Provisioning QoS-aware and Robust Applications in Internet-of-Things: A Network Perspective

Ruozhou Yu, Guoliang Xue, Xiang Zhang

***Abstract*—The Internet-of-Things (IoT) has inspired numerous new applications ever since its invention. Nevertheless, its development and utilization have always been restricted by the limited resources in various application scenarios. In this paper, we study the problem of resource provisioning for real-time IoT applications, *i.e.*, applications that process concurrent data streams from data sources in the network. We investigate joint application placement and data routing to support IoT applications that have both quality-of-service and robustness requirements. We formulate four versions of the provisioning problem, spanning across two important classes of real-time applications (parallelizable and non-parallelizable), and two provisioning scenarios (single application and multiple applications). All versions are proved to be NP-hard. We propose fully polynomial-time approximation schemes for three of the four versions, and a randomized algorithm for the forth. Through simulation experiments, we analyze the impact of parallelizability and robustness on the provisioning performance, and show that our proposed algorithms can greatly improve the quality-of-service of the IoT applications.**

Keywords*—*IoT, QoS, robustness, placement and routing, 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 $\$250$B by 2020 [1].

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 on-time delivery and analysis of real-time data and hence fast response to the users. An example is real-time sports analysis applications [16], [26], which analyze the status of live sports 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, extending from our previous conference version [39]. 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. The channels must satisfy the QoS requirement of the application, including both the bandwidth demand of each data source, and the delay bound of the application.

Compared to [39], we additionally consider the robustness of data streams: how can the application survive an arbitrary network failure. We propose a technique where for any failure, the data loss incurred on each data stream is bounded by a portion of the total data. The robustness-aware problem generalizes the problem studied in [39]. Incorporating robustness is non-trivial, since it changes the problem formulation, rendering the algorithms in [39] infeasible in achieving a rigorous theoretical bound. We propose modified algorithms that present the same theoretical bounds. Our robustness formulation, combined with error correction coding [10], can achieve lossless data transmission and processing for applications, which is crucial in many critical scenarios such as emergency handling.

Two application types are considered. A *parallelizable application* is one that can be deployed as multiple instances, possibly with certain data restrictions. A *non-parallelizable application* is one that must be centrally implemented on one host. We further consider two provisioning scenarios: single-application provisioning, and multi-application provisioning. Combining the two types with the two scenarios, we have four versions of the provisioning problem. We formally define these problems, and prove that they are all NP-hard. We then propose fully polynomial-time approximation schemes (FPTASs) for three of them, and a randomized algorithm for the last one.

Our algorithms are based on the primal-dual FPTAS for Maximum Concurrent Flow (MCF) by Garg and Kone-

Yu (ryu5@ncsu.edu) is with North Carolina State University, Raleigh, NC 27606. Xue and Zhang ({xue, xzhan229}@asu.edu) are with Arizona State University, Tempe, AZ 85287. This research was supported in part by NSF grants 1461886, 1704092, and 1717197.

mann [14]. However, considering both application hosting and routing with bandwidth and delay constraints makes our problems NP-hard, much harder than MCF which has polynomial time optimal algorithms through linear programming. We novelly combine the MCF FPTAS with an existing FPTAS for the Delay Constrained Least Cost path (DCLC) problem [33], with additional techniques in handling the multi-source multi-destination nature of parallelizable applications. We use simulations to evaluate our algorithms against both the theoretical upper bound and several heuristic solutions. It is shown that our algorithms achieve close-to-optimal performance, and 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 QoS and robustness requirements.
- We formulate four versions of the provisioning problem, and proved all of them to be NP-hard.
- We propose FPTASs for three of the four versions, and a randomized algorithm for the last one.
- We use extensive simulations to evaluate the performance of our algorithms against both the theoretical bound and several practical heuristics.

The rest of this paper is organized as follows. In Sec. II, we introduce background and related work. In Sec. III, we present our system model. In Sec. IV, we formally define the four provisioning problems we study, and present their complexity result. In Secs. V and VI, we propose our algorithms for single application provisioning and multi-application provisioning, respectively. In Sec. VII, we present our performance evaluation results. In Sec. VIII, we conclude this paper.

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 [5], communications [26], radio-frequency identification (RFID) [7], etc. A survey on IoT can be found in [21].

Fog computing has been regarded as one of the key technologies that enable IoT [5]. 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 [3], smart cities [15], etc.

The limited resources in IoT and fog have urged efforts on new resource allocation methods. Zeng *et al.* [41] 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 [11], [29], delay minimization [11], [25], [27], quality-of-experience [29], etc. However, most of these do not consider the complex structure and limited capacity of the edge network; while Deng *et al.* [11] 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.

[1]The term dates back to a talk by Kevin Ashton in 1999 [21].

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) [8] and *service function chaining* (SFC) [20], [24]. 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 [44], 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.* [24] and Kuo *et al.* [20]. 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.

C. Robust Applications and Networks

Service robustness has long been studied in the literature. There are two common approaches for building and/or maintaining robust services. One is to provide *fault resilience* through redundancy, *i.e.*, provisioning redundant resources (computing, path, bandwidth, etc.) as backup to quickly recover a service when failure happens. For example, restoration-based routing, bandwidth allocation and network embedding have been studied in [18], [23], [30], [32]. Similar approaches have also been used for service, application and virtual machine backups [4], [19], [36], [37], [40]. Due to the need for redundancy, this approach commonly leads to excessively reserved backup resources that remain idle most of the time.

The second approach is to rather leverage the *fault tolerance* of the services themselves, and to minimize either the fault probability or the fault impact incurred on the services. Zhang *et al.* [42] modeled the fault probability of a virtual infrastructure based on the availability of all its physical

components, and sought to minimize this probability during the embedding. Acharya *et al.* [2], Zhang *et al.* [43] and Yallouz *et al.* [34] explored bounding the impact of a failure on the overall throughput of the system. This was termed *tunable survivability* in [34]. This idea was further leveraged in [35] and [38]. Compared to the redundancy-based approach, this approach results in much less resource consumption, yet providing reasonably good protection in practice. We therefore take this approach in our paper, specifically due to the already scarce resources in the IoT environment.

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 node set, and $\mathcal{E}$ is the link set. The node set consists of both *facility nodes* (servers, fog-enabled switches/routers, etc.) and *network nodes* (switches/routers). We use $\mathcal{F} \subseteq \mathcal{V}$ and $\mathcal{N} \subseteq \mathcal{V}$ to denote the sets of facility and network nodes, respectively. Note that $\mathcal{F}$ and $\mathcal{N}$ may not be disjoint, as some network nodes may also have computing capabilities [9]. 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 a set of data sources, and performs joint analysis of all data. We assume each source generates data in a constant rate, *e.g.*, a camera generating video footages. Given an application, we need to both find a facility node to host it, and establish transmission channels from each source to the host. An application may require certain hardware resources, *e.g.*, GPUs for efficient video processing. Hence usually, only a subset of facility nodes can host an application. The channels need to satisfy at least two QoS requirements: 1) each source should receive bandwidth that meets its data generation rate, and 2) each channel should satisfy the delay tolerance of the application.

Formally, an IoT application is denoted by a triple, $\Gamma = (\mathcal{S}, \mathbf{B}, D)$, where $\mathcal{S} \subseteq \mathcal{V}$ denotes the set of data sources of Γ, $\mathbf{B} = \{B_s \in \mathbb{R}^+ \,|\, s \in \mathcal{S}\}$ denotes the corresponding data generation rate of each data source in $\mathcal{S}$ ($\mathbb{R}^+$ is the set of positive real numbers), and $D > 0$ is the delay bound that must be enforced for transmission from each data source. Given a 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. Basic Provisioning Model

Application provisioning involves both finding the host node and data routing. Before defining the provisioning problems, we first make the following definitions.

Definition 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 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 set of nonnegative real numbers), 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 $e \in \mathcal{E}$,*

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

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

We consider two types of applications. A *non-parallelizable* application has no parallelism capability, hence its logic must be centrally implemented on one facility node. Contrarily, a *parallelizable application* can have its logic split over multiple instances, each processing a portion of the incoming data. However, implementing parallelism may have certain data splitting restrictions, such as data synchronization among sources. An example of a parallelizable application is stateless sensor data fusion [12], where each instance can process an arbitrary portion of incoming data as long as the same portion is received synchronously from every source. Below, we formalize the provisioning of these two types of applications.

