Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective

Ruozhou Yu, Guoliang Xue, Xiang Zhang

***Abstract*—The emergence of the Internet-of-Things (IoT) has inspired numerous new applications. However, due to the limited resources in current IoT infrastructures and the stringent quality-of-service requirements of the applications, providing computing and communication supports for the applications is becoming increasingly difficult. In this paper, we consider IoT applications that receive continuous data streams from multiple sources in the network, and study joint application placement and data routing to support all data streams with both bandwidth and delay guarantees. We formulate the application provisioning problem both for a single application and for multiple applications, with both cases proved to be NP-hard. For the case with a single application, we propose a fully polynomial-time approximation scheme. For the multi-application scenario, if the applications can be parallelized among multiple distributed instances, we propose a fully polynomial-time approximation scheme; for general non-parallelizable applications, we propose a randomized algorithm and analyze its performance. Simulations show that the proposed algorithms greatly improve the quality-of-service of the IoT applications compared to the heuristics.**

***Keywords*—Internet-of-things, quality-of-service, service provisioning, fog computing, approximation algorithms**

I. Introduction

Designed to connect the digital world and the real world, the Internet-of-Things (IoT) has been recognized as one of the enabling technologies of the next era of computing. Numerous applications have been developed utilizing IoT functionalities, enabling advances in a number of areas including smart cities, smart health, connected cars, etc. It has been anticipated that the global IoT market will exceed $250B by 2020 [7].

One common type of IoT application is real-time processing applications, which process continuous data streams generated by IoT devices for pre-processing or analysis. These applications commonly have more stringent quality-of-service (QoS) requirements than traditional applications, including delay, throughput, etc., in order to ensure in-time delivery and analysis of real-time data and hence fast response to the users. An example is real-time sports analysis applications [13], [21], which analyze the status of live sport games, based on real-time data from cameras and/or other sensors.

Unfortunately, current IoT infrastructures are not built specifically for real-time processing applications. Current infrastructures use cloud computing as the underlying computing support. While cloud computing offers abundant and inexpensive computing power, it suffers from long end-to-end delay and high bandwidth usage, which greatly affect the performance of real-time processing applications. This situation is further aggravated by commonly used communication technologies in IoT, such as cellular networks and/or low-power wide-area networks (LPWANs), which offer only limited bandwidth for transmission.

Fog computing is one of the emerging technologies aiming to address these issues in current IoT. With fog nodes deployed near the IoT devices and end users, fog computing can reduce both the propagational delay and the bandwidth usage. However, ubiquitous fog node deployment is still unrealistic within the near future due to cost issues. Combined with the limited capacity of the IoT networks, this raises the problem of resource allocation in fog-enabled IoT. In particular, an infrastructure needs to allocate computing and network resources to support each application with proper QoS guarantees.

In this paper, we study this problem from a network perspective. Given a real-time processing IoT application, the infrastructure needs to decide both the fog node to host this application, and the channels along which the application's data streams will be transmitted. Furthermore, the channels must satisfy both the bandwidth demands of the application, and its delay requirement. We consider two problems: the Single-Application Provisioning (SAP) problem, and the Multi-Application Provisioning (MAP) problem, both proved to be NP-hard. For SAP, we propose a fully polynomial-time approximation scheme (FPTAS). For MAP, if the processing logic of each application can be parallelized among multiple distributed instances, we further propose another FPTAS; if the applications' processing cannot be parallelized, we propose a randomized algorithm with provable performance. To validate our algorithms, we have conducted extensive simulations, comparing our proposed algorithms to several heuristic solutions. It has been shown that our algorithms largely outperform the heuristics in terms of both bandwidth and delay.

Our contributions are summarized as follows:

- To the best of our knowledge, we are the first to study the problem of IoT application provisioning with both bandwidth and delay requirements.
- For the SAP problem, we propose an FPTAS.
- For the MAP problem, we propose an FPTAS when applications are parallelizable, and a randomized algorithm when each application can only be assigned to one host.
- We use extensive simulations to evaluate the performance of our algorithms against several heuristic approaches.

The rest of this paper is organized as follows. In Sec. II, we introduce related work. In Sec. III, we present our system model. In Secs. IV and V, we propose our results for SAP and MAP respectively. In Sec. VI, we present our performance evaluation results. In Sec. VII, we conclude this paper.

Yu, Xue and Zhang ({ruozhouy, xue, xzhan229}@asu.edu) are all with Arizona State University, Tempe, AZ 85287. This research was supported in part by NSF grants 1461886 and 1704092. The information reported here does not reflect the position or the policy of the funding agency.

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

II. Background and Related Work

A. Internet-of-Things and Fog Computing

While the concept of the "Internet-of-Things" can trace back to the last century[1], its power has barely been unleashed until recently, when several enabling technologies, including wireless networks, cloud computing and data science, have witnessed drastic advances. Since then, extensive efforts have been put into IoT-related areas, including computing architectures [2], communications [21], radio-frequency identification (RFID) [4], etc. A survey on IoT can be found in [17].

Fog computing has been regarded as one of the key technologies that enable IoT [2]. Extending from cloud computing, fog computing deploys geographically distributed fog nodes in the edge network, providing computing power closer to both the IoT devices and end users. Fog computing can improve the performance and energy efficiency in many IoT applications, including crowdsensing [1], smart cities [12], etc.

The limited resources in IoT and fog have urged efforts on new resource allocation methods. Zeng *et al.* [27] studied task scheduling and data placement to minimize I/O time, computing time and transmission delay in fog platforms. Many have studied workload offloading in edge/fog-cloud systems with different objectives, including power consumption [8], [25], delay minimization [8], [20], [22], quality-of-experience [25], etc. However, most of these do not consider the complex structure and limited capacity of the edge network; while Deng *et al.* [8] indeed considered network bandwidth constraints, they assumed that the transmission of each application's data will not interfere with each other, which does not capture the sharing nature of the IoT networks, and hence does not apply in many cases. Due to lack of existing work on network resource allocation in fog-enabled IoT, we study application provisioning from a network perspective, where we aim to guarantee the QoS of applications in terms of both transmission delay and bandwidth.

B. Network Service Provisioning

Stepping out of the IoT and fog computing domain, some related resource allocation problems have also been studied in different contexts, such as *virtual network/infrastructure embedding* (VNE/VIE) [5] and *service function chaining* (SFC) [16], [19]. The VNE/VIE problems aim to find an embedding of a virtual service topology onto the physical topology, which respects resource capacities in the substrate. The difference is that VIE allows virtual node consolidation while VNE does not. While these two problems can be viewed as a generalization of ours, they are harder to solve. To the best of our knowledge, there has yet been any non-trivial approximation ratio for VNE/VIE on general graphs. Assumptions on topologies and/or service models help in providing performance bounds [28], but are commonly too restrictive to handle the complex structures of the IoT networks.

SFC is another special case of the general VNE/VIE problems, where the virtual topology is restricted to line graphs. In this case, certain approximation bounds can be obtained, as shown by Rost *et al.* [19] and Kuo *et al.* [16]. In this paper, we consider a different service model than SFC, where the virtual topologies are star graphs. We also propose solutions with non-trivial performance guarantees.

[1]The term dates back to a talk by Kevin Ashton in 1999 [17].

III. System Model

A. Infrastructure Model

The IoT infrastructure is modeled as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the note set, and $\mathcal{E}$ is the link set; let $m = |\mathcal{E}|$. The node set consists of both *facility nodes* (general-purpose servers, fog-enabled switches/routers, etc.) and *network nodes* (switches/routers). We use $\mathcal{F} \subseteq \mathcal{V}$ to denote the set of facility nodes, and $\mathcal{N} \subseteq \mathcal{V}$ to denote the set of network nodes. Note that these two sets are not necessarily disjoint, as some network nodes may also have computing capabilities [6]. Each link $e \in \mathcal{E}$ has a capacity, denoted by $c_e > 0$, and a transmission delay, denoted by $d_e > 0$.

B. Application Model

An application receives continuous data from one or more data sources, and performs joint analysis of all received data. We assume each source generates data in a constant rate, *e.g.*, a video camera generating video footages. Given the application, the infrastructure needs to both find a facility node to host it, and establish transmission channels from each source to the host. Each application may require certain hardware resources, *e.g.*, video processing commonly requires strong GPU for efficient computation. Hence in many cases, only a subset of the facility nodes can host an application. The established channels need to satisfy at least two QoS requirements: 1) each source should receive bandwidth that meets its data generation rate, and 2) the transmission delay of each channel should be within the delay tolerance of the application.

