# EA-Market: Empowering Real-Time Big Data Applications with Short-Term Edge SLA Leases

Ruozhou Yu, Huayue Gu, Xiaojian Wang, Fangtong Zhou, Guoliang Xue, Dejun Yang

***Abstract*—Edge computing promises to bring low-latency and high-throughput computing, but the limited edge resources may cause frequent congestion and lead to unstable and unpredictable performance. To ensure performance guarantee, application owners can establish Service-Level Agreements (SLAs) with the edge provider for resource reservation or priority usage. But it is cost-inefficient for application owners to lease long-term SLAs based on peak demands, as demands can fluctuate, and the leased resources may be idle or underutilized at most times.**

**This paper studies market mechanism design for short-term edge SLA leases, focusing on real-time big data applications with throughput and latency goals. Applications submit short-term SLA requests to serve users with guaranteed performance during peak hours. As SLA requests arrive over time, the edge provider dynamically provisions edge resources to fulfill the requests, while charging application owners based on the current demands. We design *EA-Market*, an online combinatorial auction mechanism that achieves a competitive social welfare, while guaranteeing truthfulness, budget balance, individual rationality, and computational efficiency. Notably, our mechanism enables each application owner to bid without knowledge of the edge infrastructure, and gives edge provider full control over resource provisioning to fulfill the requests. We perform theoretical analysis and simulations to evaluate the efficacy of our mechanism.**

***Index Terms*—Edge Computing, Resource Provisioning, Competitive Analysis, Network Economics**

## I. Introduction

Edge computing brings high-throughput low-latency computing to applications with strict performance requirements, such as virtual/augmented reality (VR/AR) [1], [2], smart retail [3], etc. However, edge computing has limited computing and network resources. The edge infrastructure may be over-utilized, and users may suffer from unstable and unpredictable performance, causing profit loss to both the edge provider (EP) and application owners (AOs).

To ensure performance, an AO can establish Service-Level Agreements (SLAs) with the EP. An SLA specifies the quality-of-service (QoS) requirements of the application such as latency and throughput, which must be satisfied during the time when the SLA is active. The traditional long-term SLAs, however, are cost-inefficient to AOs or the EP. Specifically, edge demands are commonly geo-distributed and time-varying [4], and provisioning long-term SLAs based on peak demands can lead to resource under-utilization at most times.

In this case, *short-term SLA leases* are most suitable for many applications. An AO can purchase SLAs only during peak hours or based on user demands, and adjust SLA terms when demand distribution changes. This reduces the AO's cost for paying for under-utilized resources. Meanwhile, the EP can dynamically price the SLAs based on demand and supply to improve its own revenue.

This paper studies market mechanism design for short-term edge SLA leases, AOs submit their SLA requests dynamically, and the EP provisions network and computing resources to fulfill requests subject to resource capacities. We specifically focus on *real-time big data applications*, which process data streams from distributed data sources with throughput and real-time requirements [4], [5]. We formulate mechanism design as online social welfare maximization that involves both resource provisioning and pricing.

While there exists a rich literature on edge resource mechanism design (see Sec. II), existing work falls short on two key aspects. First, many consider computing resources only, or assume over-simplified network models such as single-hop wireless links or static bandwidth. They fail to capture the complex edge network topology and the resulting network bottlenecks. Second and more importantly, many auctions rely on the AOs to pick and bid for combinations of resources (such as virtual machines or bandwidth) to satisfy their own demands and goals. While this simplifies mechanism design, it pushes the burden of picking the most suitable resources to AOs, and risks exposing critical infrastructure information (*e.g.*, topology or resources) which could incur security vulnerabilities. We aim to design a mechanism to avoid these issues.

Our main contribution is an online auction mechanism for short-term SLA leases, called *EA-Market*, which avoids these issues. The mechanism further *efficiently* achieves a social welfare within a *competitive ratio* of the optimal offline value, while ensuring that *no* AO has an *incentive to misreport* her valuation, and that every party has a *non-negative utility*. Notably, our mechanism allows each owner to succinctly specify her application's *demands*, *QoS requirements* (*e.g.*, throughput and latency) and *valuation*, instead of obtaining system details and picking resources by herself. The EP has full control over provisioning of actual resources to satisfy the requests. The core of our design lies combination of two key techniques: a primal-dual-inspired online resource auction framework that achieves the desired economic properties including competitiveness, and an approximation scheme for the NP-hard one-shot provisioning problem. We perform rigorous theoretical analysis of the economic properties, and conduct extensive simulations for evaluation. Our main contributions are summarized as follows:

- We formulate mechanism design for three types of real-time big data applications (asynchronous, synchronous