Definition 3 (Provisioning scheme)**.** *Given network G and an application Γ, a* provisioning scheme *is defined as a triple $\Pi = (\mathbf{x}, P^\Gamma_\mathbf{x}, \mathcal{L}^\Gamma_\mathbf{x})$, where $\mathbf{x} = \{x_v \,|\, v \in \mathcal{F}\}$ is a decision variable vector with x_v denoting the fraction of data incoming to an instance of Γ on candidate host v, $P^\Gamma_\mathbf{x} \subseteq \mathcal{P}^\Gamma$ is a subset of feasible paths of Γ towards each candidate host $v \in \mathcal{F}$ with $x_v > 0$, and $\mathcal{L}^\Gamma_\mathbf{x}$ is a* feasible bandwidth allocation *of $P^\Gamma_\mathbf{x}$. We say that a provisioning scheme Π is* feasible *iff*

1) *$\sum_{v \in \mathcal{F}} x_v = 1$,*
2) *for $\forall v \in \mathcal{F}$, if Γ is non-parallelizable, then $x_v \in \{0, 1\}$, otherwise $x_v \in [0, 1]$, and*
3) *for $\forall s \in \mathcal{S}$ and $\forall v \in \mathcal{F}$, the aggregate bandwidth $b(P^\Gamma_{v,s}) \geq B_s \cdot x_v$, where $P^\Gamma_{v,s} = P^\Gamma_\mathbf{x} \cap \mathcal{P}^\Gamma_{v,s}$ is the subset of selected paths from s to v.* □

The last requirement of feasibility in Definition 3 ensures that the same portion of data generated by all data sources are received at the same instance, which can be used to enforce data synchronization for applications such as stateless sensor fusion. For simplicity, we assume that the processing results are consumed locally at the host(s). It is trivial to add channels that transmit the results, and hence is omitted for simplicity.

D. Robustness Model

Robustness means the ability to provide uninterrupted service when facing infrastructural failures. Some failures have inevitable effects in service quality, such as failures at data sources; others, however, can be avoided or alleviated by load and redundancy management. In this paper, we tackle failures that can be alleviated, including link and node failures.

As stated before, instead of the traditional "all-or-nothing" protection, we use a "soft" mechanism for robustness [38], [43], which ensures that an application incurs only bounded data loss due to any single failure in the network. Specifically, each source $s \in \mathcal{S}$ has a reliability parameter $r_s \in (0, 1]$, which denotes the *tolerable data loss ratio* for correct processing of its generated data. The idea is to ensure that the load is properly distributed such that the application incurs no more than $r_s \cdot B_s$ data loss from s due to any single failure. We call this approach *robustness through load balancing*. This can be coupled with a proper application- or network-level coding technique [10] to achieve loss-resistance in failure scenarios.

Given the above, we can extend Definition 3 to incorporate robustness. We start with the definition of a *link-robust provisioning scheme*, which protects against a single link failure:

Definition 4 (Link-robust provisioning scheme)**.** *Given network G and an application Γ, a* link-robust provisioning scheme *is a provisioning scheme Π for Γ that satisfies: for $\forall s \in \mathcal{S}$ and $\forall e \in \mathcal{E}$, $\sum_{p \in P^{\Gamma}_{\mathbf{x}}: e \in p} \mathcal{L}(p) \leq r_s \cdot B_s$.* □

The idea behind Definition 5 is that the data loss for data source s due to a single link failure is essentially bounded by the amount of data transmitted on all paths through link e. Similarly, we can define a *node-robust provisioning scheme*:

Definition 5 (Node-robust provisioning scheme)**.** *Given network G and an application Γ, a* node-robust provisioning scheme *is a provisioning scheme Π for Γ that satisfies: for $\forall s \in \mathcal{S}$ and $\forall u \in \mathcal{V} \setminus \{s\}$, $\sum_{p \in P^{\Gamma}_{\mathbf{x}}: u \in p} \mathcal{L}(p) \leq r_s \cdot B_s$.* □

Each source node s is excluded from protection, as failure of the source node will cause full blockage of data transmission, and hence cannot be alleviated through load balancing.

Before diving into the concrete problem definition, we want to highlight the different robustness capabilities of the two types of applications. Note that node-robustness is a generalization of link-robustness. For a non-parallelizable application, only link-robustness can be achieved, because its logic must centrally implemented, which becomes a single point of failure. In this case, protection over node failures can only be implemented through redundancy rather than load balancing, and hence is out of the scope of this paper. Meanwhile, a parallelizable application can achieve node-robustness, since it can balance load across instances. In the rest of this paper, we use the term "robust" to denote node-robustness for parallelizable applications, and link-robustness for non-parallelizable applications, depending on the context. Note that $r_s = 1$ means no protection for data source s, hence the problem we study generalizes the problem studied in [39].

E. Notations

We define some notations to facilitate illustration. $V = |\mathcal{V}|$ is the number of nodes. $E = |\mathcal{E}|$ is the number of links. $F_\Gamma = |\mathcal{F}_\Gamma|$ is the number of candidate hosts for application Γ, and $F = |\mathcal{F}|$ is the total number of facility nodes. $S_\Gamma = |\mathcal{S}_\Gamma|$ is the number of data sources for application Γ, and $S = \sum_{\Gamma \in \mathbf{\Gamma}} S_\Gamma$ is the total number of data sources for a set of applications $\mathbf{\Gamma}$. Notations used in our model are summarized in Table I.

TABLE I: Notations

Symbol	Meaning
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	IoT infrastructure with nodes and links
$\mathcal{F}, \mathcal{N} \subseteq \mathcal{V}$	Fog nodes, network nodes
c_e, d_e	Link capacity, link delay
$\Gamma = (\mathcal{S}, \mathbf{B}, D)$	Request: data sources, rates and delay bound
$B_s \in \mathbf{B}$	Data rate of data source $s \in \mathcal{S}$
$\mathcal{F}_\Gamma \subseteq \mathcal{F}$	Candidate host node set of Γ
$\mathcal{P}^{\Gamma}_{v,s}, \mathcal{P}^{\Gamma}_{v}, \mathcal{P}^{\Gamma}$	Feasible path sets: $v \in \mathcal{F}_\Gamma, s \in \mathcal{S}$
$\mathcal{L} : P \mapsto \mathbb{R}^*$	Bandwidth allocation over any path set P
$b(P)$	Aggregate bandwidth of $\mathcal{L}$ over path set P
$\Pi = (\mathbf{x}, P^{\Gamma}_{\mathbf{x}}, \mathcal{L}^{\Gamma}_{\mathbf{x}})$	Provisioning scheme: hosts, paths, allocation
$x_v \in \mathbf{x}$	Fraction of application hosted on $v \in \mathcal{F}_\Gamma$
r_s	Tolerable data loss ratio of source $s \in \mathcal{S}$
λ	Traffic scaling ratio (objective value), to be defined in Sec. IV

IV. Problem Statement and Complexity Analysis

We separately consider provisioning parallelizable and non-parallelizable applications. As stated before, we consider two scenarios. First, we consider provisioning a single application, which can be applied, *e.g.*, when processing online requests. Second, we consider the joint provisioning of multiple applications simultaneously. This can be useful both for batch provisioning of queued requests, and for provisioning multiple inter-related applications. Combining these, we arrive at four versions of the problem, which are formally defined below.

Definition 6 (SAP)**.** *Given network G and an application Γ, the **Single-Application Provisioning (SAP)** problem is to find a* feasible and robust *provisioning scheme Π for Γ.*

*Its optimization version, named **O-SAP**, is to find a robust provisioning scheme Π, such that for every data source $s \in \mathcal{S}$, its aggregate bandwidth satisfies $b(P^{\Gamma}_{\mathbf{x},s}) \geq \lambda \cdot B_s$, and the* traffic scaling ratio *λ is maximized.*

We use P-SAP/PO-SAP to denote the corresponding problem with a parallelizable application and node-robustness, and N-SAP/NO-SAP to denote the corresponding problem with a non-parallelizable application and link-robustness. □

Definition 7 (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 and robust *provisioning schemes $\mathbf{\Pi} = \{\Pi_1, \ldots, \Pi_K\}$, where $\Pi_k = (\mathbf{x}_k, P_k, \mathcal{L}_k)$ 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* robust *provisioning schemes $\mathbf{\Pi}$ for $\mathbf{\Gamma}$, such that the minimum* traffic scaling ratio *λ of all applications, as defined in Definition 6, is maximized.*

We use $P_{k,s} = P_k \cap \mathcal{P}^{\Gamma_k}_{\mathbf{x}_k,s}$ to denote the subset of selected paths for data source s of application Γ_k.

We use P-MAP/PO-MAP to denote the corresponding problem with parallelizable applications and node-robustness, and N-MAP/NO-MAP to denote the corresponding problem with non-parallelizable applications and link-robustness. □

Note that in the definitions, the inverse of the *traffic scaling ratio* λ is the congestion ratio $\chi = \frac{1}{\lambda}$, *i.e.*, the maximum load over capacity on any link. Maximizing λ is thus equivalent to minimizing congestion in the network. We choose to maximize λ for simplicity of our illustration. A traffic scaling ratio $\lambda \geq 1$ (congestion ratio $\chi \leq 1$) means that the corresponding decision problem instance is feasible, and vice versa.

