possible choices of request $i+1$. It is easy to derive the following inequality:

$$p_i(q_1, \ldots, q_{i-1}) \geq \min_j\{p_{i+1}(q_1, \ldots, q_{i-1}, P_{i,j})\}. \qquad (12)$$

We define $p(leaf)$ as the probability that we reach a bad leaf node at level K, and $p(leaf) = 0$ when we reach a good leaf node and $p(leaf) = 1$ otherwise.

As there exists at least one good schedule for BL-SPM by Theorem 7, there exists at least one leaf node making the following inequality hold:

$$1 > p_1 > \ldots > p_K(q_1, \ldots, q_{K-1}) > p(leaf) = 0.$$

As computing all of the conditional probabilities is time-consuming, we relax $p_i(q_1, \ldots, q_{i-1})$ to $u_i(q_1, \ldots, q_{i-1})$, which is defined to be an upper bound on $p_i(q_1, \ldots, q_{i-1})$, i.e.,

$$u_i(q_1, \ldots, q_{i-1}) \geq \min_j\{u_{i+1}(q_1, \ldots, q_{i-1}, P_{i,j})\}. \qquad (13)$$

Next we walk down the tree $\mathcal{T}$ from the root node (i.e., level 0) to a good leaf node by continuously decreasing $u_i(\cdot)$. Inspired by the pessimistic estimators in [30], we define $u_0(\cdot)$ as follows:

$$u_0 = e^{t_0 \mathcal{I}_S} \prod_{i=1}^{K} \left\{ \sum_{j=1}^{L_i} \mu \hat{x}_{i,j} e^{-t_0 v_i} + 1 - \sum_{j=1}^{L_i} \mu \hat{x}_{i,j} \right\} + \sum_{k=1}^{T \cdot N} \left\{ e^{-t_k c} \prod_{i=1}^{K} \left[\sum_{j=1}^{L_i} \mu \hat{x}_{i,j} e^{t_k r_{i,t} I_{i,j,e}} + 1 - \sum_{j=1}^{L_i} \mu \hat{x}_{i,j} \right] \right\}.$$

In the case that $t_0 = \ln\, [1 + D(\mathcal{I}_S, \frac{1}{N+1})]$, $t_k = \ln\, [1 + \frac{1-\mu}{\mu}]$, and $r_i, v_i \in [0, 1]$ for any request i, it is proved that $u_0 < 1$ holds. With the parameters set as above, we select a path for each quest i to minimize the value of $u_i(\cdot)$ from level 1 to level K. Suppose we are at level i and we have selected $P_{i,j}$ for request i, then we replace $\hat{x}_{i,j}$ and $\hat{x}_{i,j^*}$ $(j^* \neq j)$ in the expression of u_0 with 1 and 0, and calculate $u_i(\cdot)$. At each level, we compare different choices to determine the one that minimizes $u_i(\cdot)$.

Algorithm 2. Tree-Based Approximation Algorithm

```
Input: Requests: $\{1, 2, \ldots, K\}$; Paths: $\{P_1, P_2, \ldots, P_K\}$;
       Bandwidth: $\{c_e\}$;
Output: Transmission paths for requests: $\{x_{i,j}\}$;
 1: Normalize the rates and values of requests
 2: Relax BL-SPM and solve its relaxed linear program
 3: Select the scaling factor $\mu$ according to (7)
 4: for $1 \leq i \leq K$ do
 5:     for $1 \leq j \leq L_i + 1$ do
 6:         if fixing $x_{i,j}$ to 1 minimizes $u_0$ then
 7:             $x_{i,j} \leftarrow 1$
 8:         else
 9:             $x_{i,j} \leftarrow 0$
10:         end
11:     end
12: end
13: return $\{x_{i,j}\}$
```

Based on the above definitions, we propose an algorithm to reach a good leaf node from root node thus finding a good schedule for BL-SPM. For request i, there are $L_i + 1$ available paths to choose from. By selecting different paths for request i, we calculate the values of $u_i(\cdot)$ and finally choose the path that minimizes it. By this method, we continuously decrease the probability of reaching a bad leaf node thus we could reach a good leaf node at the last level. Since there are K requests, we repeat the above procedures for K times to reach a good leaf node. We present our algorithm, tree-based approximation algorithm (*TAA*), in Algorithm 2. Despite there are $\prod_{i=1}^{K}(L_i + 1)$ leaf nodes in $\mathcal{T}$, from the root node to a good leaf node, our algorithm needs to do at most $\sum_{i=1}^{K}(L_i + 1)$ examination. Compared with the brute force method, the computing time can be greatly reduced.

So far, we have proposed two algorithms, MAA and TAA, to solve RL-SPM and BL-SPM respectively. We install them in Metis to find a good schedule for cloud providers to make more service profit.

6 Handling User Requests in Sequence

In this section, we first present the overview of our online solution, then we give the design details, including an online scheduling algorithm and two subroutines.

6.1 Overview

In the previous sections, we have designed Metis and two algorithms to help the cloud providers make more service profit by handling user requests in batch. It requires users to submit their requests at the beginning of each scheduling period. In many cases, however, corporate users do not know the actual bandwidth demands in advance. Thus we design an online solution to complement the offline one that cloud providers can process user requests in real time. Intuitively, as cloud providers cannot predict the future requests exactly, when a new request requires the cloud provider to purchase more bandwidth from Internet service providers, the cloud provider has to assess the risk that the incremental cost is not paid off. Balancing this risk and the incremental service revenue from satisfying the new user request is challenging for cloud providers.

In the past few decades, simulated annealing method has been well studied and applied in different scenarios [10]. It is a probabilistic technique for approximating the global optimum that is computationally expensive to find in a reasonable amount of time. Specifically, it is a classic heuristic to approximate the global optimization in a large search space for an optimization problem. Inspired by the principles used by simulated annealing, we propose an *online scheduling algorithm* (OSA) to search for an approximate solution for cloud providers to make maximal service profit. In the following, we first introduce the routine of OSA. Then we introduce the subroutines to move from one state (i.e., schedule) to a neighbor state and compute the corresponding energy that the new state has.