Yu, Gu, Wang, and Zhou ({ryu5, hgu5, xwang244, fzhou}@ncsu.edu) are with NC State University, Raleigh, NC 27606, USA. Xue (xue@asu.edu) is with Arizona State University, Tempe, AZ 85281, USA. Yang (djyang@mines.edu) is with Colorado School of Mines, Golden, CO 80401, USA. This research was supported in part by NSF grants 2007391, 2007469, and 2045539.

and centralized) in a general edge network as a social welfare maximization problem, and prove its NP-hardness.
- We propose an online mechanism called EA-Market, which achieves competitiveness in social welfare, truthfulness, individual rationality, budget balance, and computational efficiency for all three types of applications.
- Our mechanism implies the best known bound for *offline* social welfare maximizing auction under the same setting.
- We conducted extensive simulation experiments to validate the performance and efficiency of our mechanism.

In the following, Sec. II introduces the background and related work. Sec. III presents our system, application and provisioning models. Sec. IV details our mechanism design, analysis, and discussions. Sec. V shows evaluation results. Sec. VI concludes this paper.

## II. Background and Related Work

### A. Resource Auctions in Cloud and Edge

Dynamic pricing has been used in public clouds such as Amazon EC2 [6], but real-world pricing schemes lack desirable economic properties such as truthfulness, leading to a rich literature on truthful cloud resource auctions [7]–[13]. Cloud auctions typically neglect or over-simplify the network model due to the regular network structure and abundant bandwidth in the cloud. Compared to the cloud, edge computing has limited computing and bandwidth resources, and hence both must be accounted for in resource auctions, such as in [14]–[16]. To tackle the added complexity of network structures and constraints, many have focused on specific network scenarios, such as single-hop wireless/backhaul communications [14], [16]–[19], multi-tier tree/line networks [15], [20], networks with static end-to-end bandwidth and/or delays [21], etc.

Few considered a general network topology in cloud/edge auctions, such as [9], [15], [22]. This problem falls under the realm of combinatorial auction design (see Sec. II-B), where existing work assumes AO's full knowledge about all available resources (cloud/edge nodes and network links), and can pick the best bundle of resources by herself. This is problematic in reality: 1) AOs have increased cognitive overhead to obtain such knowledge and run algorithms to decide their preferred resources; 2) EP has limited flexibility in the provisioning process; and 3) exposing system information such as topology or resources to external parties causes security concerns to the EP. Some auctions leverage properties of specific scenarios, such as machine learning [23], [24], blockchain [25], video streaming [14], healthcare [17], or device-to-device communications [26], etc. A comprehensive survey of auction and mechanisms in edge computing can be found in [27], confirming some of the above issues. In general, there lacks a mechanism that enables QoS-aware edge resource provisioning for general real-time big data applications, in general and heterogeneous edge networks.

### B. Combinatorial Auction Design

Combinatorial auctions [28] have been extensively studied in many scenarios. A difficulty in combinatorial auction design is *how to let a buyer express her utility over all bundles of items she wants to purchase*, *e.g.*, all combinations of edge resources to satisfy an application's demands. Given $N$ resources, there can be $\Theta(2^N)$ combinations that need to be specified, which is impractical. Existing designs address this by either assuming restricted spaces of the bundles, or letting the buyer specify the subset of bundles that she is interested in (in the simplest case, single-minded buyers [29]). In edge/cloud auctions, the former corresponds to the simplified network models, in which case the bundle of resources to satisfy each application's need is well-defined and usually unique [14]–[18], [21]. The latter means assuming that AO has knowledge over the cloud/edge infrastructure and can pick the best resources to include in her bid, leaving the burden to the owner herself [9]. We follow the principle of algorithmic mechanism design [30], and novelly address the difficulty without incurring issues brought by the above methods. Specifically, our mechanism takes polynomial-sized bids describing only properties (QoS goals) of the desired bundles, while searching an exponential space of bundles (resource allocation) per buyer with a competitive social welfare. To our knowledge, *no truthful mechanism has been designed that achieves a competitive social welfare in this case*. Our mechanism avoids pushing buyers to consider the exponential bidding strategy space, as required by existing combinatorial auctions [28]. Our design can open up new windows to the design of efficient and competitive/approximate mechanisms for scenarios other than edge computing.

## III. System Model

Table I summarizes key notations used in this paper.