Formally, an IoT application is denoted by a triple, $\Gamma = (\mathcal{S}, B, D)$, where $\mathcal{S} \subseteq \mathcal{V}$ denotes the set of data sources of Γ, $B : \mathcal{S} \mapsto \mathbb{R}^+$ denotes the corresponding data generation rate of each data source in $\mathcal{S}$ ($\mathbb{R}^+$ is the positive real number set), and $D > 0$ is the delay bound that must be enforced for transmission from each data source. Given an application Γ, we further use $\mathcal{F}_\Gamma \subseteq \mathcal{F}$ to denote its *candidate host set*, where each $v \in \mathcal{F}_\Gamma$ satisfies the hardware requirement of Γ.

C. Provisioning Model

As aforementioned, application provisioning involves both finding the host node and data routing. We make the following definitions to help our problem statement.

Definition 3.1 (Feasible path set): Given network G and an application Γ, let $v \in \mathcal{F}_\Gamma$ be a candidate host of Γ and $s \in \mathcal{S}$ be a data source of Γ, the *feasible path set* of Γ regarding v and s, denoted by $\mathcal{P}^\Gamma_{v,s}$, is defined as the subset of all (s, v)-paths in G such that for each path $p \in \mathcal{P}^\Gamma_{v,s}$,

$$\sum\nolimits_{e \in p} d_e \leq D. \tag{1}$$

We use $\mathcal{P}^\Gamma_v = \bigcup_{s \in \mathcal{S}} \mathcal{P}^\Gamma_{v,s}$ to denote the feasible path set from all data sources of Γ to candidate host v, and $\mathcal{P}^\Gamma = \bigcup_{v \in \mathcal{F}_\Gamma} \mathcal{P}^\Gamma_v$ the feasible path set towards all candidate hosts of Γ. □

Definition 3.2 (Bandwidth allocation): Let P be an arbitrary set of paths in G. A *bandwidth allocation* of P is defined as a mapping $\mathcal{L} : P \mapsto \mathbb{R}^*$ ($\mathbb{R}^*$ denotes the nonnegative real

number set), where $\mathcal{L}(p)$ denotes the bandwidth allocated on path p for any $p \in P$. We say that a bandwidth allocation $\mathcal{L}$ is *feasible*, iff for any link $e \in \mathcal{E}$,

$$\sum\nolimits_{p \in P: e \in p} \mathcal{L}(p) \leq c_e. \quad (2)$$

We use $b(P) = \sum_{p \in P} \mathcal{L}(p)$ to denote the *aggregate bandwidth* of $\mathcal{L}$ over path set P. □

With the above definitions, we are ready to define the expected outcome of an application provisioning decision.

Definition 3.3 (Provisioning scheme): Given network G and an application Γ, a *provisioning scheme* is defined as a triple $\Pi = (h, P_h^\Gamma, \mathcal{L}_h^\Gamma)$, where $h \in \mathcal{F}_\Gamma$ is a candidate host of Γ, $P_h^\Gamma \subseteq \mathcal{P}_h^\Gamma$ is a subset of feasible paths of Γ towards host h, and $\mathcal{L}_h^\Gamma$ is a *feasible bandwidth allocation* of P_h^Γ. We say that a provisioning scheme Π is *feasible*, iff for every data source $s \in \mathcal{S}$, the aggregate bandwidth $b(P_{h,s}^\Gamma) \geq B(s)$, where $P_{h,s}^\Gamma = P_h^\Gamma \cup \mathcal{P}_{h,s}^\Gamma$ is the subset of selected paths for s. □

For simplicity, we assume the processing results are consumed locally at the application host. However, it is trivial to add channels that transmit the results to other nodes into our model and solutions, and hence is omitted due to page limit.

IV. Single-Application Provisioning

A. Problem Statement and Computational Complexity

We first define the single-application problem:

Definition 4.1 (SAP): Given network G and application Γ, the **Single-Application Provisioning (SAP)** problem is to find a *feasible* provisioning scheme Π for Γ. Its optimization version, named **O-SAP**, is to find a provisioning scheme Π for Γ, such that for every data source $s \in \mathcal{S}$, its aggregate bandwidth satisfies $b(P_{h,s}^\Gamma) \geq \lambda \cdot B(s)$, and the *traffic scaling ratio* λ is maximized. □

Equivalently, the optimization version can also be interpreted as finding the minimum *congestion ratio*, defined as the minimum ratio between the flow and the capacity of any link, such that the demands of all data streams are fully satisfied. Clearly, if a SAP instance is *feasible*, the corresponding O-SAP instance has an optimal value $\lambda^* \geq 1$, and vice versa.

Theorem 4.1: Both SAP and O-SAP are NP-hard. □

Proof: Consider a special case of SAP where the application has only one data source s and one candidate host t. In this case, SAP becomes finding a set of (s,t)-paths and a bandwidth allocation that satisfy the bandwidth demand $B(s)$ and the delay bound D. This turns out to be the Multi-Path routing with Bandwidth and Delay constraints (MPBD) problem, which is NP-hard [18]. Hence SAP is NP-hard, and the NP-hardness of O-SAP follows. ■

B. An FPTAS to O-SAP

Based on Theorem 4.1, an FPTAS to O-SAP is the best result one can expect unless P=NP. Our algorithm is based on the decomposition of O-SAP into two subproblems: *Host Designation* (HD) that decides the host node of application Γ, and *Data Routing* (DR) that decides the routing paths and bandwidth from each data source to the host. The relationship between DR and O-SAP is stated in the following lemma.

Algorithm 1: Approximation Algorithm $A_{\text{O-SAP}}$

Input: Network G, application Γ
Output: Traffic scaling ratio λ, provisioning scheme Π

```
λ ← 0;
for each candidate host h ∈ F_Γ do
    (λ_h, P_h^Γ, L_h^Γ) ← A_DR(G, Γ, h);
    if λ_h > λ then
        λ ← λ_h, Π ← (h, P_h^Γ, L_h^Γ);
return (λ, Π).
```

Lemma 4.1: If the DR subproblem admits a polynomial-time a-approximation algorithm, so does O-SAP. □

Proof: We construct an a-approximation algorithm to O-SAP ($A_{\text{O-SAP}}$) out of an a-approximation algorithm to DR (A_{DR}), as shown in Algorithm 1. The algorithm iterates over all candidate hosts to find the best solution for the application, using the a-approximation A_{DR}. To prove Algorithm 1 is an a-approximation to O-SAP, let $\Pi^* = (h^*, P^*, \mathcal{L}^*)$ be an optimal solution to O-SAP with objective value λ^*. Then $(P^*, \mathcal{L}^*)$ is indeed a feasible solution of DR given host node h^*. Let $\lambda_{h^*}^*$ be the optimal DR solution with h^*, we have $\lambda^* \leq \lambda_{h^*}^*$. The DR solution picked in Algorithm 1 during iteration h^*, denoted by $(P_{h^*}^\Gamma, \mathcal{L}_{h^*}^\Gamma)$, has scaling ratio $\lambda_{h^*} \geq a\lambda_{h^*}^* \geq a\lambda^*$. This leads to $\lambda \geq \lambda_{h^*} \geq a\lambda^*$. The lemma follows. ■

It remains to solve the DR subproblem, which is still NP-hard due to the same argument as in the proof of Theorem 4.1. Yet, the DR subproblem turns out to be a special case of the Maximum Concurrent Flow (MCF) problem with delay bounds, which admits an FPTAS due to [3]. Details are omitted due to page limit. This, combined with Lemma 4.1, leads to our final theorem for O-SAP.

Theorem 4.2: O-SAP admits an FPTAS, as shown in Algorithm 1 combined with the FPTAS in [3]. □

V. Multi-Application Provisioning

In this section, we study a more general problem where multiple applications seek to share the IoT infrastructure.