In the cloud market, corporate users usually submit bandwidth requests whenever they need it and expect timely feedback from cloud providers. Cloud providers face two challenges in handling sequentially-arrived user requests: (1) Conflict between the duration of user requests and billing cycle posed by Internet service providers. Typically, cloud providers sign long-term contracts with Internet service providers for the large-volume bandwidth supply [6], [12]. The bandwidth charges are closely related to the peak (or near-peak) usage of bandwidth in the billing period, while users usually request short-term, small-volume bandwidth, which makes it hard for cloud providers to judge whether the acceptance of the new request is a good decision or not. (2) Too many paths for delivering user requests on. In a network with n nodes, there are $O(n!)$ possible routing paths between any two nodes. Arranging a request on different paths will lead to great differences in final service profit (as shown in Fig. 1), especially when cloud providers know nothing about future requests.

For cloud providers, a naive approach is to greedily accept user requests when they can bring in positive earnings, i.e., incremental revenue minus incremental cost is greater than zero. As users usually request short-term, low-volume bandwidth and prefer to pay a small amount of money as the reward of cloud providers, when accepting one user request requires cloud providers to purchase one more unit of bandwidth, the value of user request minus the cost of incremental bandwidth would be far less than zero. In the worst case, cloud providers would decline all submitted requests to ensure that the total profit is not negative. Obviously it is inconsistent with the intention of cloud providers, thus this greedy method will not yield good performance results.

From above discussion, buying one more unit of bandwidth to satisfy new user requests would be cost-inefficient as it will lead to huge profit loss to cloud providers. If we can make full use of the remaining bandwidth after serving new requests, it may make the large bandwidth cost pay off. Moreover, more remaining bandwidth is likely to serve more future requests and earn more service revenue without incurring the new bandwidth cost. Following this thought, we can justify the earnings of accepting one request by adding the incremental service profit and the remaining bandwidth together. More specifically, when the kth request (denoted as k) arrives at time slot t, we can calculate the "energy" of the current schedule as

$$\mathcal{E}_t \triangleq \mathcal{I}_t - \mathcal{C}_t + w \cdot \mathcal{B}_t,$$

where $\mathcal{I}_t$ and $\mathcal{C}_t$ represent the current service revenue and service cost, and $\mathcal{B}_t$ denotes the transmission capability of unallocated bandwidth in the remaining period (i.e., from current time to the end of billing cycle). w is used to balance the current service profit (i.e., $\mathcal{I}_t - \mathcal{C}_t$) and remaining transmission capability (i.e., $\mathcal{B}_t$). It is set by cloud providers. A large w means that cloud providers are willing to take risks to maximize the final service profits. Otherwise, they will use a small w to avoid loss of earned service profits. $\mathcal{I}_t$ and $\mathcal{C}_t$ can be calculated based on the definitions in Section 2.2. $\mathcal{B}_t$ is calculated as

$$\mathcal{B}_t \triangleq \sum_{e \in E} \left\{ (T - t) \cdot c_{e,t} - \sum_{i=1}^{k} \sum_{j=1}^{P_i} \sum_{l=t}^{T} r_{i,l} \cdot x_{i,j} \cdot I_{i,j,e} \right\},$$

where $c_{e,t}$ denotes the bandwidth of edge e at time t. Based on the above definitions, we can calculate the energy each schedule has at any time slot. In the following, we present the detail designs of our online scheduling algorithm.

Algorithm 3. Online Scheduling Algorithm

```
1: $\mathcal{E}_{current} \leftarrow ComputeEnergy(\mathcal{S}_{current})$
2: $\Omega \leftarrow \mathcal{E}_{current}$
3: $\mathcal{S}^* \leftarrow \mathcal{S}_{current}$
4: $\mathcal{E}^* \leftarrow \mathcal{E}_{current}$
5: while $\Omega > \epsilon$ do
6:     $\mathcal{S}_{neighbor} \leftarrow ComputeNeighbor(\mathcal{S}_{current})$
7:     $\mathcal{E}_{neighbor} \leftarrow ComputeEnergy(\mathcal{S}_{neighbor})$
8:     if $\mathcal{E}_{neighbor} > \mathcal{E}^*$ then
9:         $\mathcal{S}^* \leftarrow \mathcal{S}_{neighbor}$
10:        $\mathcal{E}^* \leftarrow \mathcal{E}_{neighbor}$
11:    end
12:    if $\mathcal{P}(\mathcal{E}_{current}, \mathcal{E}_{neighbor}, \Omega) > Rand(0,1)$ then
13:        $\mathcal{S}_{current} \leftarrow \mathcal{S}_{neighbor}$
14:        $\mathcal{E}_{current} \leftarrow \mathcal{E}_{neighbor}$
15:    end
16:    $\Omega \leftarrow \Omega \times \beta$
17: end
18: return $\mathcal{S}^*$
```

6.2 Online Scheduling Algorithm

In Algorithm 3, the current request schedule is used as the initial state. When a user request k arrives at time t, we first calculate the energy, denoted as $\mathcal{E}_{current}$, of the current state $\mathcal{S}_{current}$ without considering the arrangement of request k on line 1 by *ComputeEnergy* (Algorithm 5). We set the value of indicator variable Ω as the current energy $\mathcal{E}_{current}$ on line 2. $\mathcal{S}^*$ and $\mathcal{E}^*$ are used to store the state that has the highest energy and the energy of $\mathcal{S}^*$, respectively. We initialize the values of $\mathcal{S}^*$ and $\mathcal{E}^*$ as $\mathcal{S}_{current}$ and $\mathcal{E}_{current}$ on line 3 and 4. From line 5 to line 17, we iteratively try new state (with the consideration of k) to find the one with the highest energy. In each iteration, we use *ComputeNeighbor* (Algorithm 4) and *ComputeEnergy* to determine a new state and calculate the corresponding energy on line 6 and 7. If the energy of the new state is higher than $\mathcal{S}^*$, i.e., $\mathcal{E}_{neighbor} > \mathcal{E}^*$, we will update the values of $\mathcal{S}^*$ and $\mathcal{E}^*$ (on line 8-11). From line 12 to line 15, we use a function $\mathcal{P}(\cdot)$ to calculate the probability that we replace the current state $\mathcal{S}_{current}$ with the neighbor state $\mathcal{S}_{neighbor}$, as well as state energy. We define $\mathcal{P}(\mathcal{E}_{current}, \mathcal{E}_{neighbor}, \Omega)$ as follows: if the neighbor state $\mathcal{S}_{neighbor}$ has a higher energy than that of the current state $\mathcal{S}_{current}$, we will execute the action of moving from $\mathcal{S}_{current}$ to $\mathcal{S}_{neighbor}$ deterministically, i.e., the probability equals 1. Otherwise, we execute the movement with a probability of $e^{\frac{\mathcal{E}_{neighbor}-\mathcal{E}_{current}}{\Omega}}$. At the end of each iteration, we will decrease the value of Ω by a factor of β. With the method of simulated annealing, we make scheduling decision for the arriving request k.