### A. Edge Network Model

Consider an edge network modeled as a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of network nodes, and $\mathcal{L}$ is the set of physical links. $\mathcal{S} \subseteq \mathcal{N}$ denotes the access points (APs) via which data generating devices (sources) are connected. $\mathcal{F} \subseteq \mathcal{N}$ denotes nodes equipped with edge servers (*i.e.*, edge nodes). Note that $\mathcal{S}$ and $\mathcal{F}$ may not be disjoint, *i.e.*, some edge servers may be associated with APs. Computing resources on edge node $h \in \mathcal{F}$ are denoted by a capacity vector $C_h = (c_1^h, \ldots, c_{\mathbf{K}}^h)$, where $\mathbf{K}$ is the number of resources such as CPU, GPU, memory, etc. Each link $l \in \mathcal{L}$ has a bandwidth $b_l > 0$ and a delay $d_l > 0$. An EP manages all resources including computational resources and link bandwidth.

### B. Application Model

The EP hosts applications in edge network to earn revenue. We focus on real-time big data processing, representing a broad range of applications in Internet-of-Things (IoT), social media, and mobile computing [31]. Many applications have stringent QoS requirements *w.r.t.* throughput and latency, and can greatly benefit from leasing SLA guarantees during peak hours or congested periods.

Formally, let $\mathcal{A} = \{A_1, \ldots, A_{\mathbf{J}}\}$ be a set of SLA requests submitted by AOs. We use the words "request" and "application" interchangeably, as an AO with multiple requests

TABLE I: Notation Table

| **System Model** | |
|---|---|
| $\mathcal{G}=(\mathcal{N},\mathcal{L})$ | Edge network with nodes and links |
| $\mathcal{S},\mathcal{F}\subseteq\mathcal{N}$ | Source APs, edge servers (edge nodes) |
| $C_h=(c_1^h,\ldots,c_{\mathbf{K}}^h)$ | Computing resources of edge node $h\in\mathcal{F}$ |
| $b_l, d_l, d_p$ | Bandwidth, delay of link $l\in\mathcal{L}$ or path $p$ |
| $\mathcal{A}=\{A_1,\ldots,A_{\mathbf{J}}\}$ | Set of SLA requests (online arrival) |
| $\mathcal{S}_j\subseteq\mathcal{S}$ | Set of source APs for application $A_j$ |
| $R_j=(r_1^j,\ldots,r_{\mathbf{K}}^j)$ | Unit resource consumption of request $A_j$ |
| $B_j:\mathcal{S}_j\mapsto\mathbf{R}^+$ | Data rates of source APs of request $A_j$ |
| $\mathbf{B}_j,\mathbf{D}_j$ | Sum data rate and delay bound of $A_j$ |
| $\mathcal{P}_{s,h}^j,\mathcal{P}_s^j,\mathcal{P}^j$ | Feasible path sets between $s$ and $h$, from $s$ to all $h$, and for all $s$-$h$ pairs of $A_j$ |
| **Auction Model** | |
| $\mathcal{T}=\{T\}$ | Set of all time slots |
| $T_j^{\text{start}},T_j^{\text{end}}$ | Start and end times of request $A_j$ |
| $T_j,T_{\max}$ | Duration of $A_j$, and max duration in $\mathcal{A}$ |
| $\tau(T,j)$ | Whether $A_j$ is active in slot $T$ or not |
| $\gamma_j,v_j$ | Bid, and private value, of request $A_j$ |
| **Variables** | |
| $\psi_j=(P_j,f_j)$ | A provisioning scheme with paths $P_j$ and demand allocation function $f_j:P_j\mapsto\mathbb{R}^*$ |
| $x_j(s,h)$ | Demand from source $s$ allocated to $h$ |
| $y_j(h)$ | Demand from all sources of $A_j$ to $h$ |
| $Z=\{\zeta_j\mid j\}$ | Whether each request $A_j$ is accepted |
| $\Pi=\{\pi_j\mid j\}$ | Payment price of each request $A_j$ |
| $u_j,u_0$ | Utility functions of $A_j$, and auctioneer |
| $S(Z,\Pi)$ or $S(Z)$ | Social welfare of an auction |
| **Other** | |
| $F,\mu$ | Constants used in Assumptions 1 and 2 |
| $\sigma_l^T,\sigma_{h,k}^T,\sigma$ | Internal cost of a link, a node resource, or cost function of a provisioning scheme |
| $\xi(l,T),\xi(h,k,T)$ | Utilization ratio of a link or node resource |