A. Problem Statement and Computational Complexity

The multi-application problem is defined as follows:

Definition 5.1 (MAP): Given network G and an application set $\mathbf{\Gamma} = \{\Gamma_1, \ldots, \Gamma_K\}$, the **Multi-Application Provisioning (MAP)** problem is to find a set of *feasible* provisioning schemes $\mathbf{\Pi} = \{\Pi_1, \ldots, \Pi_K\}$, where $\Pi_k = (h_k, P_k, \mathcal{L}_k)$ is the provisioning scheme for Γ_k for $k = 1, \ldots, K$, such that the shared capacity constraint is satisfied for any link $e \in \mathcal{E}$:

$$\sum\nolimits_{k=1}^{K} \sum\nolimits_{p \in P_k} \mathcal{L}_k(p) \leq c_e.$$

Its optimization version, named **O-MAP**, is to find a set of provisioning schemes $\mathbf{\Pi}$ for $\mathbf{\Gamma}$, such that the minimum *traffic scaling ratio* λ of all applications, as defined in Definition 4.1, is maximized. We use $P_{k,s} = P_k \cap \mathcal{P}_{h_k,s}^{\Gamma_k}$ to denote the subset of selected paths for data source s of application Γ_k. □

Clearly MAP and O-MAP are both NP-hard, as they generalize SAP and O-SAP, respectively. We extend the concept of HD and DR to MAP, where HD (*Host Designation*) is to fix a host node for each application, and DR (*Data Routing*) is to fix routing paths and bandwidth for each data stream.

B. Problem Formulation

Though the hardness of O-MAP follows from that of O-SAP, in general the former is harder, as there are $O(|\mathcal{F}|^K)$ possible HD solutions in the worst case, instead of the linear number in O-SAP. This prevents us from iterating over all possible HD combinations as in the last section. Below, we first give an exact formulation of O-MAP. For simplicity, we use k to denote Γ_k if no ambiguity is introduced. We use $\mathcal{P} = \bigcup_{k=1}^{K} \mathcal{P}^k$ to denote the set of all feasible paths of all applications[2]. Define $x(k,v) \in \{0,1\}$ as the indicator of whether an application k is hosted on node $v \in \mathcal{F}_k$, $\mathcal{L}(p) \geq 0$ as the bandwidth allocation on $p \in \mathcal{P}$, and $\lambda \geq 0$ as the traffic scaling ratio. Then O-MAP is formulated as follows:

$$\max \quad \lambda \tag{3a}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}^k_{v,s}} \mathcal{L}(p) \geq B(s) \cdot \lambda \cdot x(k,v), \qquad \forall k, v, s; \tag{3b}$$

$$\sum_{v \in \mathcal{F}_k} x(k,v) = 1, \qquad \forall k; \tag{3c}$$

$$\sum_{p \in \mathcal{P}: e \in p} \mathcal{L}(p) \leq c_e, \qquad \forall e \in \mathcal{E}; \tag{3d}$$

$$x(k,v) \in \{0,1\}, \mathcal{L}(p), \lambda \geq 0, \qquad \forall k, v, p. \tag{3e}$$

Explanation: Constraint (3b) couples bandwidth allocation with the demands, the host designation, and the scaling ratio. Constraint (3c) ensures that each application is hosted on exactly one node. Constraint (3d) enforces link capacities.

Due to binary variables $x(k,v)$ and Constraint (3b), Program (3) is a Mixed Integer Quadratic Program (MIQP), which is generally hard to solve. Therefore, below we first look at a related problem where we relax some of the constraints.

C. Parallelizable Applications

We say an application is *parallelizable*, if its processing logic can be split over multiple instances, each processing a certain portion of data from *every* data source. This application model may be of interest in many contexts, *e.g.*, for stateless data fusion of multiple sensors [9] for which each instance can process an arbitrary portion of the incoming data as long as the same portion is received synchronously from every data source. The problem of provisioning multiple parallelizable applications relaxes the constraint in O-MAP that each application is hosted on exactly one host. We call this problem the **Parallelizable O-MAP** problem, or **PO-MAP** for short.

Define variable $y(k,v) = \lambda \tilde{x}(k,v)$ where $\tilde{x}(k,v) \in [0,1]$ relaxes $x(k,v)$ in Program (4). PO-MAP is formulated below.

$$\max \quad \lambda \tag{4a}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}^k_{v,s}} \mathcal{L}(p) \geq B(s) \cdot y(k,v), \qquad \forall k, v, s; \tag{4b}$$

$$\sum_{v \in \mathcal{F}_k} y(k,v) \geq \lambda, \qquad \forall k; \tag{4c}$$

$$\sum_{p \in \mathcal{P}: e \in p} \mathcal{L}(p) \leq c_e, \qquad \forall e \in \mathcal{E}; \tag{4d}$$

$$y(k,v), \mathcal{L}(p), \lambda \geq 0, \qquad \forall k, v, p. \tag{4e}$$

[2]W.l.o.g., if two applications have an overlapping feasible routing path, we still regard the same path for two different applications as two different paths.

Program (4) is a *linear program* (LP) (also called the *linear relaxation* of Program (3)). However, it still may have an exponential size due to the possibly exponential number of feasible paths in a graph. Therefore, we next propose an FPTAS to the PO-MAP problem.

D. An FPTAS to PO-MAP

Our FPTAS to PO-MAP extends the ones to MCF reported in [3], [10], [11]. However, PO-MAP is more difficult than the above, due to the need for (fractional) host designation. We first write the dual of Program (4), where we define $z(k,v,s) \geq 0$ as the dual variable of Constraint (4b) for $\forall k, v \in \mathcal{F}_k, s \in \mathcal{S}_k$, $\varphi(k) \geq 0$ as the dual variable of Constraint (4c) for $\forall k$, and $l(e)$ as the dual variable of Constraint (4d) for $\forall e \in \mathcal{E}$:

$$\max \quad \Delta(l) = \sum_{e \in \mathcal{E}} c_e l(e) \tag{5a}$$

$$\text{s.t.} \quad \sum_{e \in p} l(e) \geq z(k,v,s), \qquad \forall k, v, s, p \in \mathcal{P}^k_{v,s}; \tag{5b}$$

$$\sum_{s \in \mathcal{S}_k} B(s) z(k,v,s) \geq \varphi(k), \qquad \forall k, v; \tag{5c}$$

$$\sum_{k=1}^{K} \varphi(k) \geq 1; \tag{5d}$$

$$z(k,v,s), \varphi(k), l(e) \geq 0, \qquad \forall k, v, s, e. \tag{5e}$$

Since the primal and dual are intrinsically different from the above references, we provide our full analysis for completeness of this paper, starting from the observations below:

Lemma 5.1: Constraint (5b) is binding, *i.e.*, equality holds instead of inequality at optimality, for at least one combination of k, v, s, p, where $k = 1 \ldots K, v \in \mathcal{F}_k, s \in \mathcal{S}_k, p \in \mathcal{P}^k_{v,s}$. □

Lemma 5.2: Constraint (5d) is binding. □

Lemma 5.3: For $\forall k$, Constraint (5c) is binding for at least one candidate host $v \in \mathcal{F}_k$. □

Lemma 5.4: For $\forall k, \forall v \in \mathcal{F}_k, \forall s \in \mathcal{S}_k$, Constraint (5b) is binding for at least one feasible routing path $p \in \mathcal{P}^k_{v,s}$. □

Proof: Let ε be an arbitrarily small positive amount. If Lemma 5.1 is false, Constraint (5b) is not binding for every combination of k, v, s, p. Then we can reduce the value of $l(e)$ for an arbitrary e where $l(e) > 0$ by ε, and obtain a feasible dual solution with a strictly smaller objective value, contradicting our optimality assumption. If Lemma 5.2 is false, then we can reduce the value of $\varphi(k)$ for every k by ε. This will make every Constraint (5c) unbinding. Then we can reduce the value of $z(k,v,s)$ for every combination of k, v, s, which makes every Constraint (5b) to be unbinding, contradicting Lemma 5.1. If Lemma 5.3 is false for some k, then we can increase the value of $\varphi(k)$ by ε, which makes Constraint (5d) unbinding, contradicting Lemma 5.2. If Lemma 5.4 is false for some combination of k, v, s, then we can increase the value of $z(k,v,s)$ by ε, which makes Constraint (5c) unbinding for the corresponding k, contradicting Lemma 5.3. Therefore, we conclude that Lemmas 5.1–5.4 are all true. ■

Based on Lemmas 5.1–5.4, we have the following facts.

1) At optimality, $z(k,v,s) = \min_{p \in \mathcal{P}^k_{v,s}} \{\sum_{e \in p} l(e)\}$, *i.e.*, $z(k,v,s)$ equals the shortest feasible routing path length in $\mathcal{P}^k_{v,s}$ regarding length function $l : \mathcal{E} \mapsto \mathbb{R}^*$;

2) At optimality, $\varphi(k) = \min_{v\in\mathcal{F}_k}\{\sum_{s\in\mathcal{S}_k} B(s)z(k,v,s)\}$, *i.e.*, $\varphi(k)$ equals the minimum (over all possible candidate hosts $v \in \mathcal{F}_k$) weighted (by $B(s)$) sum (over all sources $s \in \mathcal{S}_k$) of shortest feasible routing path lengths in $\mathcal{P}^k$ regarding length function l.