Algorithm 4. Generate Neighbor State

```
1: function ComputeNeighbor($\mathcal{S}$)
2: Select one path $p$ from $P_k$
3: $P_k \leftarrow P_k \backslash p$
4: $\mathcal{S}_{neighbor} \leftarrow \mathcal{S} + p$
5: return $\mathcal{S}_{neighbor}$
```

Algorithm 5. Compute State Energy

```
 1: function ComputeEnergy($\mathcal{S}$)
 2: $\mathcal{I}_t \leftarrow 0$
 3: $\mathcal{C}_t \leftarrow 0$
 4: $\mathcal{B}_t \leftarrow 0$
 5: for $i = 1$ to $k$ do
 6:     if $i$ has been accepted then
 7:         $\mathcal{I}_t \leftarrow \mathcal{I}_t + v_i$
 8:     end
 9: end
10: $\mathcal{E} \leftarrow \mathcal{I}_t$
11: for each $e \in E$ do
12:     $\mathcal{C}_t \leftarrow \mathcal{C}_t + u_e \times c_{e,t}$
13: end
14: $\mathcal{E} \leftarrow \mathcal{E} - \mathcal{C}_t$
15: for each $e \in E$ do
16:     $\mathcal{B}_t \leftarrow \mathcal{B}_t + (T - t) \times c_{e,t}$
17:     for $i = 1$ to $k$ do
18:         for $j = 1$ to $P_i$ do
19:             if $i$ is scheduled on $P_{i,j}$ and $e \in P_{i,j}$ then
20:                 for $l = t$ to $T$ do
21:                     $\mathcal{B}_t \leftarrow \mathcal{B}_t - r_{i,l}$
22:                 end
23:             end
24:         end
25:     end
26: end
27: $\mathcal{E} \leftarrow \mathcal{E} + w \times \mathcal{B}_t$
28: return $\mathcal{E}$
```

6.3 Subroutine: ComputeNeighbor

This subroutine is used to find a routing path for request k on the basis of the current schedule for the first $k-1$ requests. As there exist L_k available paths between s_k and d_k, we have $L_k + 1$ options to schedule the request k, i.e., delivering it on one of the L_k paths or declining it. We search for a new state based on the current state to find a better schedule. We pick one possible path from the path set P_k as the (tentative) schedule of request k (on line 2). To avoid trying one path for many times, we remove the selected path p from P_k (on line 3) and add it to the current state $\mathcal{S}$ to obtain a new state $\mathcal{S}_{neighbor}$ (on line 4). We finally return the new state (on line 5). To minimize the changes in each iteration, we select path p based on current state $\mathcal{S}$ and change as few links as possible.

6.4 Subroutine: ComputeEnergy

Suppose the current time is t. In this subroutine, we calculate the energy of state $\mathcal{S}$ based on the definitions of $\mathcal{I}$, $\mathcal{C}$ in Section 2.2 and $\mathcal{B}_t$ in this section. From line 2 to 4, we initialize the values of $\mathcal{I}_t$, $\mathcal{C}_t$ and $\mathcal{B}_t$ to 0. On lines 5 to 9, we calculate $\mathcal{I}_t$ by adding up the values of all accepted requests. The time complexity is $O(K)$. We initialize the value of energy $\mathcal{E}$ as $\mathcal{I}_t$ on line 10. On lines 11 to 13, we calculate the current bandwidth cost $\mathcal{C}_t$, whose time complexity is $O(|E|)$. We update the value of $\mathcal{E}$ on line 14. On lines 15 to 26, we calculate the value of $\mathcal{B}_t$ with complexity $O(|E| \cdot K \cdot T)$. Then we update the value of $\mathcal{E}$ with $\mathcal{B}_t$ on line 27. Finally, we return the energy $\mathcal{E}$ on line 28.

7 Evaluation

In this section, we conduct extensive evaluations to compare our solutions with existing ones. We present the evaluation methodology and results in the following.

7.1 Evaluation Methodology

To comprehensively show the performance of our solutions and existing schemes, we conduct evaluations on two networks:

- B4: Google's Inter-DC WAN [7], which consists of 12 data centers and 19 bi-directional links (as shown in Fig. 3). It is a typical Inter-DC WAN.
- SUB-B4: a sub-network of B4, which consists of 6 data centers (i.e., DC1 $\sim$ DC6) and 7 links between them. It is a small-scale Inter-DC WAN.

We set the bandwidth prices of Inter-DC links based on the relative bandwidth prices provided by Cloudflare [13]. Using 10Gbps as a unit of bandwidth, the bandwidth cost is the product of bandwidth price and bandwidth usage. Following the previous work [10], [26] and the common practice in the cloud service market, we generate bandwidth requests with a synthetic model as follows: in a billing cycle consisting of 12 time slots, which represent 12 months in a year, the arrivals of requests follow Poisson distribution and the bandwidth requirements follow uniform distribution between 0.1 Gbps and 5 Gbps. The start time and the end time of requests are randomly set within the billing cycle. For any request, its source and destination are selected randomly in B4 or SUB-B4. We set the value of user request based on the bandwidth requirements and the bandwidth prices published by cloud providers [31], [32], [33].