can be regarded as different applications; we use $j$ to index requests. Hereafter we use $\mathbb{R}^+$ and $\mathbb{R}^*$ to denote positive and non-negative real numbers respectively, and $[\mathbf{X}]$ to denote the set $\{1,2,\ldots,\mathbf{X}\}$. Each SLA request $A_j\in\mathcal{A}$ is represented by a 4-tuple: $A_j=(\mathcal{S}_j,B_j,R_j,\mathbf{D}_j)$, where $\mathcal{S}_j\subseteq\mathcal{S}$ is the set of *source APs* from which data sources are connected, $B_j:\mathcal{S}_j\mapsto\mathbb{R}^+$ denotes the demand (data generation rate) entering from each source AP, $R_j=(r_1^j,\ldots,r_{\mathbf{K}}^j)$ denotes the resource needed to process unit demand for $A_j$, and $\mathbf{D}_j\in\mathbb{R}^+$ is the application-wide delay bound for all data sources[1]. We use $\mathbf{B}_j\triangleq\sum_{s\in\mathcal{S}_j}B_j(s)$ to denote the total input demand from all source APs. Our model is similar to that in [5]; the difference is that we explicitly model the computing resource requirement of an application with $R_j$.

**Example 1 (Social VR/AR):** A social VR/AR application (*e.g.*, AR games like Pokémon GO [1] or VR teleconferencing with Horizon Workrooms [2]) processes real-time data such as user actions or video captures at a central server, and then sends processed states to user devices distributed in the network. A game or meeting may last for a period of time, during which strict throughput and real-time bounds need to be satisfied to provide acceptable VR/AR experiences. A short-term SLA lease can provide guaranteed performance paid by the users, or the AO based on user membership statuses. Demands, resources and delay are set based on each user's update frequency, data size, and latency bound for VR/AR.

**Example 2 (Mobile Advertisement):** Location-based mobile advertisement [3] dynamically pushes ad content to users' mobile devices based on location. Central or distributed servers continuously process updates on user locations and preferences to accurately recommend stores near each user's location. During peak hours (*e.g.*, weekends or special sales events), short-term edge computing leases can help ad providers reach more users faster, hence increasing revenue. In this case, an ad provider leases bandwidth/resources based on estimated or desired number of customers to track and push to, and sets delay bounds based on type of ad and user mobility.

[1]QoS requirements other than throughput and delay can also be considered, such as cost, reliability, delay jitter, etc. Our mechanism and analysis can be extended to any additive QoS attribute, powered by existing approximation algorithms for Multi-Constrained Path [32] to replace the approximate DCLC-Path used in our Algorithm 2. We omit the details due to page limit.

### C. SLA Provisioning Model

To fulfill an SLA, the EP needs to 1) host the application on one or more edge nodes with sufficient resources, and 2) reserve communication channels from sources to application instance(s) with throughput and delay guarantee. Let $p$ be a path, and define $d_p=\sum_{l\in p}d_l$ as the delay of $p$. We then define the *feasible path set* from a source $s\in\mathcal{S}_j$ to an arbitrary edge node $h\in\mathcal{F}$, as $\mathcal{P}_{s,h}^j\triangleq\{p\,|\,p\text{ is an }(s,h)\text{-path}, d_p\le\mathbf{D}_j\}$ for each request $j$; we use $\mathcal{P}_s^j\triangleq\bigcup_{h\in\mathcal{F}}\mathcal{P}_{s,h}^j$ to denote feasible paths from $s$ to all edge nodes, and $\mathcal{P}^j\triangleq\bigcup_{s\in\mathcal{S}_j}\mathcal{P}_s^j$ to denote such paths from all sources. Then we can define an allocation, or a *provisioning scheme*, of an SLA request:

**Definition 1.** *Given $\mathcal{G}$, a* ***provisioning scheme*** *for request $A_j$ is defined as $\psi_j\triangleq(P_j,f_j)$, where $P_j=\bigcup_s P_s^j$, $P_s^j\subseteq\mathcal{P}_s^j$ is a subset of paths selected for source $s\in\mathcal{S}_j$, and $f_j:P_j\mapsto\mathbb{R}^*$ is the corresponding* demand allocation *on each path.* □

**Remark:** Definition 1 defines basic elements of the EP's provisioning plan for an SLA request. Path set $P_j$ defines 1) the edge node(s) selected to deploy the application (destination(s) of the paths), and 2) data routes from sources to the edge node(s). The function $f_j$ further defines the demand allocated to each path, and implicitly the input demand to be processed by each edge node. For simplicity, we can expand $f_j(\cdot)$ to $\mathcal{P}^j$ and let $f_j(p)=0$ for $p\in\mathcal{P}^j\backslash P_j$. Hence $P_j$ is implicitly defined as $P_j=\{p\in\mathcal{P}^j\,|\,f_j(p)>0\}$. We further define auxiliary functions $x_j:\mathcal{S}_j\times\mathcal{F}\mapsto[0,1]$ where $x_j(s,h)=\frac{1}{B_j(s)}\sum_{p\in\mathcal{P}_{s,h}^j}f_j(p)$ is the fraction of input demand from source $s\in\mathcal{S}_j$ allocated to edge node $h\in\mathcal{F}$ for processing, and $y_j:\mathcal{F}\mapsto[0,1]$ where $y_j(h)=\frac{1}{\mathbf{B}_j}\sum_{s\in\mathcal{S}_j}B_j(s)x_j(s,h)$ is the fraction of total input demand allocated to $h$. Note that $x_j$ and $y_j$ can be computed from $\psi_j$, and hence are used only for explanation. We call a node $h\in\mathcal{F}$ a *selected node* when $y_j(h)>0$. □