Let $\zeta_{k,v,s}(l) = \min_{p\in\mathcal{P}^k_{v,s}}\{\sum_{e\in p} l(e)\}$ be the shortest feasible path length in $\mathcal{P}^k_{v,s}$ regarding length function l, and $\psi_k(l) = \min_{v\in\mathcal{F}_k}\{\sum_{s\in\mathcal{S}_k} B(s)\zeta_{k,v,s}(l)\}$ be the minimum weighted sum of shortest path lengths of all data sources of k over any candidate host v. Further define $\alpha(l) = \sum_{k=1}^{K}\psi_k(l)$. Then, Program (5) is equivalent to $\min_{l\geq 0}\Delta(l)/\alpha(l)$, *i.e.*, finding a length function $l : \mathcal{E} \mapsto \mathbb{R}^*$ minimizing $\Delta(l)/\alpha(l)$.

Algorithm 2: Approximation Scheme $A_{\text{PO-MAP}}$

Input: Network G, application set $\mathbf{\Gamma}$, tolerance ω
Output: Scaling ratio λ, host designations $\{y(k,v)\}_{k,v}$, path sets $\{P^k_{v,s}\}_{k,v,s}$, bandwidth allocation $\mathcal{L}$

1 Initialize $\epsilon = \omega' = \frac{\omega}{4}$, $\gamma = \left(\frac{1+\epsilon(1+\omega')}{m}\right)^{1+\frac{1}{\epsilon(1+\omega')}}$, $l(e) = \frac{\gamma}{c_e}$ for $\forall e \in \mathcal{E}$, $P^k_{v,s} = \emptyset$ for $\forall k, v, s$, $\mathcal{L} = \emptyset$;
2 $\rho \leftarrow 0$;
3 **while** $\Delta(l) < 1$ **do** // phase
4 $\quad\rho \leftarrow \rho + 1$;
5 $\quad$**for** $k = 1 \dots K$ **do** // iteration
6 $\quad\quad\eta \leftarrow 1.0$;
7 $\quad\quad$**while** $\eta > 0$ **do** // step
8 $\quad\quad\quad(\tilde{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{v}, \tilde{\eta}) \leftarrow \text{PrimUpdt}(G, \mathbf{\Gamma}, k, l, \omega')$;
9 $\quad\quad\quad$**if** $\tilde{\eta} > \eta$ **then**
10 $\quad\quad\quad\quad\boldsymbol{\phi} \leftarrow \eta\boldsymbol{\phi}/\tilde{\eta}$; $\tilde{\eta} \leftarrow \eta$;
11 $\quad\quad\quad y(k,v) \leftarrow y(k,v) + \tilde{\eta}$; $\eta \leftarrow \eta - \tilde{\eta}$;
12 $\quad\quad\quad$**for** $s \in \mathcal{S}_k$ **do**
13 $\quad\quad\quad\quad P^k_{v,s} \leftarrow P^k_{v,s} \cup \{\tilde{p}_s\}$;
14 $\quad\quad\quad\quad\mathcal{L}(\tilde{p}_s) \leftarrow \mathcal{L}(\tilde{p}_s) + \phi_s$;
15 $\quad\quad\quad l(e) \leftarrow l(e)(1 + \epsilon\phi_e/c_e)$ for $\forall e \in \mathcal{E}_{\tilde{\boldsymbol{p}}}$, where $\mathcal{E}_{\tilde{\boldsymbol{p}}} = \bigcup_{s\in\mathcal{S}_k}\tilde{p}_s$, and $\phi_e = \sum_{s\in\mathcal{S}_k: e\in\tilde{p}_s}\phi_s$;
16 Scale $\mathcal{L}$ after phase $\rho - 1$ by $1/\log_{1+\epsilon} 1/\gamma$;
17 $\lambda \leftarrow (\rho - 1)/\log_{1+\epsilon} 1/\gamma$;
18 **return** $(\lambda, \{y(k,v)\}_{k,v}, \{P^k_{v,s}\}_{k,v,s}, \mathcal{L})$.

Algorithm 3: Algorithm PrimUpdt$(G, \mathbf{\Gamma}, k, l, \omega')$

Input: Network G, application set $\mathbf{\Gamma}$, index k, length function l, tolerance ω'
Output: Paths $\tilde{\boldsymbol{p}} = (\tilde{p}_s)^{\mathrm{T}}_{s\in\mathcal{S}_k}$, bandwidth $\boldsymbol{\phi} = (\phi_s)^{\mathrm{T}}_{s\in\mathcal{S}_k}$, selected node $\tilde{v}$, fraction of flow $\tilde{\eta}$

// path computation
for $\forall v \in \mathcal{F}_k$ **do**
$\quad$**for** $\forall s \in \mathcal{S}_k$ **do**
$\quad\quad\tilde{p}_{v,s} \leftarrow \arg\min_{p\in\mathcal{P}^k_{v,s}}\{\sum_{e\in p} l(e)\}$;
$\tilde{v} \leftarrow \arg\min_{v\in\mathcal{F}_k}\{\sum_{s\in\mathcal{S}_k} B(s)\zeta_{k,v,s}(l)\}$;
$\tilde{p}_s \leftarrow \tilde{p}_{\tilde{v},s}$ for $\forall s \in \mathcal{S}_k$;
// bandwidth allocation
$\Upsilon(e) \leftarrow 0$ for $\forall e \in \mathcal{E}$;
for $\forall s \in \mathcal{S}_k$ **do**
$\quad$**for** $\forall e \in \tilde{p}_s$ **do**
$\quad\quad\Upsilon(e) \leftarrow \Upsilon(e) + B_k(s)$;
$\Upsilon_{\max} \leftarrow \max_{e\in\mathcal{E}}\{\Upsilon(e)/c_e\}$;
$\phi_s \leftarrow B_k(s)/\Upsilon_{\max}$;
return $(\tilde{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{v}, \tilde{\eta})$.

Our FPTAS to PO-MAP is presented in Algorithm 2. A bold symbol denotes a vector of normal symbols hereafter. In the process, the algorithm keeps track of both a primal solution, denoted by variables $(\boldsymbol{y}, \mathcal{L})$ (note that λ can be computed based on $\mathcal{L}$), and a dual solution, denoted by the length function l (note that both variables $\boldsymbol{z}$ and $\boldsymbol{\varphi}$ can be computed based on l). Both solutions will be gradually updated. Initially, each link e's dual length is initialized to γ/c_e. The algorithm runs in *phases* (Lines 3–15), in each phase going through an *iteration* for each application k (Lines 5–15). In each iteration, the algorithm tries to push exactly $B_k(s)$ amount of flow for each data source s of application k. This is done in *steps* (Lines 7–15), where in each step, we push the same fraction of flow ($\tilde{\eta}$) to the same candidate host ($\tilde{v}$) from all data sources. This ensures that when we update the primal solution, the increment in variable $y(k,v)$ is proportional to the flow pushed to v from any data source $s \in \mathcal{S}_k$, thus satisfying both Constraints (4b) and (4c). This is achieved by first calling the PrimUpdt subroutine to get a feasible primal update, denoted by $(\tilde{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{v}, \tilde{\eta})$, and then updating the primal solution in Lines 9–14. After primal update, the algorithm then updates the dual lengths $l(e)$ based on the bandwidth ϕ_e pushed along each link e, in Line 15. It stops when $\Delta(l) \geq 1$, after which it then scales the obtained flows to enforce the link capacity constraints in Lines 16–17.

A key building block is the PrimUpdt subroutine, which produces a primal update for application k that will be incorporated into the current primal solution. Its algorithm is shown in Algorithm 3. It starts from finding the dual-shortest feasible path from every data source $s \in \mathcal{S}_k$ to every candidate host $v \in \mathcal{F}_k$, denoted as $\tilde{p}_{v,s}$. The candidate host $\tilde{v}$ corresponding to the minimum value $\psi_k(l)$ is picked, along with the corresponding paths to $\tilde{v}$, denoted as $\tilde{\boldsymbol{p}}$. Next, it derives a bandwidth allocation, such that 1) each data source s's bandwidth (ϕ_s) is proportional to its demand $B_k(s)$, 2) total bandwidth on every link e does not exceed e's capacity c_e, and 3) the minimum ratio ($\tilde{\eta}$) between any source's bandwidth and its demand is maximized. This is done in Lines 6–11 of Algorithm 3. Node $\tilde{v}$, paths $\tilde{\boldsymbol{p}}$ and bandwidth allocation $\boldsymbol{\phi}$ are then returned along with the resulting scaling ratio $\tilde{\eta}$.