We select a set of representative solutions to compare with our solutions: 1) *MinCost*. Using fixed rules in scheduling, it always selects the path with the least bandwidth price (i.e., min-cost path) to deliver traffic data between data centers. In our evaluation, it reserves exclusive bandwidth for users on the min-cost paths. 2) *Amoeba* [26]. It is an Inter-DC flow scheduler to satisfy as many user requests as possible under a fixed amount of bandwidth. 3) *EcoFlow* [20]. It is an economical and deadline-driven flow scheduler. It sends an

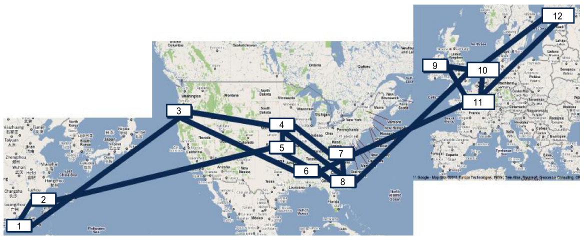


Fig. 3. Network topology of B4 [7].

inter-datacenter flow on multiple routing paths simultaneously to make full use of the WAN bandwidth, without considering the performance concerns caused by packet reordering. In our evaluation, it handles user requests one by one and accepts the user requests that generate higher service profits. As the service values of requests are fixed in our problem, it is difficult to compare Metis and OSA with the pricing-based solutions such as Pretium [12]. Thus we have not compared our solutions with them. We implement a simulator that integrates the above solutions with C++ and call the Gurobi Optimizer 7.5.2 [34] to solve the LP and ILP problems.

7.2 Evaluation Results and Analyses

We conduct extensive evaluations to test our solutions and compare them with existing solutions.

7.2.1 *Performance of MAA*

In Fig. 4a, we compare the service cost of MAA and MinCost on B4 network. To satisfy the same set of requests, the service cost of MinCost is up to 21.1 percent higher than that of MAA. Scheduling requests based on the fixed policy to reserve bandwidth on the min-cost path without considering the relationship between requests, MinCost suffers from higher bandwidth usage and higher bandwidth cost. Different from MinCost, MAA is an LP-based algorithm. Benefitting from the relax-and-round method and selecting routing path based on the optimal solution of the relaxed RL-SPM problem, it efficiently schedules user requests to make full use of the purchased bandwidth. In Fig. 4a, the difference of service cost between Metis and MinCost increases with the request number. This is because the shortages of the fixed scheduling policy used by MinCost worsens its path selection and increases its bandwidth cost, while MAA has more potential to fully use the purchased bandwidth and generate less bandwidth cost.

In MAA, we adopt randomized rounding method to decide routing paths for user requests based on the optimal solution of relaxed RL-SPM problem. In Fig. 4b, we present the ratio of bandwidth cost with randomized rounding to that of the optimal scheduling in different network settings. For each set of user requests, we repeat the randomized rounding procedure for 1,000 times and calculate the bandwidth cost. It shows that, the ratio is always less than 1.2, which demonstrates that our algorithm can achieve a good performance that is comparable to the optimal solution.

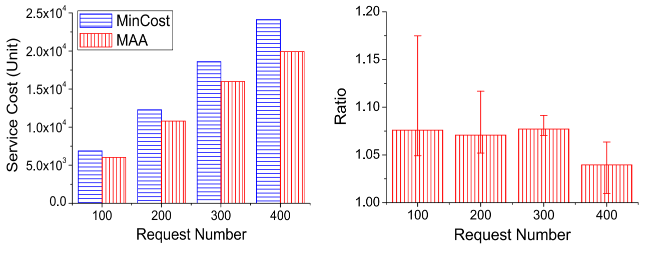


Fig. 4. Performance of MAA on B4.

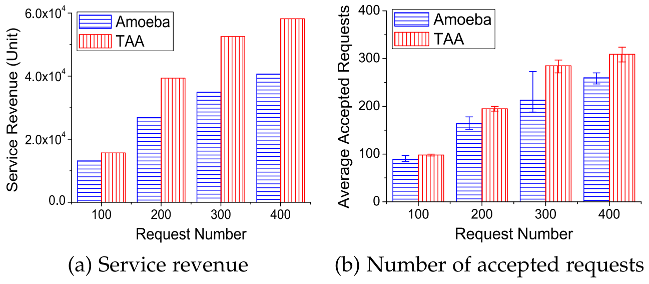


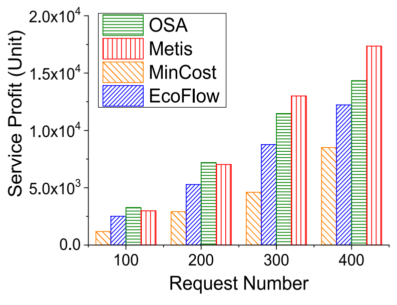
Fig. 5. Performance of TAA on B4.

7.2.2 *Performance of TAA*

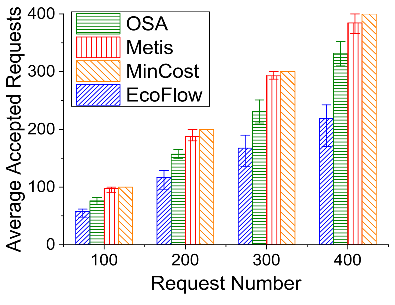
As TAA and Ameoba [26] work with fixed bandwidth to satisfy more requests for the maximum service revenue, following the setup in [26], where each link has a uniform capacity, we set the bandwidth of links in the B4 network to 100 Gbps, i.e., 10 units of bandwidth. We evaluate the service revenue and the number of satisfied requests under different situations. In Fig. 5a, with the increase of request number, the growth of service revenue of TAA is faster than that of Amoeba. TAA can get up to 50.4 percent more service revenue than Amoeba. Benefitting from the optimal solution of relaxed BL-SPM and the efficient search method in the decision tree, TAA accepts more user requests than Amoeba with fixed bandwidth. In Fig. 5b, the average number of accepted requests of TAA is up to 33 percent higher than that of Amoeba. As Amoeba handles requests one by one to accept the ones that can be accommodated by the residual bandwidth without considering future requests, the performance is compromised. Taking into account all user requests, TAA makes more service revenue with a relatively good scheduling solution.