A typical provisioning scenario is in Fig. 1. A traditional application is usually deployed as a central entity processing all input streams. This aligns with the centralized computing provided by the cloud. Recent advances in *serverless* and *network function virtualization* enable distributed processing at

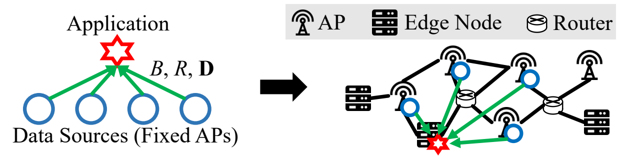


Fig. 1: Basic application provisioning in the edge network. Source APs are fixed, but routing and edge node are flexible.

the network edge, with improved flexibility and performance. We thus consider three scenarios, some also studied in [5]:

- **Asynchronous:** An asynchronous application can be deployed as multiple distributed instances, each processing a fraction of demand from any source. Examples include stateless sensor fusion and analytics, serverless applications, or virtual network functions such as intrusion detection. A *feasible* provisioning scheme for an asynchronous application is simply one that satisfies all the demands:
$$\sum_{h\in\mathcal{F}} y_j(h) \geq 1; \quad \text{or} \quad \sum_{h\in\mathcal{F}} x_j(s,h) \geq 1, \forall s \in \mathcal{S}_j. \tag{1}$$
- **Synchronous:** A synchronous application can be distributed over multiple instances as well, but requires each instance to process the same portion of demands from all sources at any time. For instance, a social-aware mobile AR app needs to receive and process simultaneous data generated from multiple locations at the same instance, but data generated at different times may be distributed across instances. Eq. (2) combined with Eq. (1) expresses the feasibility criteria for a synchronous application:
$$y_j(h) = x_j(s,h), \quad \forall h \in \mathcal{F}, s \in \mathcal{S}_j. \tag{2}$$
- **Centralized:** A centralized application has to be implemented as a single instance on one edge node. Most traditional applications fall into this category. An integrality constraint as follows plus Eq. (1) expresses the feasibility criteria for a centralized application:
$$y_j(h) \in \{0,1\}, \quad \forall h \in \mathcal{F}. \tag{3}$$

## IV. EA-Market: Online Mechanism Design

### A. Online Mechanism Preliminaries

We consider an *online resource auction* scenario from the view of the EP. The SLA requests $\mathcal{A} = \{A_1, \ldots, A_\mathbf{J}\}$ arrive over time. Each request belongs to one of three categories: $\mathcal{A}_{\mathsf{async}}$ for asynchronous applications, $\mathcal{A}_{\mathsf{sync}}$ for synchronous applications, and $\mathcal{A}_{\mathsf{cent}}$ for centralized applications. An AO (bidder) submits request $A_j$ along with a tuple $\Gamma_j = (\gamma_j, T_j^{\mathsf{start}}, T_j^{\mathsf{end}})$, where $\gamma_j > 0$ is the *bid* value, and $T_j^{\mathsf{start}}$ and $T_j^{\mathsf{end}}$ are the arrival (start) and departure (end) times of the requested service, respectively. Essentially, the AO indicates that she is willing to pay up to $\gamma_j$ to have guaranteed throughput and delay between $T_j^{\mathsf{start}}$ and $T_j^{\mathsf{end}}$. For simplicity, we assume a system with uniform time slots $\mathcal{T} = (1, 2, \ldots, \mathbf{T})$; this is not a restriction of our design, which can be trivially extended to the case with continuous time, as shown in our simulations. We use $T_j$ to denote an SLA request's duration, $T_{\mathsf{max}} = \max_j\{T_j\}$ to denote the maximum duration, and $\tau(T,j) \in \{0,1\}$ to indicate if $A_j$ is *active* in time $T$.

We assume source APs and demands are static during the duration of an SLA request. User mobility is handled by the APs seamlessly, and adding resource redundancy at near-by APs for mobility handling is at the discretion (and cost) of the AO. The AO knows the *amount* of resources that she needs to request to serve the expected demands from all sources, and/or to handle potential demand surges. The resources needed to serve unit demand can be obtained via profiling in a controlled environment.