PrimUpdt relies on finding the dual-shortest feasible routing paths, as in Line 9. However, this task itself is non-trivial, as it is equivalent to the Delay Constrained Least Cost path (DCLC) problem, which itself is NP-hard. Nevertheless, there exist FPTASs for DCLC [26], which can output a $(1+\omega')$-approximation of the dual-shortest feasible path within time polynomial to the input size and $1/\omega'$. Combined with the selection of ϵ, ω' and γ, we can prove that such an approximation is sufficient for obtaining our desired performance guarantee. The performance of $A_{\text{PO-MAP}}$ is summarized in Theorem 5.1.

Theorem 5.1: Given G, $\mathbf{\Gamma}$, and $\omega \in (0,1)$, $A_{\text{PO-MAP}}$ (with Line 3 of the PrimUpdt subroutine replaced by a DCLC FPTAS) can compute a $(1-\omega)$-approximation of the optimal PO-MAP solution, within time polynomial to both the input size and $1/\omega$, and hence is an FPTAS to PO-MAP. □

Proof: We first prove the approximation ratio of $A_{\text{PO-MAP}}$, and then prove its time complexity.

Part I (Approximation Ratio): We first assume the optimal primal objective $\lambda^* \geq 1$; this assumption will be removed later on. Due to the strong duality of LP, the optimal dual objective Δ^* is equal to λ^*. Let (ρ, k, τ) denote step τ of

iteration k of phase ρ in the algorithm. Given a symbol used in the algorithm, $\nu \in \{l, \zeta_{k,v,s}, \psi_k, \alpha, \Delta, \phi_s, \tilde{v}, \tilde{p}_s\}_{k,v,s,e}$, we use $\nu^{\rho,k,\tau}$, $\nu^{\rho,k}$ and ν^{ρ} to denote the corresponding values in/after the corresponding step, iteration and phase, respectively. We also use ν to denote $\nu(l)$ if no ambiguity is introduced.

Based on the primal-dual updates, we have the following:

$$\begin{aligned}\Delta^{\rho,k,\tau} &= \sum_{e\in\mathcal{E}} c_e l^{\rho,k,\tau-1}(e) + \epsilon \sum_{s\in\mathcal{S}_k} \phi_s^{\rho,k,\tau} \sum_{e\in\tilde{p}_s^{\rho,k,\tau}} l^{\rho,k,\tau-1}(e) \\ &\le \Delta^{\rho,k,\tau-1} + \epsilon(1+\omega') \sum\nolimits_{s\in\mathcal{S}_k} \phi_s^{\rho,k,\tau} \zeta_{k,\tilde{v}^{\rho,k,\tau},s}^{\rho,k,\tau},\end{aligned}$$

due to that each path $\tilde{p}_s^{\rho,k,\tau}$ is a $(1+\omega')$-approximation of the dual-shortest feasible $(s, \tilde{v}^{\rho,k,\tau})$-path, and the dual-shortest feasible path lengths are non-decreasing during the algorithm.

As in each iteration k, we push exactly $B(s)$ flow for $\forall s \in \mathcal{S}_k$, we have the following by summing up for all steps:

$$\begin{aligned}\Delta^{\rho,k} &\le \Delta^{\rho,k-1} + \epsilon(1+\omega') \min_{v\in\mathcal{F}_k} \sum\nolimits_{s\in\mathcal{S}_k} B(s)\zeta_{k,v,s}^{\rho,k} \\ &\le \Delta^{\rho,k-1} + \epsilon(1+\omega')\psi_k^{\rho,k}.\end{aligned}$$

Summing up for all applications (iterations), we then have:

$$\Delta^{\rho} \le \Delta^{\rho-1} + \epsilon(1+\omega')\alpha^{\rho}.$$

Since we know that $\frac{\Delta^\rho}{\alpha^\rho} \ge \Delta^* \ge 1$, we further have:

$$\begin{aligned}\Delta^{\rho} &\le \frac{\Delta^{\rho-1}}{1-\frac{\epsilon(1+\omega')}{\Delta^*}} \le \frac{\Delta^0}{\left(1-\frac{\epsilon(1+\omega')}{\Delta^*}\right)^{\rho}} \\ &\le \frac{\Delta^0}{(1-\epsilon(1+\omega'))} \exp\left(\frac{(\rho-1)\epsilon(1+\omega')}{\Delta^*(1-\epsilon(1+\omega'))}\right),\end{aligned}$$

where the last inequality is due to that $(1+x) \le \exp(x)$.

The initial dual objective value is $\Delta^0 = m\gamma$ given the initial length function l. Let ρ^* be the last phase before the algorithm stops. We know that $\Delta^{\rho^*} \ge 1$ and $\Delta^{\rho^*-1} < 1$. Then we can bound the optimal dual objective value Δ^* as follows:

$$\Delta^* \le \frac{(\rho^*-1)\cdot\epsilon(1+\omega')}{(1-\epsilon(1+\omega'))\ln\frac{1-\epsilon(1+\omega')}{m\gamma}}.$$

To bound the optimal primal objective value λ^*, first observe that each primal update only increases the bandwidth on each link e by at most c_e. Therefore, when the flow through a link e increases by exactly c_e, its dual length $l(e)$ is increased by at least $(1+\epsilon)$ times, due to the dual update in Line 15. Now, as $\Delta^{\rho^*-1} < 1$, we have $l^{\rho^*-1}(e) < 1/c_e$ for $\forall e \in \mathcal{E}$. Therefore, the final flow after phase ρ^*-1 scaled by a factor of $1/\log_{1+\epsilon} 1/\gamma$ is strictly feasible. Since in each phase we push exactly $B_k(s)$ flow for each data stream, the scaling ratio after ρ^*-1 phases is exactly ρ^*-1. Scaled by $1/\log_{1+\epsilon} 1/\gamma$, the scaling ratio $\lambda = (\rho^*-1)/\log_{1+\epsilon} 1/\gamma$ is strictly feasible.

Based on these, the primal-dual ratio is bounded as follows:

$$\frac{\lambda}{\Delta^*} \ge \frac{(1-\epsilon(1+\omega'))\cdot\ln\frac{1-\epsilon(1+\omega')}{m\gamma}}{\epsilon(1+\omega')\cdot\log_{1+\epsilon}\frac{1}{\gamma}}.$$

Given our selection of ϵ, ω' and γ, we have $\frac{\lambda}{\Delta^*} \ge 1-\omega$.

It remains to remove our assumption that $\lambda^* \ge 1$. Based on [11], if we can obtain a pair of bounds $(\lambda_{\mathsf{LB}}, \lambda_{\mathsf{UB}})$ such that $\lambda^* \in [\lambda_{\mathsf{LB}}, \lambda_{\mathsf{UB}}]$, then we can guarantee $\lambda^* \ge 1$ by scaling all demands by $1/\lambda_{\mathsf{LB}}$. Following [10], we use a path-based method to find λ_{LB} and λ_{UB}. For each data stream (k,s), we use a binary search to find a *maximum-capacity feasible routing path* $\bar{p}_{v,s}^k$ to each candidate host $v \in \mathcal{F}_k$. Given v, the search sets a threshold β, and then finds a shortest (s,v)-path (*w.r.t.* delay) in G_β, a subgraph of G that has all links in $\{e : c_e < \beta\}$ pruned. If the path has delay no more than D_k, β is increased; otherwise it is decreased. Let $\bar{b}_{v,s}^k = \min_{e\in\bar{p}_{v,s}^k}\{c_e\}$ be the capacity of $\bar{p}_{v,s}^k$, and $\bar{\lambda}_v^k = \min_{s\in\mathcal{S}_k}\{\bar{b}_{v,s}^k/B_k(s)\}$. For each k, we then select candidate host $\bar{v}_k = \arg\max_{v\in\mathcal{F}_k}\{\bar{\lambda}_v^k\}$, and let $\bar{\lambda}_k = \bar{\lambda}_{\bar{v}_k}^k$. Then, our upper bound is $\lambda_{\mathsf{UB}} = m\min_k\{\bar{\lambda}_k\}$, as each flow can be decomposed into up to m paths, with no contention among each other. Let $S = \sum_k |\mathcal{S}_k|$ be the total number of data streams. A lower bound is $\lambda_{\mathsf{LB}} = \min_k\{\bar{\lambda}_k\}/S$, by scaling using the maximum number of competing flows.