7.2.3 *Performance of Metis and OSA*

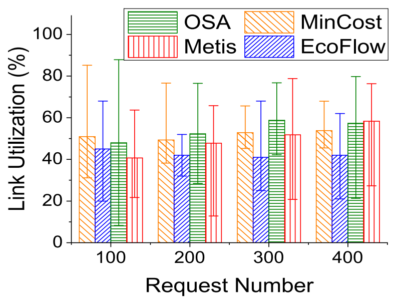
In Fig. 6, we compare Metis and OSA with EcoFlow [20] and MinCost on the B4 network. In Fig. 6a, the service profit of Metis and OSA is up to 48.4 and 36.3 percent higher than that of EcoFlow. EcoFlow adopts a greedy method to make acceptance decision and it declines too many user requests, while Metis alternately runs MAA and TAA to look for better acceptance and scheduling solutions to maximize the service profit. OSA makes decisions online by taking into account the current profit and the potential earnings simultaneously. The profit of Metis and OSA is at least 68.5 percent higher than that of MinCost, which delivers user requests on the min-cost paths without considering the relationship between requests. Benefitted from simulated annealing, OSA arranges accepted requests on proper routing paths (may not be the min-cost path) to make full use of the purchased bandwidth.



(a) Service profit

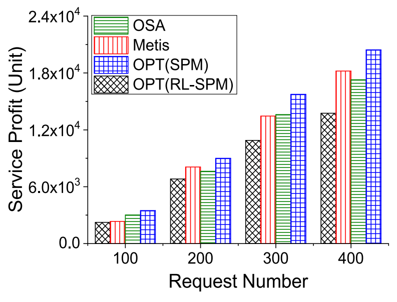


(b) Number of accepted requests

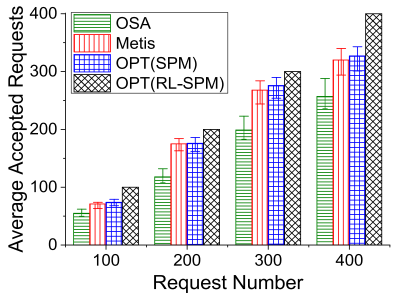


(c) Link utilization

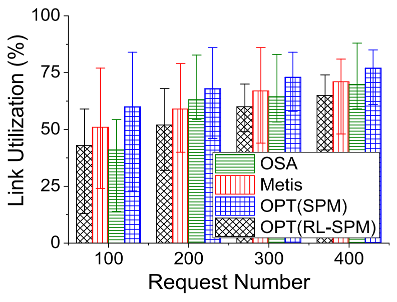
Fig. 6. Performance of solutions on B4.



(a) Service profit



(b) Number of accepted requests



(c) Link utilization

Fig. 7. Performance of solutions on SUB-B4.

In Fig. 6b, we compare the numbers of accepted requests in different scheduling solutions. MinCost accepts all user requests while EcoFlow, OSA and Metis decline some user requests to maximize the service profit of cloud providers. The accepted request number of EcoFlow is up to 43.1 and 33.9 percent less than that of Metis and OSA respectively. The request acceptance rates of Metis and OSA exceed 90 and 76 percent. By declining a few requests, our solution can gain much higher service profit than the full-acceptance solution.

As shown in Fig. 6c, the average link utilization of Metis and OSA is up to 38.8 and 43.2 percent higher than that of EcoFlow, which is generally lower than other solutions. This is because Metis and OSA accept more requests and properly schedule them to fully use the purchased bandwidth and make more service profit. Although MinCost has a higher average link utilization than EcoFlow, its acceptance of all user requests is not conducive to maximizing the service profit. From the above result, we find that it is necessary to make good acceptance and scheduling decisions simultaneously for maximizing the service profit.

7.2.4 Comparing With Optimum Solution

To evaluate the difference between our solutions and the optimal solution of SPM on maximizing service profit and verify the benefit of declining some user requests for higher cloud service profit, we compare Metis and OSA with the optimal solution of SPM, denoted as "OPT(SPM)", and the optimal solution with all user requests accepted, denoted as "OPT(RL-SPM)", on the SUB-B4 network. In our evaluation, OPT(RL-SPM) takes the optimal solution to schedule user requests with the minimum bandwidth cost. We show the comparing results in Fig. 7.

In Fig. 7a, OPT(SPM) makes more service profit than Metis, OSA and OPT(RL-SPM) by directly solving the ILP problem to maximize the profit. When there are 400 user requests, it takes more than 1,000 seconds to get the optimal request schedule while Metis uses only several hundreds of milliseconds to calculate a feasible schedule. Although the profit of Metis is 11 percent lower than that of the optimal one using OPT(SPM), it is still 32.3 percent higher than that of OPT(RL-SPM), which shows the advantage of Metis on making more service profit and the shortage of accepting all user requests without considering the bandwidth cost. The service profit of OSA is 14.5 percent lower than that of OPT(SPM) but 24.4 percent higher than that of OPT(RL-SPM). This is because OPT(SPM) makes decisions with the perfect request information while OSA handles requests in real-time.

In Fig. 7b, OPT(RL-SPM) accepts all the user requests, while OPT(SPM), Metis and OSA accept only part of the user requests. To satisfy all requests, OPT(RL-SPM) has to purchase more bandwidth than others, while some of the bandwidth is underused. Without accepting all requests, OPT (SPM), Metis and OSA have larger spaces to search for a better schedule to fully use the purchased bandwidth and make more service profit than OPT(RL-SPM). The request acceptance ratio of OSA is 61 percent on average, as it declines more requests to avoid incremental bandwidth cost that do not pay off. The request acceptance rates of OPT(SPM) and Metis are 84 and 82 percent on average. Compared to Fig. 6b, Metis and OSA accept more user requests in the larger network, since they have more scheduling options.

In Fig. 7c, we present the link utilization of different solutions, including the maximum, minimum and average link utilization. OPT(SPM) has the highest average link utilization, while OPT(RL-SPM) has the lowest one by satisfying all user requests even with profit loss.