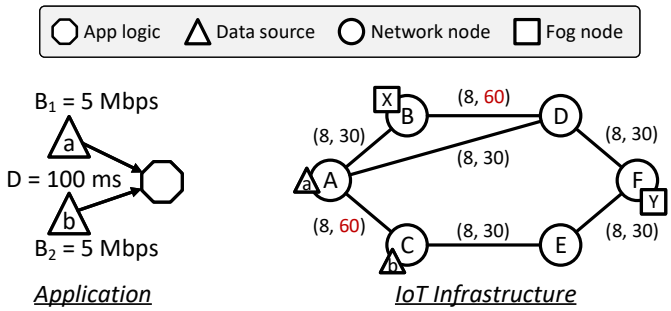


Fig. 1: SAP problem example with 2 data sources (a and b) and 2 candidate host nodes (X and Y). The values beside links are (capacity (Mbps), delay (ms)).

We show an example of SAP in Fig. 1 with 2 data sources (a and b), and 6 network nodes (A–F) and 2 facility nodes (X and Y). If the application is parallelizable, the goal here is to allocate demands from a and b to the facility nodes X and Y, such that each facility node processes the same amount of data from both a and b. If the application is non-parallelizable, the goal is to select either X or Y to process all data. We then need to route traffic to X and/or Y through one or more paths for each source. In doing so, we try to minimize congestion, *i.e.*, to maximize the traffic scaling ratio of both sources, meanwhile ensuring that all paths satisfy the delay bound 100 ms.

Theorem 1. *All problems defined above are NP-hard.* □

Proof: Since SAP problems are special cases of the corresponding MAP problems, it suffices to prove that P-SAP and N-SAP are NP-hard. Consider a special case of P-SAP or N-SAP where Γ has one data source s, one candidate host t, and no protection ($r_s = 1$). 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 [22]. Hence SAP is NP-hard, and the NP-hardness of the rest follows. ■

Solution overview. Due to the NP-hardness, we seek to approximate the optimal solutions. By definition, PO-MAP and NO-MAP are generalizations of PO-SAP and NO-SAP respectively. Below, we first show through a theorem that NO-SAP admits an FPTAS following a trivial extension of the FPTAS for an established flow problem. Both PO-SAP and PO-MAP admit another FPTAS, which, however, is a non-trivial extension of that FPTAS and requires involved steps in handling application hosting. We have yet been able to find an FPTAS for the most difficult NO-MAP problem due to the combinatorial hosting decisions, and hence we propose a randomized algorithm based on the FPTAS for PO-MAP. The existence of an FPTAS for NO-MAP remains open.

Algorithm 1: Approximation Algorithm $A_{\text{NO-SAP}}$

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

$\lambda \leftarrow 0$, $\mathbf{x} \leftarrow 0$;
for *each candidate host* $v \in \mathcal{F}_\Gamma$ **do**
  $(\lambda_v, P_v^\Gamma, \mathcal{L}_v^\Gamma) \leftarrow A_{\text{DR}}(G, \Gamma, v)$;
  if $\lambda_v > \lambda$ **then**
    $\lambda \leftarrow \lambda_v$, $\mathbf{x} \leftarrow 0$, $x_v \leftarrow 1$;
    $\Pi \leftarrow (\mathbf{x}, P_v^\Gamma, \mathcal{L}_v^\Gamma)$;
  end
end
return (λ, Π).

V. Single-Application Provisioning

We start with provisioning one application at a time. Due to the two application types, we have two versions of the problem (PO-SAP and NO-SAP). In this section, we propose an FPTAS for NO-SAP. We leave PO-SAP to Sec. VI, where we propose an FPTAS for both PO-SAP and PO-MAP. In the rest of this section, we omit the term "non-parallelizable".

Our algorithm to NO-SAP is based on the decomposition of NO-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. For simplicity, we extend this decomposition method throughout the rest of this paper, with HD denoting determination of the decision vector $\mathbf{x}$, and DR denoting the routing process, *i.e.*, determining $P_\mathbf{x}^\Gamma$ and $\mathcal{L}_\mathbf{x}^\Gamma$. For the NO-SAP problem, the relationship between this problem and its DR subproblem is stated in the following lemma.

Lemma 1. *If the DR subproblem admits a polynomial-time a-approximation algorithm, so does NO-SAP.* □

Proof: We construct an a-approximation algorithm to NO-SAP ($A_{\text{NO-SAP}}$) from an a-approximation algorithm to DR (A_{DR}), shown in Algorithm 1. The algorithm enumerates all candidate hosts to find the best one, using the a-approximation A_{DR}. To prove Algorithm 1 is an a-approximation to NO-SAP, let $\Pi^* = (\mathbf{x}^*, P^*, \mathcal{L}^*)$ be an optimal solution to NO-SAP with objective value λ^* and $x_{v^*}^* = 1$. Then $(P^*, \mathcal{L}^*)$ is indeed a feasible solution of DR given host node v^*. Let $\lambda_{v^*}^*$ be the optimal DR solution with v^*, we have $\lambda^* \leq \lambda_{v^*}^*$. The DR solution picked in Algorithm 1 during iteration v^*, denoted by $(P_{v^*}^\Gamma, \mathcal{L}_{v^*}^\Gamma)$, has scaling ratio $\lambda_{v^*} \geq a\lambda_{v^*}^* \geq a\lambda^*$. This leads to $\lambda \geq \lambda_{v^*} \geq a\lambda^*$. 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 1. Yet, the DR subproblem turns out to be a special case of the QoS-aware and Reliable Traffic Steering (QRTS) problem studied in [38]. Specifically, the DR subproblem is equivalent to having a policy routing requirement that contains no service function and having all traffic flows pointing to the same destination node in QRTS. Combining the FPTAS proposed in [38] with Lemma 1 leads to our final theorem for NO-SAP:

Theorem 2. *NO-SAP admits an FPTAS, as shown in Algorithm 1 combined with the FPTAS in [38].* □

VI. Multi-Application Provisioning

In this section, we study multiple applications sharing the IoT. We first propose an FPTAS for PO-MAP that also solves PO-SAP as a special case. Since we do not have an FPTAS for NO-MAP, we then propose a randomized algorithm for it.

A. Problem Formulation for PO-MAP

We first formulate PO-MAP. For simplicity, we use k to denote Γ_k. We use $\mathcal{P} = \bigcup_{k=1}^{K} \mathcal{P}^k$ to denote the set of all feasible paths of all applications[2]. We then use $\mathcal{P}_s^k \subseteq \mathcal{P}$ to denote all feasible paths for application k's source s. For consistency of notation, we define variables $x(k,v) \triangleq x_v^k$ as the fraction of application k hosted on candidate host $v \in \mathcal{F}_k$, $\mathcal{L}(p)$ as the bandwidth allocation on path $p \in \mathcal{P}$, and λ still as the traffic scaling ratio. NO-MAP is formulated as follows:

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

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

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

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

$$\sum_{p \in \mathcal{P}_s^k: u \in p \setminus \{s\}} \mathcal{L}(p) \leq r_s^k \cdot B_s^k, \quad \forall k, s, u \in \mathcal{V} \setminus \{s\}; \tag{3e}$$

$$x(k,v) \in [0,1], \mathcal{L}(p), \lambda \geq 0, \quad \forall k, v \in \mathcal{F}_k, p. \tag{3f}$$

Explanation: Constraint (3b) couples bandwidth allocation with the demands, host designation, and the scaling ratio. Constraint (3c) ensures that a feasible host designation. Constraint (3d) enforces link capacities. Constraint (3e) enforces *node-robustness*, such that the flow over each node $u \in \mathcal{V} \setminus \{s\}$ is bounded by $r_s^k \cdot B_s^k$ for each source s of each application k.

Program (3) seems like a Quadratic Program (QP) due to Constraint (3b). However, with a simple transformation shown below, it can be transformed into an equivalent Linear Program (LP). Define new variables $y(k,v) = \lambda \cdot x(k,v)$, we have

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

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

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

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

$$\sum_{p \in \mathcal{P}_s^k: u \in p} \mathcal{L}(p) \leq r_s^k \cdot B_s^k, \quad \forall k, s, u \in \mathcal{V} \setminus \{s\}; \tag{4e}$$

$$y(k,v), \mathcal{L}(p), \lambda \geq 0, \quad \forall k, v \in \mathcal{F}_k, p. \tag{4f}$$

Programs (3) and (4) are clearly equivalent. However, Program (4) may still have an exponential size due to the possibly exponential number of feasible paths, and hence cannot be solved directly as an LP. Hence, we next propose an FPTAS.

[2]W.l.o.g., we regard the same path for two applications as two different paths.