Part II (Time Complexity): For simplicity, we define notation $O^*(f) = O(f\log^{O(1)}\mathbb{L})$, where f is a function of the input size $\mathbb{L}$. Based on [10], [11], the number of phases is bounded by $\rho^* \le \lceil\Delta^*\log_{1+\epsilon}\frac{1}{\gamma}\rceil = O^*(\frac{\Delta^*}{\omega^2})$, each with K iterations, and the total number of steps is bounded by $m\log_{1+\epsilon}\frac{1+\epsilon}{\gamma} = O^*(m\frac{\Delta^*}{\omega^2})$ plus the total number of iterations. Each step incurs one PrimUpdt call, which both finds (approximate) dual-shortest feasible paths for every (v,s) pair, and allocates bandwidth. According to Xue *et al.* [26], each path is found in $O^*(\frac{1}{\omega'}|\mathcal{V}|m)$ time. Bandwidth allocation in PrimUpdt takes $O(S|V|)$ time, as each path consists of at most $|V|-1$ links. Combining the above, the time complexity of $A_{\text{PO-MAP}}$ is given by $O^*(\frac{\Delta^*}{\omega^3}S|\mathcal{F}||\mathcal{V}|m(m+K))$.

To remove the dependency on Δ^*, we employ the demand scaling technique in [11]. If the algorithm does not stop after $\lceil 2\log_{1+\epsilon}\frac{1}{\gamma}\rceil$ phases, we know that $\Delta^* \ge 2$. We then double all demands, hence halving Δ^*, and then re-run Algorithm 2. Now, we have $\Delta^* \in [1, Sm]$ after the initial scaling in Part I. Hence at most $O(\log_2(Sm))$ demand scaling rounds are needed to bring Δ^* within $[1,2]$, each spending $O^*(\frac{1}{\omega^3}S|\mathcal{F}||\mathcal{V}|m(m+K))$ time. Omitting the logarithm terms, the final complexity is $O^*(\frac{1}{\omega^3}S|\mathcal{F}||\mathcal{V}|m(m+K))$ combined with the initial scaling in Part I. The theorem follows. ■

E. A Randomized Algorithm to O-MAP

Based on the FPTAS to PO-MAP, we further propose a randomized algorithm to O-MAP, as shown in Algorithm 4. It starts by solving PO-MAP using Algorithm 3. With the fractional solution, it then randomly selects a host $v \in \mathcal{F}_k$ with probability equal to $\tilde{y}(k,v)$ (normalized $y(k,v)$) for each application. After that, it solves the original O-MAP program with fixed hosts $\mathbf{v} = \{v_k\}_k$ to ensure solution feasibility. This turns out to be a trivial generalization of the DR subproblem of O-SAP, and hence can be solved using the FPTAS in [3].

The performance of Algorithm 4 is summarized below.

Theorem 5.2: Given G, $\mathbf{\Gamma}$, and ω, Algorithm 4 outputs a $\Omega\left(\frac{(1-\omega)^2\log\log m}{\log m}\right)$-approximation of the optimal O-MAP solution *with high probability*, within time polynomial to the input size and $1/\omega$. □

Proof: We again first analyze the performance of $A_{\text{O-MAP}}$, and then study its time complexity.

Part I (Approximation Ratio): Let $\lambda_{\text{PO-MAP}}$ be the objective value obtained by approximately solving PO-MAP (Line 1).

Algorithm 4: Randomized Algorithm $A_{\text{O-MAP}}$

Input: Network G, application set $\mathbf{\Gamma}$, tolerance ω
Output: Scaling ratio λ, host selections $\{v_k\}_{k,v}$, path sets $\{P_s^k\}_{k,s}$, bandwidth allocation $\mathcal{L}$

1 $(\lambda, \boldsymbol{y}, \boldsymbol{P}, \mathcal{L}) \leftarrow A_{\text{PO-MAP}}(G, \mathbf{\Gamma}, \omega)$;
2 **for** $k = 1$ *to* K **do**
3 $\quad \tilde{y}(k,v) \leftarrow y(k,v) / \sum_{v \in \mathcal{F}_k} y(k,v)$ for $\forall v \in \mathcal{F}_k$;
4 $\quad$ Select $v \in \mathcal{F}_k$ with probability $\tilde{y}(k,v)$ as v_k;
5 Solve O-MAP (Program (3)) with fixed HD solution $\mathbf{v} = \{v_k\}_k$, with accuracy ω;
6 **return** $(\lambda, \{v_k\}_k, \{P_s^k\}_{k,s}, \mathcal{L})$.

Since PO-MAP is the relaxed problem of O-MAP, and Algorithm 3 is an FPTAS, it follows that the optimal value of the O-MAP instance satisfies $\lambda_{\text{PO-MAP}} \geq (1-\omega)\lambda^*_{\text{O-MAP}}$. Let $\mathbf{v} = \{v_k\}_k$ be a possible HD solution obtained by independently rounding for each application k, and $\boldsymbol{P}_{\mathbf{v}} = \{P^k_{v_k,s}\}_{k,s}$ be the corresponding selected path sets given $\mathbf{v}$. We know that $(\mathbf{v}, \boldsymbol{P}_{\mathbf{v}}, \mathcal{J}_{\mathbf{v}})$, where $\mathcal{J}_{\mathbf{v}}$ is the scaled bandwidth allocation such that $\mathcal{J}_{\mathbf{v}}(p) = \mathcal{L}(p)/\tilde{y}(k, v_k)$ for $\forall p \in P^k_{v_k,s}$, is a solution to O-MAP that satisfies Constraints (3b) and (3c) with objective $\lambda = \lambda_{\text{PO-MAP}}$. However, now the solution $(\mathbf{v}, \boldsymbol{P}_{\mathbf{v}}, \mathcal{J}_{\mathbf{v}})$ may violate Constraint (3d) due to the scaled bandwidth allocation.

For each link $e \in \mathcal{E}$, define random variable $X_k(e)$ to be the amount of traffic of application k going through link e in $\mathcal{J}$. The total traffic is given by random variable $X(e) = \sum_k X_k(e)$, with expectation $\mathbf{E}\{X(e)\} \leq c_e$. Since each application is rounded independently, all random variables $X_k(e)$ are independent for a given e. Further define normalized variables $\widetilde{X}_k(e) = X_k(e)/c_e$, and $\widetilde{X}(e) = X(e)/c_e$ with expectation $\mathbf{E}\{\widetilde{X}(e)\} \leq 1$. Based on [15], [24], we apply the Chernoff bound with $1 + \delta = \frac{4 \log m}{\log \log m}$, and we have

$$\Pr\{\widetilde{X}(e) \geq (1+\delta)\} \leq 1/m^2. \tag{6}$$

In other words, the total traffic through e is at most $(1+\delta)$ times its capacity, with a probability of at least $1 - 1/m^2$. Taking the union bound over all links, the maximum capacity violation ratio, r_{vio}, is at most $\frac{4 \log m}{\log \log m}$ with a probability of at least $1 - 1/m$. Finally, we integrate the violation ratio r_{vio} into the objective: by dividing the bandwidth allocation on every path (and hence the objective value) by r_{vio}, the capacity constraints are satisfied. We then achieve a feasible solution to O-MAP with objective of $\Omega\left(\frac{\log \log m}{\log m}\right) \cdot \lambda_{\text{PO-MAP}}$, w.h.p.

The last step (Line 5), which achieves a $(1-\omega)$-approximation of the optimal O-MAP solution with fixed HD, is essential for obtaining good solutions. Although this brings an additional factor of $(1-\omega)$ to the theoretical bound of the algorithm, in general, an extra round of solving DR outputs much better solutions than directly using the rounded and scaled DR solutions obtained in solving PO-MAP. Combining the above with Theorem 5.1, the final ratio is achieved.

Part II (Time Complexity): The major complexity comes from solving PO-MAP (Line 1) and solving O-MAP with fixed HD (Line 5), both within $O^*(\frac{1}{\omega^3} S|\mathcal{F}||\mathcal{V}|m(m+K))$ time. ■

Note that the above bound is merely a worst-case guarantee. In practice, our algorithm can achieve much better solutions than its theoretical bound in general, as shown in our simulation experiments in the next section.