TABLE 2
Computation Time of OPT(SPM) and Metis

Request Number	Computation Time (second)			
	B4		SUB-B4	
	OPT(SPM)	Metis	OPT(SPM)	Metis
100	$13 \sim 35$	< 0.01	$1 \sim 5$	< 0.01
200	$150 \sim 450$	$0.01 \sim 0.1$	$5 \sim 15$	< 0.01
300	$5000 \sim 8000$	$0.3 \sim 2$	$100 \sim 300$	$0.01 \sim 0.1$
400	> 10000	$5 \sim 15$	$300 \sim 1200$	$0.1 \sim 0.5$

As Metis and OPT(SPM) handle user requests in batch and the computation time may be a concern for cloud providers. We compare the computation time of Metis and OPT(SPM) with different network settings, including network scale and request number. We run them on a laptop with Intel Core i7-7500U 2.70 GHz CPU, 8 GB memory, 240 GB SSD and Windows 10 64-bit operating system. The network settings and the computing time of OPT and Metis are summarized in Table 2. As the network scale and request number increase, the computation time of OPT increases quickly. When there are 400 requests on B4, OPT takes more than 10,000 seconds to make the request acceptance and scheduling decisions. Even on the small-scale network SUB-B4, the computation time exceeds 1,000 seconds sometimes. This is because the ILP optimizer needs to deal with tons of integer variables under complicated constraints.

Generally, compared with an ILP problem, solving an LP problem of the same size can be completed with much less computation time. As Metis schedules user requests by alternately running MAA and TAA, both of which are LP-based algorithms, it needs only several seconds to deal with 400 requests in a typical Inter-DC WAN, which is hundreds of times faster than that of the optimal solution. Taking into account the computation time and profit difference between our solutions and the optimal solution, it is more proper to use Metis and OSA in practice.

7.2.5 *Performance of OSA With Different Routing Paths*

In a large-scale network, there are many available routing paths between any node pairs and exploring all of them is time-consuming. In Fig. 8, we explore H shortest paths to test the service profit using OSA. We set the "length" of links as the their bandwidth prices and use Yen's algorithm [35] to find the top-H shortest paths between any node pairs and number them from 1 to H. As shown in Fig. 8a, for the same request number (RN), the service profit changes little when we vary H from 1 to 20. It proves that, cloud providers can get a relatively high service profit in short time by trying a small number of shortest routing paths, rather than exploring all routing paths.

Fig. 8b shows the number of accepted flows with different H. We can find that, when there are more available routing paths, the number of accepted requests changes slightly. This is because OSA tends to deliver user requests on shorter routing paths. Even if there are many other paths, OSA arranges requests on one of a few shortest paths.

To help cloud providers set a proper value for H, we present the maximal number of selected paths for user requests in Fig. 8c. In SUB-B4 and B4, the maximal path numbers are 3 and 8, respectively. Setting H to 5 and 10 in SUB-B4 and B4 would be good to balance the service profit and time overhead of OSA. We also evaluate the performance of Metis with different H and obtain the similar results.

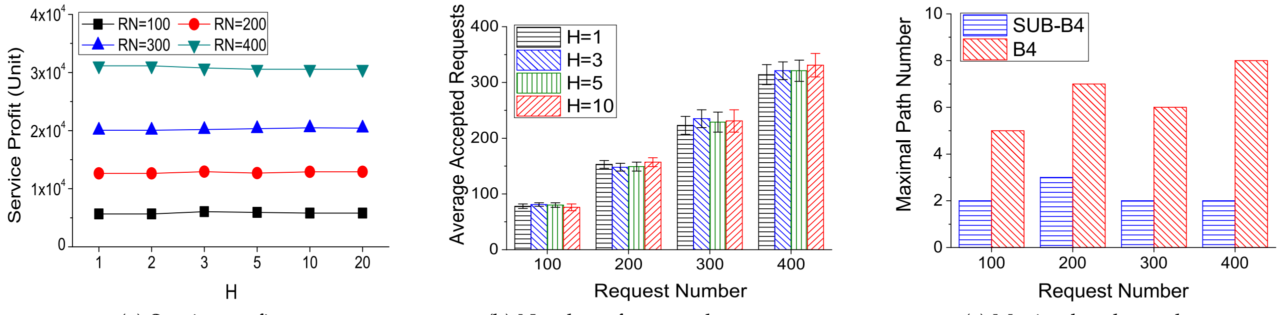
7.2.6 *Performance of OSA With Different Weight Settings*

To calculate the energy of each schedule, we use weight w to connect the current service profit and unallocated transmission capability. As the choice of w reflects the willingness of cloud providers to take risks and it is sensitive to the final service profit, we conduct evaluations on the B4 network to test the effect of different weights for OSA on making service profit. We set w to different values and present the changes of service profit and the number of accepted requests in Fig. 9.

In Fig. 9a, we can see that, OSA obtains the highest service profit when w is 20. When RN is 400 and w increases from 0 to 20, the service profit increases by 21.4 percent. However, it drops quickly when w is further increased from 20 to 500. This is because a proper w encourages OSA to accept more user requests while ensuring a low risk of not paying off the cost of purchasing additional bandwidth. If w is too large, OSA tends to keep more bandwidth unallocated to achieve a high "energy" by declining many user requests.

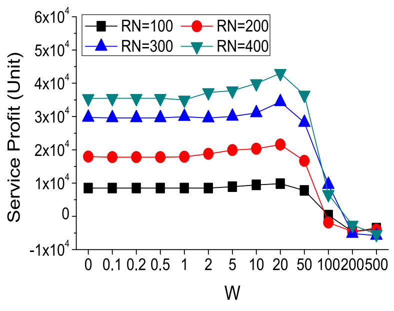
The number of accepted requests in Fig. 9b has a trend similar to that of Fig. 9a. When RN is 400 and w is 20, OSA accepts 321 user requests, which is much higher than that of $w = 0$ and $w = 500$. This demonstrates the significance of setting a proper weight w for OSA to make more service profit.

In Fig. 9c, we present the changes of average link utilization with different w. When w is increased from 0 and 20,

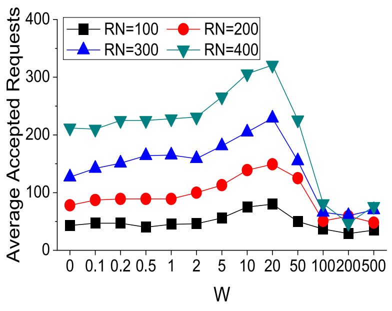


(a) Service profit (b) Number of accepted requests (c) Maximal path number

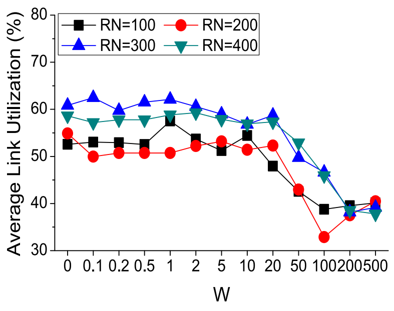
Fig. 8. Performance of OSA with different H.