Each AO asks for an *all-or-nothing* deal: either she pays no more than $\gamma_j$ to have guarantees for all demands, or she pays 0 for no reservation at all; partial provisioning is not acceptable due to potential fairness issues when some users are served with guarantees but others are not. The AO also asks for an immediate decision, *i.e.*, EP must either accept or reject the request at once without waiting for other requests to arrive.

Due to scarcity and costs of edge resources, the EP runs a resource auction to allocate computing and network resources and serve requests with the highest values. We assume each AO has a private value $v_j$ for her request, which may differ from her bid $\gamma_j$, and is independent from AOs' requests or valuation. Let outcome of the auction be denoted by $Z = \{\zeta_j \in \{0,1\} \mid A_j \in \mathcal{A}\}$ (reject or accept for each $A_j$), and let the price each AO needs to pay be denoted by $\Pi = \{\pi_j \geq 0 \mid A_j \in \mathcal{A}\}$. The *utility* of an AO is defined as:
$$u_j(\zeta_j, \pi_j) \triangleq (v_j - \pi_j) \cdot \zeta_j. \tag{4}$$
The *utility* of the auctioneer (EP) is defined as:
$$u_0(Z, \Pi) \triangleq \sum\nolimits_{A_j \in \mathcal{A}} \pi_j \cdot \zeta_j. \tag{5}$$
The *social welfare* is then the sum of utilities of all parties, which is also the aggregate value of all accepted SLA request:
$$S(Z,\Pi) \triangleq \sum_{A_j\in\mathcal{A}} u_j(\zeta_j,\pi_j) + u_0(Z,\Pi) = \sum_{A_j\in\mathcal{A}} v_j \cdot \zeta_j. \tag{6}$$
Note that the social welfare is independent of the payments $\Pi$, and hence we will write it simply as $S(Z)$ subsequently.

The EP has the following goals as the *auctioneer*:

- **Social welfare:** The EP seeks to maximize social welfare $S(Z)$. Social welfare maximization has been regarded as a way of maximizing long-term user satisfaction and thus revenue, which suits the EP's role as a long-term infrastructure provider. It has been used in other similar scenarios such as Facebook's advertisement auction [33].
- **Truthfulness:** An auction is truthful iff no bidder can increase her utility by bidding $\gamma_j \neq v_j$, *i.e.*, bidding $\gamma_j = v_j$ is a *dominant strategy* of any AO. Truthfulness is critical because 1) it simplifies each AO's strategy to simply reporting its true valuation, and 2) it allows the EP to maximize its objective (social welfare in our case) without guessing the true valuations of the AOs.
- **Individual rationality, budget balance:** In participating, each AO requires her utility to be non-negative (individual rationality). Likewise, the EP cannot have a negative utility for running the auction (budget balance).
- **Computational efficiency:** The auction should run in polynomial time for both the EP and each AO.

### B. Offline Social Welfare Maximization

We first define the offline social welfare maximization problem, assuming all requests and bids are known and truthful.

**Definition 2.** *Given network $\mathcal{G}$ and SLA requests $\mathcal{A}$, the* offline social welfare maximization *problem can be formulated as:*

$$\max_{\Phi,Z} \quad S(Z) = \sum\nolimits_j v_j \cdot \zeta_j \tag{7}$$

$$s.t. \quad \sum\nolimits_{h\in\mathcal{F}} x_j(s,h) \geq \zeta_j, \quad \forall j, s \in \mathcal{S}_j; \tag{7a}$$

$$\sum_j \sum_{p\in\mathcal{P}_l\cap\mathcal{P}^j} \tau(T,j)\cdot f_j(p) \leq b_l, \quad \forall l\in\mathcal{L}, T\in\mathcal{T}; \tag{7b}$$

$$\sum\nolimits_j \tau(T,j)\cdot \mathbf{B}_j \cdot r_k^j \cdot y_j(h) \leq c_k^h, \quad \forall h\in\mathcal{F}, k\in[\mathbf{K}], T\in\mathcal{T}; \tag{7c}$$

$$(2) \text{ for } \forall A_j \in \mathcal{A}_{\mathsf{sync}}; \qquad (3) \text{ for } \forall A_j\in\mathcal{A}_{\mathsf{cent}}. \tag{7d}$$

*Here $\Phi = \{f_j \,|\, A_j \in \mathcal{A}\}$ and $\mathcal{P}_l = \{p \in \bigcup_j \mathcal{P}^j \,|\, l\in p\}$, and recall that $x_j(s,h) = \frac{1}{B_j(s)}\sum_{p\in\mathcal{P}_s^j} f_j(p)$ and $y_j(h) = \frac{1}{\mathbf{B}_j}\sum_{s\in\mathcal{S}_j} B_j(s)x_j(s,h)$ as defined in Sec. III-C.* □

**Explanation:** Objective maximizes social welfare in Eq. (6). Constraint (7a) echos Eq. (1) for every accepted SLA request. Constraints (7b) and (7c) limit the demand carried on each link and processed at each node by bandwidth and computing capacities. Constraint (7d) applies based on application's category. The program does not involve the payments $\Pi$, which will be addressed independently to ensure truthfulness.