VI. Performance Evaluation

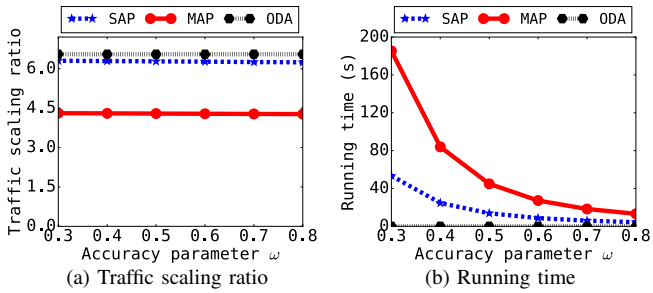
A. Experiment Settings

We used randomly generated topologies and applications for performance evaluation. The random topologies were generated using the Waxman model [23]. Each random topology has 20 nodes, where 20% of all nodes were randomly selected as facility nodes. Links were created using parameters α and β in the Waxman model, where $\alpha = \beta = 0.6$. Link capacities were randomly generated in $[10, 100]$ Mbps, and delays were randomly generated in $[1, 10]$ ms. In each experiment, we generated 5 IoT applications. An application had $[3, 10]$ data streams, each from a different data source. Application delay bounds were randomly generated in $[15, 25]$ ms. For each data stream, its bandwidth demand were randomly generated in $[1, 25]$ Mbps. We set accuracy $\omega = 0.5$ for the approximation algorithms. Above were the default parameters. We varied one control parameter in each set of experiments in order for evaluation under various scenarios.

SAP	Our FPTAS to O-SAP (Algorithm 1).
MAP	Our randomized algorithm to O-MAP (Algorithm 4).
ODA	*Optimal Delay-Agnostic* algorithm that attempts all combinations of application host designations, and for each combination solves an edge-flow multi-commodity flow LP that neglects applications' delay bounds; it yields an upper bound on the optimal delay-aware solution.
NS (HD)	*Nearest Selection* HD heuristic which for each application selects the host with minimum delay from all data sources.
RS (HD)	*Random Selection* HD heuristic where a random candidate host that is within the delay bound from every data source is selected for each application.
GH (DR)	*Greedy Heuristic* for DR that works in rounds where in each round, first the shortest path (w.r.t. delay) with positive capacity is found for every data stream, and then bandwidth allocation is done as in Lines 6–11 of Algorithm 3; it stops when any data stream's shortest path exceeds the application's delay bound.
DA (DR)	*Delay-Agnostic* optimal DR solution where an edge-flow MCF LP, which neglects application delay bounds, is solved; it yields an upper bound on DR.

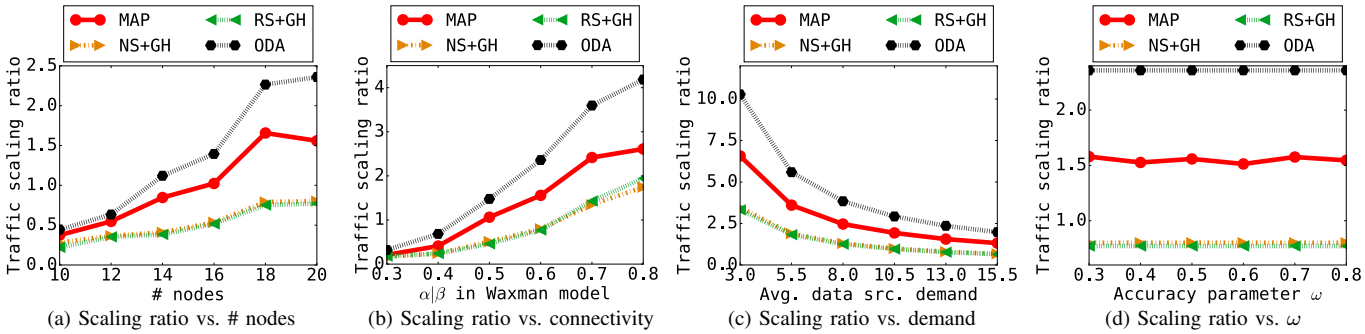
TABLE I: Implemented Algorithms

Our comparison algorithms are shown in Table I. Note that we proposed algorithms to solve HD and DR both jointly (**SAP**, **MAP**, **ODA**) and separately (**NS** and **RS** for HD, and **GH** and **DA** for DR). In the experiments, we further decomposed the entire **MAP** algorithm into its subroutines for solving HD (Lines 1–4) and DR (Line 5) respectively. Each combination of HD and DR algorithms was denoted by



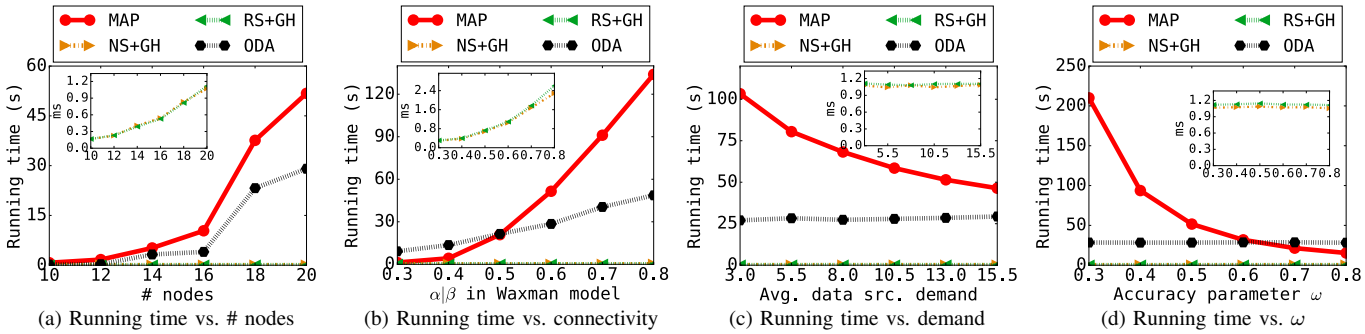
(a) Traffic scaling ratio (b) Running time

Fig. 1: Single-application experiments against accuracy ω.



(a) Scaling ratio vs. # nodes (b) Scaling ratio vs. connectivity (c) Scaling ratio vs. demand (d) Scaling ratio vs. ω

Fig. 2: Objective value against number of nodes, connectivity (α, β), bandwidth demand, and accuracy (ω).



(a) Running time vs. # nodes (b) Running time vs. connectivity (c) Running time vs. demand (d) Running time vs. ω

Fig. 3: Running time against number of nodes, connectivity (α, β), bandwidth demand, and accuracy (ω).

{HD}+{DR}, *e.g.*, **NS+GH** uses **NS** for HD and **GH** for DR.

We used the following metrics in performance evaluation. *Traffic scaling ratio* is the optimization objective λ, which is the minimum ratio between the allocated bandwidth and the demand of every data stream. *Maximum delay ratio* is the average ratio between the maximum transmission delay received by any application and its delay bound. *Running time* is the average running time of an algorithm in an experiment.

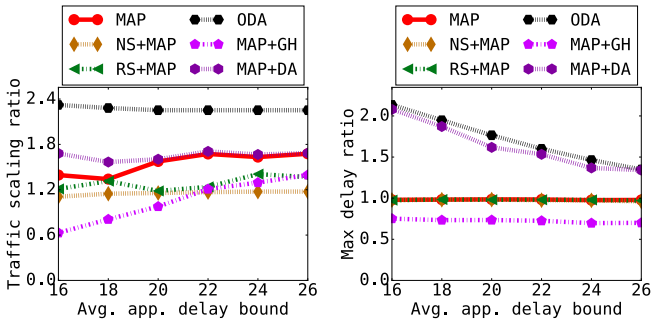
We developed a C++-based simulator which implements all the above algorithms. The Gurobi optimizer [14] was used to solve the LPs. Experiments were conducted on a Ubuntu Linux PC with Quad-Core 3.4GHz CPU and 16GB memory. Each experiment was repeated for 50 times under the same setting, and results were taken as the average over all runs.

B. Evaluation Results

1) Single-Application Scenario: Fig. 1 shows experiment results for provisioning a single application, where we changed the accuracy parameter ω from 0.3 to 0.8. Note that **ODA** is the theoretical upper bound as it neglects the application delay bounds. First, our algorithms achieve much better performance than the bounds proved in Theorem 5.1, as in Fig. 1(a). For instance, **SAP** achieves near-optimal solutions even when $\omega = 0.8$. Second, **SAP** performs significantly better than **MAP**, which matches their proved bounds. Third, the performance degradation of both **SAP** and **MAP** regarding increasing ω is minor. On the other hand, the running times of both **SAP** and **MAP** reduce drastically with increasing ω in Fig. 1(b). Finally, **SAP** is faster than **MAP**, first because it does not need to solve DR after selecting the host, and second because it results in fewer phases than **MAP** by quickly pushing flow towards each host alone, while **MAP** distributes flow among all hosts simultaneously. **ODA** is the fastest as it neglects the delay bounds, and hence becomes solving a linear number of polynomial-size LPs. Combining the above, we suggest using **SAP** with moderate or loose ω (*e.g.*, $\omega \geq 0.5$) for single-application provisioning or using as an online algorithm.