B. An FPTAS to PO-MAP

Our FPTAS to PO-MAP extends the ones to MCF reported in [6], [13], [14]. However, PO-MAP is more difficult than the above, due to the need for (fractional) host designation as well as the node-robustness constraint. 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 $\forall k$, $l(e)$ as the dual variable of Constraint (4d) for $\forall e \in \mathcal{E}$, and $\sigma(k,s,u)$ as the dual variable of Constraint (4e) for $\forall k, s \in \mathcal{S}_k, u \in \mathcal{V} \setminus \{s\}$:

$$\min \quad \Delta(l,\sigma) = \sum_{e \in \mathcal{E}} c_e l(e) + \sum_{k=1}^{K} \sum_{s \in \mathcal{S}_k} \sum_{u \in \mathcal{V}}^{u \neq s} r_s^k \cdot B_s^k \cdot \sigma(k,s,u) \tag{5a}$$

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

$$\sum_{s \in \mathcal{S}_k} B_s^k \cdot z(k,v,s) \geq \varphi(k), \quad \forall k, v; \tag{5c}$$

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

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

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

Lemma 3. *Constraint* (5d) *is binding.* □

Lemma 4. *For $\forall k$, Constraint* (5c) *is binding for at least one candidate host $v \in \mathcal{F}_k$.* □

Lemma 5. *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 $p \in \mathcal{P}_{v,s}^k$.* □

Proof: Let ε be an arbitrarily small positive amount. If Lemma 2 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 3 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 2. If Lemma 4 is false for some k, then we can increase the value of $\varphi(k)$ by ε, which makes Constraint (5d) unbinding, contradicting Lemma 3. If Lemma 5 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 4. Therefore, we conclude that Lemmas 2–5 are all true. ■

Based on Lemmas 2–5, we have the following facts.

1) At optimality, $z(k,v,s) = \min_{p \in \mathcal{P}_{v,s}^k} \{\sum_{e \in p} l(e) +$

$\sum_{u\in p\setminus\{s\}}\sigma(k,s,u)\}$, *i.e.*, $z(k,v,s)$ equals the shortest feasible routing path length in $\mathcal{P}^k_{v,s}$ regarding length functions $l(\cdot)$ for links and $\sigma(k,s,\cdot)$ for nodes;

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

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

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

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

1 Initialize $\epsilon = \omega' = \frac{\omega}{4}$, $\gamma = \left(\frac{1+\epsilon(1+\omega')}{E+(V-1)S}\right)^{1+\frac{1}{\epsilon(1+\omega')}}$, $l(e) = \frac{\gamma}{c_e}$ for $\forall e\in\mathcal{E}$, $\sigma(k,s,u) = \frac{\gamma}{r^k_s B^k_s}$ for $\forall k,s,u\in\mathcal{V}\setminus\{s\}$, $P^k_{v,s} = \emptyset$ for $\forall k,v,s$, $\mathcal{L} = \emptyset$;
2 $\rho\leftarrow 0$;
3 **while** $\Delta(l,\sigma) < 1$ **do** // phase
4 $\rho\leftarrow\rho+1$;
5 **for** $k = 1\dots K$ **do** // iteration
6 $\eta\leftarrow 1.0$;
7 **while** $\eta > 0$ **do** // step
8 $(\tilde{\boldsymbol{p}},\boldsymbol{\phi},\tilde{v},\tilde{\eta})\leftarrow$ PrimUpdt$(G,\mathbf{\Gamma},k,l,\sigma,\omega')$;
9 **if** $\tilde{\eta} > \eta$ **then**
10 $\boldsymbol{\phi}\leftarrow\eta\boldsymbol{\phi}/\tilde{\eta}$; $\tilde{\eta}\leftarrow\eta$;
11 **end**
12 $y(k,v)\leftarrow y(k,v)+\tilde{\eta}$; $\eta\leftarrow\eta-\tilde{\eta}$;
13 **for** $s\in\mathcal{S}_k$ **do**
14 $P^k_{v,s}\leftarrow P^k_{v,s}\cup\{\tilde{p}_s\}$;
15 $\mathcal{L}(\tilde{p}_s)\leftarrow\mathcal{L}(\tilde{p}_s)+\phi_s$;
16 **end**
17 $l(e)\leftarrow l(e)(1+\frac{\epsilon\phi_e}{c_e})$ 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 $\phi_e = \sum_{s\in\mathcal{S}_k:e\in\tilde{p}_s}\phi_s$;
18 $\sigma(k,s,u)\leftarrow\sigma(k,s,u)(1+\frac{\epsilon\phi_u}{r^k_s B^k_s})$ for $\forall s\in\mathcal{S}_k, u\in\mathcal{V}_{\tilde{\boldsymbol{p}}}\setminus\{s\}$, where $\mathcal{V}_{\tilde{\boldsymbol{p}}} = \bigcup_{s\in\mathcal{S}_k}\{v\in\tilde{p}_s\}$, and $\phi_u = \sum_{s\in\mathcal{S}_k:u\in\tilde{p}_s\setminus\{s\}}\phi_s$;
19 **end**
20 **end**
21 **end**
22 Scale $\mathcal{L}$ and $\mathbf{y}$ after phase $\rho-1$ by $1/\log_{1+\epsilon}1/\gamma$;
23 $\lambda\leftarrow(\rho-1)/\log_{1+\epsilon}1/\gamma$;
24 **return** $(\lambda,\mathbf{y},\mathbf{P},\mathcal{L})$.

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 functions (l,σ) (note that both variables $\boldsymbol{z}$ and $\boldsymbol{\varphi}$ can be computed based on l and σ). Both solutions will be gradually updated. Initially, each link e's dual length is initialized to γ/c_e, and each node u's dual length (regarding application k's data source s) is initialized to $\gamma/r^k_s B^k_s$. The algorithm runs in *phases* (Lines 3–21), in each phase going through an *iteration* for each application k (Lines 5–20). 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 7–19), 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 9–16. 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 17; it also updates $\sigma(k,s,u)$ based on the bandwidth at each node, in Line 18. It stops when $\Delta(l,\sigma)\geq 1$, after which it scales the flows to enforce the link capacity constraints in Lines 22–23.

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

Input: Network G, application set $\mathbf{\Gamma}$, index k, length functions l and σ, 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
1 **for** $\forall v\in\mathcal{F}_k$ **do**
2 **for** $\forall s\in\mathcal{S}_k$ **do**
3 $\tilde{p}_{v,s}\leftarrow\arg\min_{p\in\mathcal{P}^k_{v,s}}\{\sum_{e\in p} l(e) + \sum_{u\in p\setminus\{s\}}\sigma(k,s,u)\}$;
4 **end**
5 **end**
6 $\tilde{v}\leftarrow\arg\min_{v\in\mathcal{F}_k}\{\sum_{s\in\mathcal{S}_k} B^k_s\zeta_{k,v,s}(l,\sigma)\}$;
7 $\tilde{p}_s\leftarrow\tilde{p}_{\tilde{v},s}$ for $\forall s\in\mathcal{S}_k$;
// bandwidth allocation
8 $\Upsilon_e\leftarrow 0$ for $\forall e\in\mathcal{E}$;
9 $\Upsilon_{s,u}\leftarrow 0$ for $\forall s\in\mathcal{S}_k, u\in\mathcal{V}\setminus\{s\}$;
10 **for** $\forall s\in\mathcal{S}_k$ **do**
11 **for** *link* $\forall e\in\tilde{p}_s$ **do**
12 $\Upsilon_e\leftarrow\Upsilon_e+B^k_s$;
13 **end**
14 **for** *node* $\forall u\in\tilde{p}_s\setminus\{s\}$ **do**
15 $\Upsilon_{s,u}\leftarrow\Upsilon_{s,u}+B^k_s$;
16 **end**
17 **end**
18 $\Upsilon^1_{\max}\leftarrow\max_{e\in\mathcal{E}}\{\Upsilon_e/c_e\}$;
19 $\Upsilon^2_{\max}\leftarrow\max_{s\in\mathcal{S}_k,u\in\mathcal{V}\setminus\{s\}}\{\Upsilon_{s,u}/r^k_s B^k_s\}$;
20 $\tilde{\eta}\leftarrow 1/\max\{\Upsilon^1_{\max},\Upsilon^2_{\max}\}$, $\phi_s\leftarrow B^k_s\cdot\tilde{\eta}$ for $\forall s$;
21 **return** $(\tilde{\boldsymbol{p}},\boldsymbol{\phi},\tilde{v},\tilde{\eta})$.

A key building block is the PrimUpdt subroutine shown in Algorithm 3, which produces a primal update for application k that will be incorporated into the current primal solution. 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_s^k, 2) total bandwidth on every link e does not exceed e's capacity c_e, 3) the robustness requirement is also satisfied at each node u for each application's each data source, and 4) the minimum ratio ($\tilde{\eta}$) between any source's bandwidth and its demand is maximized. This is done in Lines 8–20 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 dual-shortest feasible paths in Line 12. 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 [33], which can obtain a $(1+\omega')$-approximation of the dual-shortest feasible path within polynomial time. With carefully selected ϵ, ω' and γ, such an approximation is sufficient for obtaining our desired performance bound.