(a) Service profit



(b) Number of accepted requests



(c) Link utilization

Fig. 9. Performance of OSA with different w.

the link utilization changes a little. It decreases a lot from $w = 20$ to 500, as OSA declines many user requests when w is too large.

8 Deployment Discussion

In this section, we discuss some issues to consider for our solutions to be used in a practical system.

8.1 Solution Selection

To help cloud providers make more service profit in geo-distributed cloud, we propose two solutions to handle different user request submission formats, offline and online. For these two solutions, Metis is fit for the scenarios that user requests are submitted independently and simultaneously at the beginning of a scheduling period, while OSA is designed for the scenarios that users submit their requests dynamically. Considering that both submission schemes may co-exist in the same cloud, cloud providers could use them cooperatively according to their features.

8.2 Parameter Setting

To successfully use our solutions in practice, cloud providers need to properly set the parameters according to their service characteristics and network scales.

The Metis framework consists of three main components, RL-SPM Solver, BW Limiter and BL-SPM Solver. By alternately minimizing the service cost and maximizing the service revenue for a certain number of rounds, Metis outputs the final scheduling solutions for the requests. By adjusting the number of rounds, cloud providers can balance the service profit and time overhead. Cloud providers can also customize rules to limit bandwidth in BW Limiter according to their request patterns and their needs. Besides MAA and TAA, RL-SPM Solver and BL-SPM Solver can be implemented in other ways, thus cloud providers can adopt different methods to realize Metis.

OSA is an online scheduler for cloud providers to handle the user requests that arrive dynamically. It is derived from classic simulated annealing method. Besides the regular parameters in simulated annealing, we use a parameter w to balance the current service profit and the remaining transmission capability, as introduced in Section 6. The evaluation results in Section 7 show that it is important for cloud providers to set it right to make more service profit. As finding and exploring all possible routing paths in a network is time-consuming but profit-limited, cloud providers can explore some shortest paths to accelerate OSA. Setting parameters to proper values plays an important role in maximizing the service profit of cloud providers. Emerging auto-tuning methods can be used to find the optimal or near-optimal parameters [36].

9 Related Work

Traffic engineering for Inter-DC WANs is an attractive topic recently. Benefitting from the emerging software-defined networking, SWAN [6] and B4 [7], [37] are proposed to boost the utilization of Inter-DC WANs by scheduling traffic with centralized controllers. Tempus [38] is proposed to maximize the fraction of transfers delivered before their deadlines, which achieves fairness among all transfers. All of them are cost-unaware and deadline-agnostic solutions [39]. Although Amoeba [26] has addressed the deadline-agnostic problem and provides deadline guarantees for the accepted transfers, it does not consider the bandwidth cost. To maximize the service profit for cloud providers, we alternately minimize the bandwidth cost and maximize the service revenue for many rounds.

Economically scheduling transfers in Inter-DC WANs has also attracted much research attention to reduce the bandwidth cost. NetStitcher [17] and Postcard [18] adopt store-and-forward strategy to deliver data between data centers, where large data transfers are delayed and sent at non-peak hours to fully utilize the already-paid bandwidth and reduce the charge. However, they do not consider deadlines of large transfers, and require large storage in data centers to store data temporarily. EcoFlow [20] splits inter-datacenter flows into multiple paths to avoid the increases of charging volumes and bandwidth cost, which may introduce packet-level reordering problems and degrade the service performance. In contrast, our solution does not need any extra storage in data centers or packet-reordering. We alternately maximize the service revenue by accepting a subset of requests with high service values and minimize the service cost by delivering them over selected paths with low bandwidth prices.

Dynamic pricing is a common method to improve the cost efficiency of Inter-DC WANs. For example, Pretium [12] combines it with traffic engineering to maximize the social welfare, i.e., the total value generated to society (over all served requests) minus the operational cost. It requires cloud providers to modify the current pricing mechanism. Besides, in peak hours, it requires customers to make a concession to either lower their service performance or raise their delivery bids. Zheng *et al.* [21] propose a pricing method to maximize the service revenue of Inter-DC WANs. Requiring neither modification to the current pricing mechanism nor complicated negotiation process with customers, our solutions can be implemented in current systems to greatly increase the service profits for cloud providers.

10 Conclusion

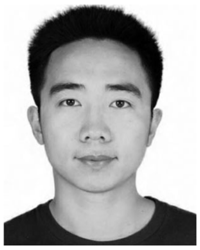
The popularity of cloud computing and the emergence of new clouds increase the competition between cloud providers. Maximizing service profit is critical for them to win the business, while it is challenging because of its NP-hardness. In this paper, we formulate the problem of service profit maximization and prove its NP-hardness. Then we propose two solutions, Metis and OSA, to handle user requests in different ways. Extensive evaluations demonstrate that, comparing with the existing solutions, our solutions increase the service profit by more than 60 percent.

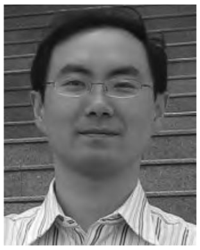
Acknowledgments

This work was supported by National Key R&D Program of China under Grants 2018YFB1800303, 2017YFB1010002, NSFC under Grant 61872211. The work of Minming Li was supported by NNSF of China under Grant 11771365, and sponsored by Project CityU 11200518 from Research Grants Council of HKSAR.