2) Multi-Application Scenario: Figs. 2 and 3 show experiment results for multi-application provisioning, with varying number of nodes, connectivity, average bandwidth demand, and accuracy ω. First, **MAP** outperforms both **RS+GH** and **NS+GH** in relatively large scales. Specifically, **MAP** can serve up to $2\times$ the traffic that can be served by **RS+GH** or **NS+GH** in a majority of the experiments. The cost of its superior performance is its higher running time. **MAP** is slower than **ODA** mainly because the latter does not consider application delay bounds. Also, with more applications, the running time of **MAP** will soon beat that of **ODA**, as the former is a polynomial-time algorithm, while the latter's time complexity is exponential to the number of applications. The shown trends basically match our intuition, *e.g.*, increased nodes or links lead to increased scaling ratios and running times, while larger bandwidth demands of the applications lead to smaller scaling ratios. In Fig. 3(c), the running time of **MAP** decreases with the scaling ratio. This matches their dependency shown in the proof of Theorem 5.1. However, if the objective value is very large, such a dependency will be removed by the demand scaling technique as illustrated in the proof. Finally, Figs. 2(d) and 3(d) show similar results as in Fig. 1 for **MAP**, where a looser accuracy parameter ω does not lead to noticeable performance loss, but greatly reduces its running time.

The above experiments show the superior performance of our joint algorithm **MAP**. In Figs. 4, we further analyze its performance for HD and DR separately, where we combined **MAP**'s subroutines with different heuristics respectively. Shown in Fig. 4(a), delay-aware DR solutions (**GH** and **MAP**'s DR subroutine) achieve better scaling ratios with larger delay bounds, while delay-agnostic solutions do not. Comparing different HD algorithms, **MAP**'s HD subroutine still achieves



(a) HD comparisons (b) DR comparisons

Fig. 4: HD and DR experiments with varying delay.

much better traffic scaling ratios than either **NS** or **RS**. Interestingly, **RS** outperforms **NS**, which is because **NS** can lead to congestion when a host is significantly closer to most data sources than the others. Comparing different DR algorithms, **MAP**'s DR subroutine also outperforms **GH**. On the other hand, since given fixed HD, the **DA** algorithm is optimal for delay-agnostic DR, we can see that **MAP**'s DR subroutine is near the optimal when delay bounds are large. In Fig. 4(b), with **MAP**'s DR subroutine, the maximum delay ratio is always bounded by but close to 1, meaning it utilizes paths of various delays, yet strictly sticks to the application delay bounds. **GH** also respects the delay bounds, yet it uses shorter paths, which leads to its low traffic scaling ratios in Fig. 4(a). Both **ODA** and **DA** are delay-agnostic, hence they can result in delays more than $2\times$ the bounds, violating application QoS requirements. In summary, the superior performance of the **MAP** algorithm comes from the advantages of both its HD and DR subroutines, compared to the heuristic algorithms.

VII. Conclusions

In this paper, we studied the provisioning of real-time processing applications in IoT. Two problems were defined and proved to be NP-hard: the single-application provisioning (SAP) problem, and the multi-application provisioning (MAP) problem. Both problems have yet been studied to the best of our knowledge. For SAP, we proposed an FPTAS. For MAP, we first proposed an FPTAS to the relaxed problem where each application is parallelizable among multiple instances, followed by a randomized algorithm with provable performance for the problem where applications are non-parallelizable based on our solution to the relaxed problem. We validated the advantages of our proposed algorithms over several heuristic solutions through extensive simulation experiments.

Acknowledgment

The authors would like to thank Ms. Rui Sun for her patient and meticulous help in improving the quality of this paper.

References

[1] S. Basudan, X. Lin, and K. Sankaranarayanan, "A Privacy-Preserving Vehicular Crowdsensing-Based Road Surface Condition Monitoring System Using Fog Computing," *IEEE Internet Things J.*, 4(3): 772–782, 2017.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *ACM MCC*, 2012.

[3] Z. Cao, P. Claisse, R. J. Essiambre, M. Kodialam, and T. V. Lakshman, "Optimizing Throughput in Optical Networks: The Joint Routing and Power Control Problem," in *IEEE INFOCOM*, 2015.

[4] H. Chen, G. Xue, and Z. Wang, "Efficient and Reliable Missing Tag Identification for Large-Scale RFID Systems With Unknown Tags," *IEEE Internet Things J.*, 4(3): 736–748, 2017.

[5] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in *IEEE INFOCOM*, 2009.

[6] Cisco, "Cisco IOx." URL: http://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html

[7] L. Columbus, "Internet Of Things Market To Reach $267B By 2020." URL: https://www.forbes.com/sites/louiscolumbus/2017/01/29/internet-of-things-market-to-reach-267b-by-2020/

[8] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal Workload Allocation in Fog-Cloud Computing Towards Balanced Delay and Power Consumption," *IEEE Internet Things J.*, 3(6): 1171–1181, 2016.

[9] W. Elmenreich, "Fusion of Continuous-valued Sensor Measurements Using Confidence-weighted Averaging," *J. Vib. Control*, 13(9-10): 1303–1312, 2007.

[10] L. K. Fleischer, "Approximating Fractional Multicommodity Flow Independent of the Number of Commodities," *SIAM J. Discret. Math.*, 13(4): 505–520, 2000.

[11] N. Garg and J. Konemann, "Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems," in *ACM FOCS*, 1998.

[12] A. Giordano, G. Spezzano, and A. Vinci, "Smart Agents and Fog Computing for Smart City Applications," in *Smart-CT*, 2016.

[13] M. Gowda, A. Dhekne, S. Shen, R. R. Choudhury, L. Yang, S. Golwalkar, and A. Essanian, "Bringing IoT to Sports Analytics," in *USENIX NSDI*, 2017.

[14] Gurobi, "Gurobi Optimizer." URL: http://www.gurobi.com/

[15] Jiaqi Zheng, Hong Xu, Xiaojun Zhu, Guihai Chen, and Yanhui Geng, "We've Got You Covered: Failure Recovery with Backup Tunnels in Traffic Engineering," in *IEEE ICNP*, 2016.

[16] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, "Service Chain Embedding with Maximum Flow in Software Defined Network and Application to the Next-Generation Cellular Network Architecture," in *IEEE INFOCOM*, 2017.

[17] S. Li, L. D. Xu, and S. Zhao, "The Internet of Things: A Survey," *Inf. Syst. Front.*, 17(2): 243–259, 2015.

[18] S. Misra, G. Xue, and D. Yang, "Polynomial Time Approximations for Multi-Path Routing with Bandwidth and Delay Constraints," in *IEEE INFOCOM*, 2009.

[19] M. Rost and S. Schmid, "Service Chain and Virtual Network Embeddings: Approximations Using Randomized Rounding," *arXiv:1604.02180*, 2016.

[20] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online Job Dispatching and Scheduling in Edge-Clouds," in *IEEE INFOCOM*, 2017.

[21] L. Toka, B. Lajtha, E. Hosszu, B. Formanek, D. Gehberger, and J. Tapolcai, "A Resource-Aware and Time-Critical IoT Framework," in *IEEE INFOCOM*, 2017.

[22] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *IEEE INFOCOM*, 2016.

[23] B. M. Waxman, "Routing of Multipoint Connections," *IEEE J. Sel. Areas Commun.*, 6(9): 1617–1622, 1988.

[24] S. Xiao, Y. Cui, X. Wang, Z. Yang, S. Yan, and L. Yang, "Traffic-aware Virtual Machine Migration in Topology-Adaptive DCN," in *IEEE ICNP*, 2016.

[25] Y. Xiao and M. Krunz, "QoE and Power Efficiency Tradeoff for Fog Computing Networks with Fog Node Cooperation," in *IEEE INFOCOM*, 2017.

[26] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing," *IEEE/ACM Trans. Netw.*, 16(3): 656–669, 2008.

[27] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System," *IEEE Trans. Comput.*, 65(12): 3702–3712, 2016.

[28] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang, "Towards Bandwidth Guarantee in Multi-Tenancy Cloud Computing Networks," in *IEEE ICNP*, 2012.