Let us now discuss the example in Fig. 1. For simplicity, we omit robustness and the dual variables $\sigma(k,s,w)$. Dual variable $l(e)$ acts as the *shadow price* for using a link, *i.e.*, how congested the link is based on the existing flow. Initially, all links in Fig. 1 have the same shadow price since they have the same capacity; we assume the price is $l(e) = 1.0$ for $\forall e$. Executing PrimUpdt, we are to find 1) a host node $v \in \{\mathsf{X}, \mathsf{Y}\}$, and 2) a path from a to v and a path from b to v, such that the demand-weighted sum of shadow prices of both paths is minimized. Based on the graph, the host selected is X, and the paths are A–B and C–A–B respectively. We then have both paths allocated with 4 Mbps of flow, so as not to exceed the bottleneck capacity of 8 Mbps on link A–B. Next is to update the shadow prices on all links used. For simplicity, we assume $\epsilon = 0.5$. Each link's update is based on the flow pushed along it, and we have $l(\mathsf{A}\text{–}\mathsf{B}) = 1.0 \cdot (1 + 0.5 \cdot 8/8) = 1.5$, and $l(\mathsf{C}\text{–}\mathsf{A}) = 1.0 \cdot (1 + 0.5 \cdot 4/8) = 1.25$. Clearly, both A–B and C–A are now *more expensive* than the other links due to the flow, with A–B being even more expensive due to being saturated in the first round. If we repeat the above process, the next host that we find is Y, since now the sum price of C–E–F and A–D–F is lower than that of C–A–B and A–B. Again, we allocate the bottleneck bandwidth along these paths, this time both equal to 5 Mbps. Note that when calculating the bandwidth, we omit all the flow pushed in previous rounds. Also, if we have multiple applications, they must be all fulfilled once in each phase, and hence we will push 1 Mbps instead of 5 Mbps to finish the current phase, as shown in the algorithm. Repeating this process, the algorithm will approximately distribute the load over different host nodes and/or paths after a number of phases. Though we may exceed link capacities by pushing the bottleneck flow for many times, this can be resolved by scaling the final flow based on the capacities. With the above in mind, we next rigorously analyze the performance of our algorithm.

Theorem 3. *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, $\nu \in \{l, \sigma, \zeta_{k,v,s}, \psi_k, \alpha, \Delta, \phi_s, \tilde{v}, \tilde{p}_s\}_{k,v,s,e}$, 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 $\nu(l,\sigma)$ if no ambiguity is introduced.

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

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

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^k flow for $\forall s \in \mathcal{S}_k$, we have the following by summing up for all steps:

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

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

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

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

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

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

The initial dual objective value is $\Delta^0 = (E + (V-1)S)\gamma$ given the initial l and σ. Let ρ^* be the last phase before the algorithm stops. We know that $\Delta^{\rho^*} \geq 1$ and $\Delta^{\rho^*-1} < 1$. Then we can bound the optimal dual objective value Δ^* as follows:

$$\Delta^* \leq \frac{(\rho^*-1)\cdot\epsilon(1+\omega')}{(1-\epsilon(1+\omega'))\ln\frac{1-\epsilon(1+\omega')}{(E+(V-1)S)\gamma}}.$$

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, and the bandwidth at each node u by $r_s^k B_s^k$ for application k's data source s. 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 17; similarly, when the flow of (k,s) through

a node u increases by $r_s^k B_s^k$, the node's dual length $\sigma(k,s,u)$ is increased by at least $(1+\epsilon)$ times, due to the dual update in Line 18. Now, as $\Delta^{\rho^*-1} < 1$, we have $l^{\rho^*-1}(e) < 1/c_e$ for $\forall e \in \mathcal{E}$, and $\sigma^{\rho^*-1}(k,s,u) < 1/r_s^k B_s^k$ for $\forall k,s,u$. 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_s^k 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:

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

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

It remains to remove our assumption that $\lambda^* \geq 1$. Based on [14], 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 $\lambda^* \geq 1$ by scaling all demands by $1/\lambda_{\mathsf{LB}}$. Following [13], 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 $\bar{\lambda}_v^k = \min_{s\in\mathcal{S}_k}\{\bar{b}_{v,s}^k/B_s^k, r_s^k\}$. 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 $\lambda_{\mathsf{UB}} = E\min_k\{\bar{\lambda}_k\}$, as each flow can be decomposed into up to E paths, with no contention among each other. 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 [13], [14], the number of phases is bounded by $\rho^* \leq \lceil \Delta^* \log_{1+\epsilon}\frac{1}{\gamma}\rceil = O^*(\frac{\Delta^*}{\omega^2})$, each with K iterations, and the total number of steps is bounded by $(E+(V-1)S)\log_{1+\epsilon}\frac{1+\epsilon}{\gamma} = O^*((E+(V-1)S)\frac{\Delta^*}{\omega^2})$ 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.* [33], each path is found in $O^*(\frac{1}{\omega'}VE)$ time. Bandwidth allocation in PrimUpdt takes $O(SV)$ 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 $O^*(\frac{\Delta^*}{\omega^3}SFVE(E+(V-1)S+K))$.

To remove the dependency on Δ^*, we employ the demand scaling technique in [14]. If the algorithm does not stop after $\lceil 2\log_{1+\epsilon}\frac{1}{\gamma}\rceil$ phases, we know that $\Delta^* \geq 2$. We then double all demands, hence halving Δ^*, and then re-run Algorithm 2. Now, we have $\Delta^* \in [1, SE]$ after the initial scaling in Part I. Hence at most $O(\log_2(SE))$ demand scaling rounds are needed to bring Δ^* within $[1,2]$, each spending $O^*(\frac{1}{\omega^3}SFVE(E+K))$ time. Omitting the logarithm terms, the final complexity is $O^*(\frac{1}{\omega^3}SFVE(E+(V-1)S+K))$ combined with the initial scaling. The theorem follows. ■

C. NO-MAP Formulation and Randomized Algorithm

NO-MAP is the hardest among the four problems. Though its hardness follows from that of NO-SAP, there are $O(F^K)$ possible HD solutions in the worst case for NO-MAP, instead of the linear number in NO-SAP. This prevents us from iterating over all possible HD combinations. Below, we first give an exact formulation of NO-MAP. We similarly define $x(k,v) \in \{0,1\}$ as the indicator of whether 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.

$$\max \quad \lambda \tag{6a}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}_{v,s}^k}\mathcal{L}(p) \geq B_s\cdot\lambda\cdot x(k,v), \qquad \forall k,v,s; \tag{6b}$$

$$\sum_{v\in\mathcal{F}_k} x(k,v) = 1, \qquad \forall k; \tag{6c}$$

$$\sum_{p\in\mathcal{P}:e\in p}\mathcal{L}(p) \leq c_e, \qquad \forall e\in\mathcal{E}; \tag{6d}$$

$$\sum_{p\in\mathcal{P}_s^k:e\in p}\mathcal{L}(p) \leq r_s^k\cdot B_s^k, \qquad \forall k,s,e\in\mathcal{E}; \tag{6e}$$

$$x(k,v)\in\{0,1\}, \mathcal{L}(p), \lambda \geq 0, \quad \forall k,v,p. \tag{6f}$$

Explanation: Program (6) has the same form as Program (3), except Constraint (6e) that enforces *link-robustness* for each link e instead of *node-robustness* for each node u. This is because NO-MAP cannot support node-robustness: its final selected host is always a single point of failure.

Due to binary variables $x(k,v)$ and Constraint (6b), Program (6) is a Mixed Integer Quadratic Program (MIQP), which is generally hard to solve. However, by relaxing the integer constraints on $x(k,v)$, we arrive at a QP that has almost the same structure as Program (3). We can then apply the same transformation as from Program (3) to Program (4), which also generates an LP, in other words, the *linear relaxation* of Program (6). The linear relaxation is written as follows:

$$\max \quad \lambda \tag{7a}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}_{v,s}^k}\mathcal{L}(p) \geq B_s\cdot y(k,v), \qquad \forall k,v,s; \tag{7b}$$

$$\sum_{v\in\mathcal{F}_k} y(k,v) \geq \lambda, \qquad \forall k; \tag{7c}$$

$$\sum_{p\in\mathcal{P}:e\in p}\mathcal{L}(p) \leq c_e, \qquad \forall e\in\mathcal{E}; \tag{7d}$$

$$\sum_{p\in\mathcal{P}_s^k:e\in p}\mathcal{L}(p) \leq r_s^k\cdot B_s^k, \qquad \forall k,s,e\in\mathcal{E}; \tag{7e}$$

$$y(k,v), \mathcal{L}(p), \lambda \geq 0, \qquad \forall k,v,p. \tag{7f}$$

Due to the similar structure of Program (7) and Program (4), we can basically adopt the same method as in Algorithms 2 and 3 to obtain an FPTAS to Program (7), for which the details are omitted. Based on the FPTAS, we then propose a randomized algorithm to NO-MAP, as shown in Algorithm 4. It starts by solving Program (7) using a modified version of 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 NO-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 NO-SAP, and hence can be solved using the FPTAS in [38].