**Theorem 1.** *Maximizing social welfare is NP-hard.* □

*Proof.* The hardness is multi-fold. First, with one edge node $h \in \mathcal{F}$ and one SLA request $A_j \in \mathcal{A}$ with one data source, the problem becomes Multi-Path routing with Bandwidth and Delay constraints (MPBD), which is NP-hard [34]. Second, when the network has infinite bandwidth, the accept/reject problem can be viewed as a 0-1 Multi-Dimensional Knapsack (MDK) problem, which is NP-hard and even APX-hard [35]. Third, when the applications are centralized, with just one type of computational resource and infinite bandwidth, the problem is a 0-1 Multiple Knapsack (MK) problem, which is again NP-hard [36]. The combined problem generalizes all three. □

## C. Online Auction for Edge Provisioning

In the online scenario, EP has no knowledge of requests arriving in the future. We utilize the widely adopted *competitive analysis* framework to design and theoretically analyze our mechanism. Let $S^{\mathsf{opt}}$ be the maximum *offline* social welfare.

**Definition 3.** *A $\theta$-*competitive mechanism *achieves at least $S^{\mathsf{opt}}/\theta$ in social welfare, while satisfying all constraints.* □

We design an online mechanism with a polylogarithmic ratio in social welfare. We follow a primal-dual framework [37], [38], which maintains an *internal cost* for each resource (including bandwidth). Cost is set exponentially to level of resource utilization when a request arrives. To set the cost, we make two assumptions on demands and AOs' valuations.

**Assumption 1.** *$(|\mathcal{N}| + \mathbf{K})\mathbf{B}_j \leq \gamma_j/T_j \leq FB_j(s)$, $\forall A_j \in \mathcal{A}, s\in\mathcal{S}_j$, where $F$ is a constant.* □

**Assumption 2.** *$\mathbf{B}_j \leq \min\{\{b_l \,|\, l\in\mathcal{L}\} \cup \{c_k^h/r_k^j \,|\, h\in\mathcal{F}, k\in[\mathbf{K}]\}\}/\log\mu$, $\forall A_j \in \mathcal{A}$, where $\mu \triangleq 2(FT_{\mathsf{max}}+1)$.* □

**Algorithm 1:** EA-Market: Online Provisioning Mechanism

```
Input: Network $\mathcal{G}$, arriving SLA request $A_j$
Output: Decision $\zeta_j$, provisioning scheme $\psi_j = (P_j, f_j)$,
        payment $\pi_j$
1 Set $\sigma_l$ and $\sigma_{n,k}$ based on current $\xi(l,T)$ and $\xi(h,k,T)$;
2 $(P_j^*, f_j^*) \leftarrow \arg\min\{\sigma(P_j,f_j)\}$;  // Algorithm 2
3 if $(P_j^*, f_j^*) \neq (\emptyset,\emptyset)$ and $\sigma(P_j^*, f_j^*) \leq \gamma_j$ then
4     for $\forall p \in P_j^*$, and $\forall T \in [T_j^{\mathsf{start}}, T_j^{\mathsf{end}}]$ do
5         $\xi(l,T) \leftarrow \xi(l,T) + f_j^*(p)/b_l$, $\forall l \in p$;
6         $\xi(h_p,k,T) \leftarrow \xi(h_p,k,T) + r_k^j \cdot f_j^*(p)/c_k^{h_p}$, $\forall k \in [\mathbf{K}]$;
7     return Accept ($\zeta_j = 1$), $\psi_j = (P_j^*, f_j^*)$, $\pi_j = \sigma(P_j^*, f_j^*)$.
8 return Reject ($\zeta_j = 0$), $\psi_j = (\emptyset,\emptyset)$, $\pi_j = 0$.
```

Assumption 1 assumes that the number of sources of a request and the per-unit time-demand valuation of each AO are bounded. Assumption 2 assumes that the demand of *each* request is relatively small compared to resources or link bandwidth in the network. $\mu$ is used below to set the cost of each link or node resource, and later used to reason about the competitiveness of the online mechanism.