References

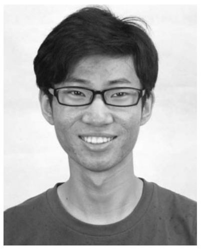
[1] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[2] A. Raghavan *et al.*, "Computational sprinting," in *Proc. IEEE Int. Symp. High-Perform. Comp Archit.*, 2012, pp. 1–12.
[3] N. Deng, C. Stewart, D. Gmach, M. Arlitt, and J. Kelley, "Adaptive green hosting," in *Proc. 9th Int. Conf. Auton. Comput.*, 2012, pp. 135–144.
[4] Z. Xu, C. Stewart, N. Deng, and X. Wang, "Blending on-demand and spot instances to lower costs for in-memory storage," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[5] C. Wang, B. Urgaonkar, G. Kesidis, A. Gupta, L. Y. Chen, and R. Birke, "Effective capacity modulation as an explicit control knob for public cloud profitability," *ACM Trans. Auton. Adaptive Syst.*, vol. 13, no. 1, pp. 1–25, 2018.
[6] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 15–26.
[7] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 3–14.
[8] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via yahoo! datasets," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2011, pp. 1620–1628.
[9] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, no. 4, 2015, Art. no. 63.
[10] X. Jin *et al.*, "Optimizing bulk transfers with software-defined optical WAN," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 87–100.
[11] I. Takanori, "Large-capacity optical transmission technologies supporting the optical submarine cable system," *NEC Tech. J.*, vol. 10, pp. 8–12, 2010.
[12] V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache, "Dynamic pricing and traffic engineering for timely inter-datacenter transfers," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 73–86.
[13] R. Nitin, "Bandwidth costs around the world," Accessed: Aug. 18, 2016. [Online]. Available: https://blog.cloudflare.com/bandwidth-costs-around-the-world/
[14] K. Hsieh *et al.*, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. 14th USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 629–647.
[15] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 323–336.
[16] W. Li, K. Li, D. Guo, G. Min, H. Qi, and J. Zhang, "Cost-minimizing bandwidth guarantee for inter-datacenter traffic," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 483–494, Secondquarter 2019.
[17] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with NetStitcher," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 74–85.
[18] Y. Feng, B. Li, and B. Li, "Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward," in *Proc. 32nd Int. Conf. Distrib. Comput. Syst. Workshops*, 2012, pp. 43–50.
[19] Y. Wang, S. Su, A. X. Liu, and Z. Zhang, "Multiple bulk data transfers scheduling among datacenters," *Comput. Netw.*, vol. 68, pp. 123–137, 2014.
[20] Y. Lin, H. Shen, and L. Chen, "EcoFlow: An economical and deadline-driven inter-datacenter video flow scheduling system," in *Proc. 23rd ACM Int. Conf. Multimedia*, 2015, pp. 1059–1062.
[21] Z. Zheng, R. Srikant, and G. Chen, "Pricing for revenue maximization in Inter-DataCenter networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 28–36.
[22] Z. Zhang, M. Zhang, A. G. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, "Optimizing cost and performance in online service provider networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, Art. no. 3.
[23] M. R. Gary and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," San Francisco, CA, USA: Freeman, vol. 174, 1979.
[24] Minimum-cost flow problem, Accessed: Apr. 18, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Minimum-cost_flow_problem
[25] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar, "Approximation algorithms for the unsplittable flow problem," *Algorithmica*, vol. 47, no. 1, pp. 53–78, 2007.
[26] H. Zhang *et al.*, "Guaranteeing deadlines for inter-datacenter transfers," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–14.
[27] Z. Yang *et al.*, "Achieving efficient routing in reconfigurable DCNs," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 1–30, 2019.
[28] P. Raghavan and C. D. Tompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.
[29] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis, "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems," *J. Comput. Syst. Sci.*, vol. 67, no. 3, pp. 473–496, 2003.
[30] P. Raghavan, "Probabilistic construction of deterministic algorithms: Approximating packing integer programs," *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 130–143, 1988.
[31] Aliyun, [Online]. Available: https://cn.aliyun.com
[32] Azure, [Online]. Available: https://azure.microsoft.com/
[33] AWS, [Online]. Available: https://aws.amazon.com/
[34] Gurobi Optimizer, [Online]. Available: http://www.gurobi.com/
[35] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quart. Appl. Math.*, vol. 27, no. 4, pp. 526–530, 1970.
[36] Z. Cao, V. Tarasov, S. Tiwari, and E. Zadok, "Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2018, pp. 893–907.
[37] C.-Y. Hong *et al.*, "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-DefinedWAN," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 74–87.
[38] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 515–526.
[39] W. Li, X. Zhou, K. Li, H. Qi, and D. Guo, "More peak, less differentiation: Towards a pricing-aware online control framework for inter-datacenter transfers," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2105–2110.


Zhenjie Yang received the BE degree on networking engineering from the Dalian University of Technology, Liaoning, China, in 2015. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include data center networking and cloud computing.


Yong Cui received the BE and PhD degrees both on computer science and engineering from Tsinghua University, China, respectively, in 1999 and 2004. He is currently a full professor at the Computer Science Department, Tsinghua University, China. He published more than 100 papers in the refereed conferences and journals with several Best Paper Awards. He co-authored 7 Internet standard documents (RFC) for his proposal on IPv6 technologies. His major research interests include mobile cloud computing and network architecture. He served or serves at the editorial boards on *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, and *IEEE Internet Computing*. He is currently a working group co-chair with IETF.

Xin Wang received the BS and MS degrees in telecommunications engineering and wireless communications engineering respectively from the Beijing University of Posts and Telecommunications, Beijing, China, and the PhD degree in electrical and computer engineering from Columbia University, New York. She is currently an associate professor with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, New York. Before joining Stony Brook, she was a member of technical staff in the area of mobile and wireless networking at Bell Labs Research, Lucent Technologies, New Jersey, and an assistant professor with the Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, New York. Her research interests include algorithm and protocol design in wireless networks and communications, mobile and distributed computing, as well as networked sensing and detection. She has served in executive committee and technical committee of numerous conferences and funding review panels, and serves as the associate editor of the *IEEE Transactions on Mobile Computing*. She achieved the NSF Career Award, in 2005, ONR Challenge Award, in 2010.

Minming Li received the PhD degree. He is currently an associate professor with the Department of Computer Science, City University of Hong Kong, China. He is the winner of City University of Hong Kong Teaching Excellence Award, in 2011–2012. His research interests include algorithms design and analysis, combinatorial optimization, scheduling, key management, and algorithmic game theory.


Yadong Liu received the BE degree in software engineering from the Beijing Institute of Technology, Beijing, China, in 2017. He is currently working toward the MS degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests are in the areas of data center networking.