Algorithm 4: Randomized Algorithm $A_{\text{NO-MAP}}$

Input: Network G, application set $\mathbf{\Gamma}$, tolerance ω
Output: Scaling ratio λ, host selections $\mathbf{v}$, path sets $\mathbf{P}$, 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 **end**
6 Solve NO-MAP (Program (6)) with fixed HD solution $\mathbf{v} = \{v_k\}_k$, with accuracy ω;
7 **return** $(\lambda, \{v_k\}_k, \{P_s^k\}_{k,s}, \mathcal{L})$.

The time complexity of Algorithm 4 is dominated by the complexity of the FPTAS to PO-MAP and the FPTAS for solving the DR subproblem with fixed HD. Therefore, it also runs in time polynomial to the input size and $\frac{1}{\omega}$. Unfortunately, the randomized algorithm does not have a constant approximation ratio. Non-constant performance bound can be obtained via conventional stochastic theorems such as the Chernoff bound. Such a result, however, is far from providing a realistic performance bound that is useful in practical settings. We thus omit the theoretical analysis of Algorithm 4 for simplicity.

VII. Performance Evaluation

A. Experiment Settings

We used randomly generated topologies and applications for performance evaluation. The topologies were generated using the Waxman model [28]. Each topology has 20 nodes, where 20% randomly selected nodes were facility nodes. Links were created using parameters α and β in the Waxman model, where $\alpha = \beta = 0.6$. Link capacities were uniformly generated in $[10, 100]$ Mbps, and delays were uniformly generated in $[1, 10]$ ms. In each experiment, we generated 5 IoT applications. An application had $[3, 10]$ data sources. Application delay bounds were randomly generated in $[15, 25]$ ms. For each data stream, its bandwidth demand were randomly generated in $[1, 25]$ Mbps. The default robustness (maximum tolerable data loss ratio) was 0.5 for all data streams. 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.

Our comparison algorithms are shown in Table II. 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 (Algorithm 4 for the non-parallelizable case) into its subroutines for solving HD (Lines 1–5) and DR (Line 6) respectively. Each combination of HD and DR algorithms was denoted by {HD}+{DR}, *e.g.*, NS+GH uses NS for HD and GH for DR.

TABLE II: Implemented Algorithms

SAP	Our SAP algorithm. For parallelizable applications, this is our FPTAS to PO-MAP (Algorithm 2). For non-parallelizable applications, this is our FPTAS to NO-SAP (Algorithm 1).
MAP	Our MAP algorithm. For parallelizable applications, this is our FPTAS to PO-MAP (Algorithm 2). For non-parallelizable applications, this is our randomized algorithm to NO-MAP (Algorithm 4).
ODA	*Optimal Delay-Agnostic* algorithm. For parallelizable applications, it just solves an edge-flow multi-commodity flow (MCF) LP. For non-parallelizable applications, it attempts all combinations of application HD, each solving an edge-flow MCF LP that neglects applications' delay bounds. ODA yields an upper bound on the optimal delay-bounded solution.
NS (HD)	*Nearest Selection* HD heuristic. For each application, this selects the host with minimum maximum delay from all data sources.
RS (HD)	*Random Selection* HD heuristic. For each application, a random candidate host is selected that is within the delay bound from every data source.
GH (DR)	*Greedy Heuristic* for DR. This works in rounds where in each round, the delay-shortest path with positive capacity is found for every data stream, and then bandwidth allocation is done as in Lines 8–20 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. An edge-flow MCF LP, which neglects application delay bounds, is solved. This yields an upper bound on DR.

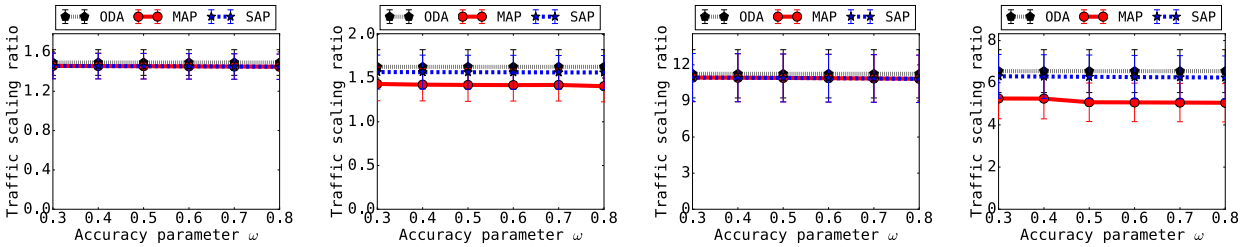
We used the following metrics in performance evaluation. *Traffic scaling ratio* is the optimization objective λ, which is the minimum ratio between the allocated bandwidth and the demand of every data stream. *Maximum delay ratio* is the average ratio between the maximum transmission delay received 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 [17] 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

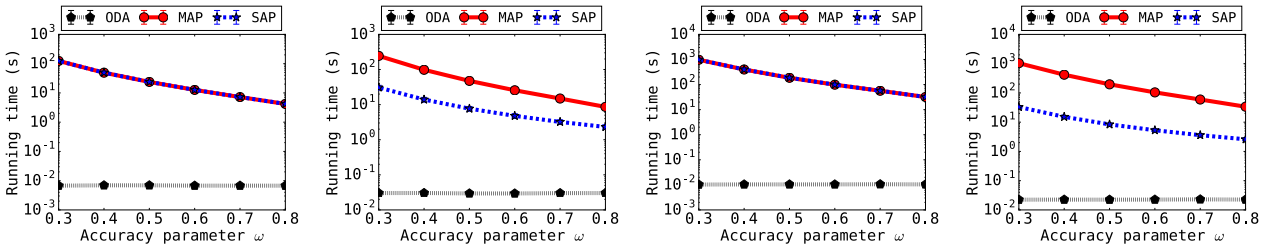
In figures, error bars show 95% confidence intervals (CIs).

1) Single-Application Scenario: We use our single application experiments to show 1) that our algorithms are close-to-optimal through comparison with the theoretical upper bound (ODA), 2) the impact of robustness on provisioning performance, and 3) the impact of parallelizability on our algorithms. The results are shown in Figs. 2–5. For a single parallelizable application, SAP and MAP are essentially the same algorithm (Algorithm 2), and hence they have the same performance.



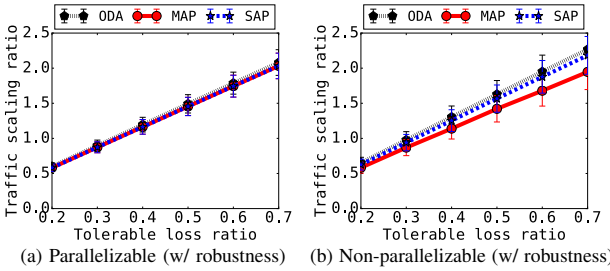
(a) Parallelizable w/ robustness (b) Non-parallelizable w/ robustness (c) Parallelizable w/o robustness (d) Non-parallel. w/o robustness

Fig. 2: Single application: objective value against accuracy parameter ω.



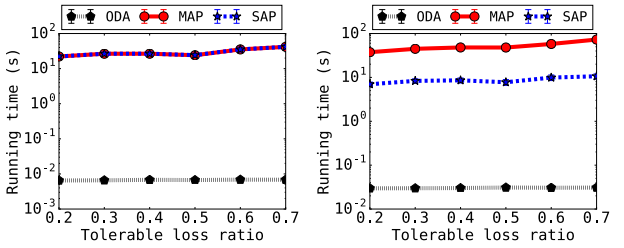
(a) Parallelizable w/ robustness (b) Non-parallelizable w/ robustness (c) Parallelizable w/o robustness (d) Non-parallelizable w/o robustness

Fig. 3: Single application: running time vs. accuracy parameter ω.



(a) Parallelizable (w/ robustness) (b) Non-parallelizable (w/ robustness)

Fig. 4: Single application: objective value vs. robustness.



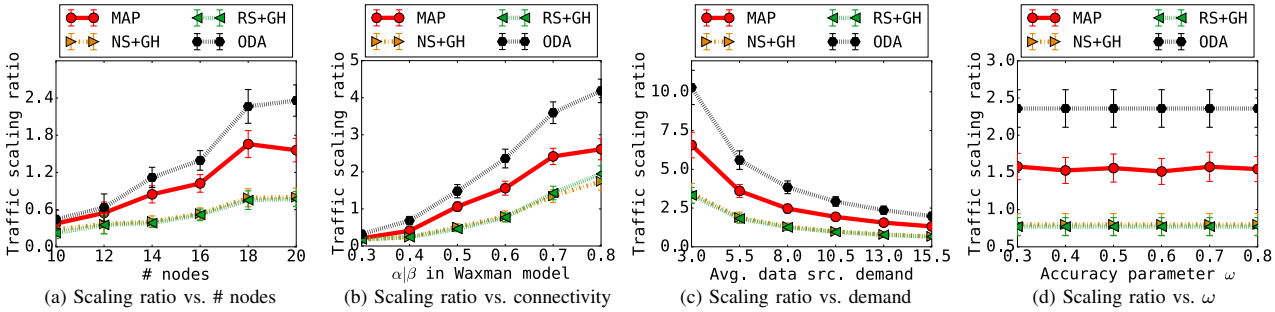
(a) Parallelizable (w/ robustness) (b) Non-parallelizable (w/ robustness)

Fig. 5: Single application: running time vs. robustness.

Figs. 2 and 3 show four combination scenarios: robust parallelizable application, robust non-parallelizable application, non-robust parallelizable application, and non-robust non-parallelizable application. We varied accuracy parameter ω from 0.3 to 0.8. Note that ω must be strictly positive ($\omega = 0$ represents the optimal solution and hence cannot be obtained due to the NP-hardness), while $\omega = 1$ means no guarantee and hence is also meaningless for approximation algorithms. First, we can see that our SAP FPTASs (MAP in Figs. 2(a) and 2(c) and SAP in Figs. 2(b) and 2(d)) achieve objective values extremely close to the upper bound ODA, much greater than their theoretical bounds ($(1-\omega)$ times the optimal). In Figs. 2(b) and 2(d), the MAP randomized algorithm achieves slightly worse performance than the FPTASs, yet its performance is still pretty close to ODA and even higher than $(1-\omega)$ times the optimal (though this is not theoretically guaranteed). This shows that the proved theoretical bounds are pretty conservative in practice. Meanwhile, with decreasing ω, little changes can be observed on the objective value, while the running time is greatly reduced in Figs. 3(a)–3(d). An empirical setting of $\omega \geq 0.5$ can be used in practice to achieve close-to-optimal performance with reasonable overhead.

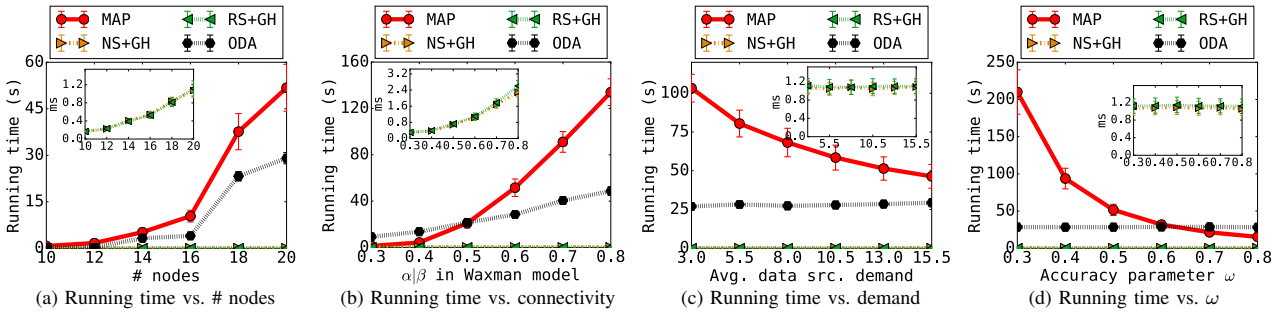
Comparing Figs. 2(a) and 2(c) (and similarly Figs. 2(b) and 2(d)), we can observe the impact of robustness. Enforcing robustness clearly reduces the objective value by great amounts. This shows that in practice, applications with robustness requirements can find it much harder to get accommodated when the system has limited resources. Looking at Fig. 3, running time increases when robustness is removed, which is due to that the complexity depends on the objective value, matching our previous analysis. Note that we did not use the polynomial-time demand scaling technique in our experiments, in order to better present this correlation.

Comparing Figs. 2(a) and 2(b), we can see the impact of parallelizability when robustness is enforced. Both ODA and SAP show that parallelizability reduces the objective value. This is because parallelizable applications can enjoy the more strict *node-robustness*, while non-parallelizable applications can only achieve *link-robustness*; clearly the former consumes more resource, as it guarantees the latter while providing additional protection. However, when robustness is not enforced, we see the opposite in Figs. 2(c) and 2(d). Applications with parallelizability achieve better scaling ratios than those without, since without robustness, the parallelizable problem is an LP relaxation of the non-parallelizable problem, and hence the former represents an upper bound on the latter. Looking at running times in Figs. 3(a) and 3(b), the time for the non-parallelizable case approximately doubles that for the parallelizable case. This is because in the non-parallelizable case, the same formulation is solved twice for HD and DR



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

Fig. 6: Multi-application: objective value vs. number of nodes, connectivity (α, β), bandwidth demand, and accuracy (ω).



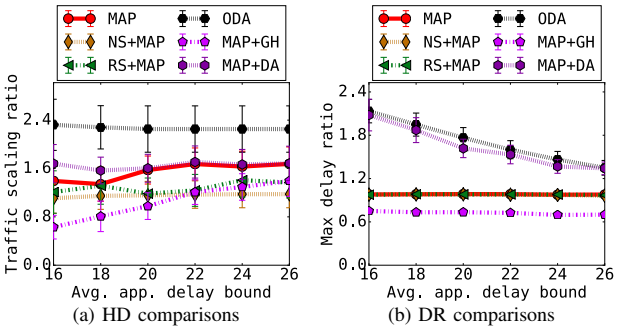
(a) Running time vs. # nodes (b) Running time vs. connectivity (c) Running time vs. demand (d) Running time vs. ω

Fig. 7: Multi-application: running time vs. number of nodes, connectivity (α, β), bandwidth demand, and accuracy (ω).

respectively; in the parallelizable case, both are solved simultaneously. Note that Algorithm 1 (SAP in Fig. 3(b)) is much faster than both Algorithm 2 (SAP/MAP in Fig. 3(a)) and Algorithm 4 (MAP in Fig. 3(b)), since Algorithm 1 solves a formulation that has fewer variables. Figs. 3(c) and 3(d) show similar comparisons. The reason why the running time of MAP does not double in Fig. 3(d) is that the optimal objective value decreases greatly due to the non-parallelizability, and hence the running time of DR is dominated by the time of HD.

We further show in Figs. 4 and 5 the impact of different robustness parameters. We varied the tolerable loss ratio r from 0.2 to 0.7. The value r is also strictly positive, since our robustness scheme cannot (and does not aim to) provide full protect but is instead to bound the loss due to a failure; $r = 1$ means no protection and hence leads to the same results as the experiments without robustness. With our formulation, the objective value should increase with the tolerable loss ratio when the resources are relatively abundant, which is validated in Fig. 4. Figs. 4(a) and 4(b) show the same comparison as in Figs. 2(a) and 2(b), *i.e.*, parallelizable applications achieve worse scaling ratios than non-parallelizable ones due to the enforcement of node-based instead of link-based protection. The running time increases with higher objective value due to the relaxation of robustness requirement (larger tolerable loss).

2) Multi-Application Scenario: Here, we omitted robustness, and focused on the non-parallelizable application case which is more common in practice. Figs. 6 and 7 show experiment results for multiple applications, 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 lead to smaller scaling ratios. The positive correlation between time complexity and the scaling ratio is further validated in Fig. 7(c). Finally, Figs. 6(d) and 7(d) show similar results as in Figs. 2 and 3 for MAP, where a looser accuracy parameter ω does not lead to noticeable performance loss, but greatly reduces its running time.

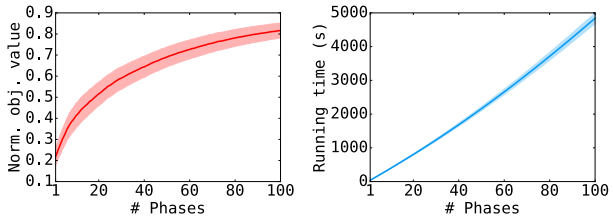


(a) HD comparisons (b) DR comparisons

Fig. 8: Multi-application: HD and DR with varying delay.

The above experiments show the superior performance of our MAP algorithm. In Figs. 8, we further analyze its performance for HD and DR separately, where we combined MAP's subroutines (denoted as MAP-HD and MAP-DR respectively) with different heuristics. Shown in Fig. 8(a), delay-aware DR solutions (GH and MAP-DR) achieve better scaling ratios

with larger delay bounds, while delay-agnostic solutions do not. Comparing HD algorithms, MAP-HD still achieves much better scaling ratios than either NS or RS. Interestingly, RS outperforms NS, because NS can lead to congestion when a host is closer to all data sources than the others. Comparing DR algorithms, MAP-DR outperforms GH. Given fixed HD, the DA algorithm is optimal for delay-agnostic DR. We can see that MAP-DR is close-to-optimal when delay bounds are large. In Fig. 8(b), with MAP-DR, the max delay ratio is always bounded by but close to 1, meaning it utilizes paths of various yet strictly bounded delays. GH also respects delay bounds but uses shorter paths, which leads to low traffic scaling ratios in Fig. 8(a). ODA and DA are delay-agnostic, hence they can lead to delays over $2\times$ the bounds, violating the QoS requirements. In summary, the advantage of MAP comes from both its HD and DR subroutines, compared to the heuristics.



(a) Objective value vs. # phases (b) Running time vs. # phases

Fig. 9: Heuristic convergence. Shadow shows 95% CIs.

3) Heuristic Usage: The proposed algorithms are close-to-optimal, but require a large number of phases to be executed. However, they can also be used in a heuristic manner, by omitting the demand scaling technique and/or setting a fixed *phase bound*. In Fig. 9, we test the heuristic implementation on larger problem instances, with $5\times$ more nodes ($\alpha = \beta = 0.3$ in Waxman model) and $5\times$ more applications. In Fig. 9(a), we show the objective value after each phase, normalized by the upper bound ODA. The heuristic algorithm converges stably, and can quickly find a reasonably accurate solution. A solution no less than 0.5 (our default ω) of the upper bound is obtained within 25 phases in most cases. For complexity, the slightly increased time at higher phase numbers is due to the extra overhead for maintaining the results of all phases so far. Since in IoT, typically provisioning is done within a small geographical area, we expect the number of nodes and the number of concurrent applications both not to exceed tens to a hundred. Given that application provisioning is a relatively infrequent operation that happens in days or even months, our algorithms and/or this heuristic implementation can well handle such typical scenarios in practice. Further running time reduction can be achieved by using heuristics instead of the FPTAS for the DCLC computations, such as [31].

VIII. Conclusions

In this paper, we studied the provisioning of real-time processing applications in IoT. We considered both the QoS and the robustness requirements of the applications. We considered two application types: parallelizable and non-parallelizable. For either type, we further studied both the provisioning of a single application, and the joint provisioning of multiple applications. We proved all four versions of the problem NP-hard. We then showed that three of the four versions admit FPTASs. For the last one, we proposed a randomized algorithm. We validated the advantages of our proposed algorithms over several heuristics through simulations.

References

[1] "IoT Market Forecasts." URL: https://www.postscapes.com/internet-of-things-market-size/

[2] S. Acharya, B. Gupta, P. Risbood, and A. Srivastava, "PESO: Low Overhead Protection for Ethernet over SONET Transport," in *Proc. IEEE INFOCOM*, 2004, 165–175.

[3] 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, jun 2017.

[4] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing High Availability Goals for Virtual Machine Placement," in *Proc. IEEE ICDCS*, 2011, 700–709.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proc. ACM MCC*, 2012, 13–16.

[6] 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 *Proc. IEEE INFOCOM*, 2015, 1921–1929.

[7] 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, jun 2017.

[8] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in *Proc. IEEE INFOCOM*, 2009, 783–791.

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

[10] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol," *IEEE/ACM Trans. Netw.*, 23(2): 465–478, apr 2015.

[11] 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.

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

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

[14] N. Garg and J. Konemann, "Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems," in *Proc. ACM FOCS*, 1998, 300–309.

[15] A. Giordano, G. Spezzano, and A. Vinci, "Smart Agents and Fog Computing for Smart City Applications," in *Smart-CT*, 2016, 137–146.

[16] M. Gowda, A. Dhekne, S. Shen, R. R. Choudhury, L. Yang, S. Golwalkar, and A. Essanian, "Bringing IoT to Sports Analytics," in *Proc. USENIX NSDI*, 2017, 499–513.

[17] Gurobi, "Gurobi Optimizer." URL: http://www.gurobi.com/products/gurobi-optimizer

[18] 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 *Proc. IEEE ICNP*, 2016, 1–10.

[19] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz, "Optimizing Virtual Backup Allocation for Middleboxes," in *Proc. IEEE ICNP*, 2016, 1–10.

[20] 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 *Proc. IEEE INFOCOM*, 2017.

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

[22] S. Misra, G. Xue, and D. Yang, "Polynomial Time Approximations for Multi-Path Routing with Bandwidth and Delay Constraints," in *Proc. IEEE INFOCOM*, 2009, 558–566.

[23] M. R. Rahman and R. Boutaba, "SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization," *IEEE Trans. Netw. Serv. Manag.*, 10(2): 105–118, jun 2013.

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

[25] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online Job Dispatching and Scheduling in Edge-Clouds," in *Proc. IEEE INFOCOM*, 2017, 1–9.

[26] L. Toka, B. Lajtha, E. Hosszu, B. Formanek, D. Gehberger, and J. Tapolcai, "A Resource-Aware and Time-Critical IoT Framework," in *Proc. IEEE INFOCOM*, 2017, 1–9.

[27] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *Proc. IEEE INFOCOM*, 2016, 1–9.

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

[29] Y. Xiao and M. Krunz, "QoE and Power Efficiency Tradeoff for Fog Computing Networks with Fog Node Cooperation," in *Proc. IEEE INFOCOM*, 2017, 1–9.

[30] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, "Enhancing Survivability in Virtualized Data Centers: A Service-Aware Approach," *IEEE J. Sel. Areas Commun.*, 31(12): 2610–2619, dec 2013.

[31] G. Xue, "Minimum-cost QoS Multicast and Unicast Routing in Communication Networks," *IEEE Trans. Commun.*, 51(5): 817–824, may 2003.

[32] G. Xue, L. Chen, and K. Thulasiraman, "Quality-of-service and Quality-Of-Protection Issues in Preplanned Recovery Schemes Using Redundant Trees," *IEEE J. Sel. Areas Commun.*, 21(8): 1332–1345, oct 2003.

[33] 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, jun 2008.

[34] J. Yallouz and A. Orda, "Tunable QoS-Aware Network Survivability," *IEEE/ACM Trans. Netw.*, 25(1): 139–149, feb 2017.

[35] J. Yallouz, O. Rottenstreich, and A. Orda, "Tunable Survivable Spanning Trees," *IEEE/ACM Trans. Netw.*, 24(3): 1853–1866, jun 2016.

[36] H. Yanagisawa, T. Osogami, and R. Raymond, "Dependable Virtual Machine Allocation," in *Proc. IEEE INFOCOM*, 2013, 629–637.

[37] H. Yu, C. Qiao, V. Anand, X. Liu, H. Di, and G. Sun, "Survivable Virtual Infrastructure Mapping in a Federated Computing and Networking System under Single Regional Failures," in *Proc. IEEE GLOBECOM*, 2010, 1–6.

[38] R. Yu, G. Xue, and X. Zhang, "QoS-Aware and Reliable Traffic Steering for Service Function Chaining in Mobile Networks," *IEEE J. Sel. Areas Commun.*, 35(11): 2522–2531, nov 2017.

[39] ——, "Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective," in *Proc. IEEE INFOCOM*, 2018, 1–9.

[40] R. Yu, G. Xue, X. Zhang, and D. Li, "Survivable and Bandwidth-Guaranteed Embedding of Virtual Clusters in Cloud Data Centers," in *Proc. IEEE INFOCOM*, 2017, 1–9.

[41] 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, dec 2016.

[42] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable Virtual Data Center Embedding in Clouds," in *Proc. IEEE INFOCOM*, 2014, 289–297.

[43] W. Zhang, J. Tang, C. Wang, and S. de Soysa, "Reliable Adaptive Multipath Provisioning with Bandwidth and Differential Delay Constraints," in *Proc. IEEE INFOCOM*, 2010, 1–9.

[44] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang, "Towards Bandwidth Guarantee in Multi-Tenancy Cloud Computing Networks," in *Proc. IEEE ICNP*, 2012, 1–10.

Ruozhou Yu (Student Member 2013, Member 2019) is an Assistant Professor of Computer Science at North Carolina State University. He received his Ph.D degree (2019) in Computer Science from Arizona State University, Tempe, Arizona, USA. His research interests include internet-of-things, cloud/edge computing, security and privacy, blockchain, learning-based networking, algorithms and optimization, etc.

Guoliang Xue (Member 1996, Senior Member 1999, Fellow, 2011) is a Professor of Computer Science and Engineering at Arizona State University. He received the Ph.D degree (1991) in computer science from the University of Minnesota, Minneapolis, USA. His research interests include survivability, security, and resource allocation issues in networks. He is an Editor of *IEEE Network* and the Area Editor of *IEEE Transactions on Wireless Communications* for the area of Wireless Networking.

Xiang Zhang (Student Member 2013) received his B.S. degree from University of Science and Technology of China, Hefei, China, in 2012. Currently he is a Ph.D candidate in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. His research interests include network economics and game theory in crowdsourcing and cognitive radio networks.