**Remark:** Constants $F$ and $T_{\max}$ can be estimated from past request data and dynamically adjusted (see Sec. IV-F). □

When an SLA request $A_j$ arrives at the time $T_j^{\mathsf{start}}$, for any $T \in [T_j^{\mathsf{start}}, T_j^{\mathsf{end}}]$, let $\xi(l,T)$ and $\xi(h,k,T)$ be the *resource utilization ratio* for link $l$'s bandwidth and node $h$'s type-$k$ resource, respectively, defined as the consumption of all accepted and active SLAs over the corresponding capacity at time $T$. Initially all $\xi(l,0)$ and $\xi(n,k,0)$ are set to 0 when no SLA request has arrived. The **internal cost** of $l \in \mathcal{L}$ for time slot $T$ at $A_j$'s arrival is then set as

$$\sigma_l^T = b_l \cdot (\mu^{\xi(l,T)} - 1), \tag{8}$$

and the **cost** of type-$k$ resource on node $h \in \mathcal{F}$ at time $T$ is

$$\sigma_{h,k}^T = c_k^h \cdot (\mu^{\xi(h,k,T)} - 1). \tag{9}$$

From a primal-dual viewpoint, internal costs $\sigma_l^T$ and $\sigma_{h,k}^T$ are dual variables of constraints (7b) and (7c). In general, our algorithm utilizes the internal (dual) costs to enforce (7b) and (7c), while relying on a primal method to enforce the rest constraints in Program (7). As the dual of (7) is only used to enforce (7b)–(7c) in our mechanism, we omit describing the entire dual program, and refer the reader to [37], [38] for general descriptions of the primal-dual framework.

Let $(P_j, f_j)$ be a provisioning scheme for $A_j$. The cost of $(P_j, f_j)$ is defined as the total cost of all consumed bandwidth and computing resources over $A_j$'s entire active period:

$$\sigma(P_j, f_j) \triangleq \sum_{T\in[T_j^{\mathsf{start}}, T_j^{\mathsf{end}}]} \sum_{p\in P_j} f_j(p) \left( \sum_{l\in p} \frac{\sigma_l^T}{b_l} + \sum_k \frac{\sigma_{h_p,k}^T \cdot r_k^j}{c_k^{h_p}} \right), \tag{10}$$

where $h_p \in \mathcal{F}$ is the edge node that path $p$ leads to. Let $\sigma_j(p) \triangleq \sum_{T\in[T_j^{\mathsf{start}}, T_j^{\mathsf{end}}]} \left( \sum_{l\in p} \frac{\sigma_l^T}{b_l} + \sum_k \sigma_{h_p,k}^T \cdot \frac{r_k^j}{c_k^{h_p}} \right)$ be the cost of path $p$ summed over the entire active period of $A_j$.

Our algorithm for processing an in-coming request is shown in Algorithm 1. When an SLA request arrives, the EP tries to find a provisioning scheme (satisfying Eq. (1) for any application type, and (2) for synchronous or (3) for centralized) with the minimum cost (Line 2). The request is accepted if such a provisioning scheme exists and has cost $\sigma(P_j^*, f_j^*)$ bounded by the AO's bid $\gamma_j$, and the payment of the AO is set to exactly $\sigma(P_j^*, f_j^*)$; otherwise, the request is rejected and the AO pays 0. Then, the utilization ratios are updated with the accepted SLA request's provisioning scheme within its entire duration.

## *D. Approximation Scheme for One-Shot Provisioning*

The core step in Algorithm 1 is to find a provisioning scheme satisfying the cost bound (Line 2). This one-shot step turns out to be non-trivial as well. For simplicity, we assume working on *asynchronous applications* hereafter (without the constraints in Eq. (2) or (3)), until Sec. IV-F where we extend our result to synchronous and centralized cases. Let us formulate this step:

$$\min_{f_j \geq 0} \quad \sigma(\mathcal{P}^j, f_j) = \sum\nolimits_{p \in \mathcal{P}^j} f_j(p) \cdot \sigma_j(p) \tag{11}$$

$$\text{s.t.} \quad \sum\nolimits_{h \in \mathcal{F}} x_j(s,h) \geq 1, \quad s \in \mathcal{S}_j. \tag{11a}$$

Note that Program (11) *does not involve capacity constraints*, since those are enforced implicitly via the internal cost function as will be shown later. Below, we present a simple yet important lemma that is at the core of our design and analysis:

**Lemma 1.** *If Program* (11) *is feasible, then there is always an optimal solution to* (11) *with only one path $p \in \mathcal{P}_s^j$ that has positive $f_j(p)$ for each $s \in \mathcal{S}_j$.* ☐

*Proof.* An optimal solution routes all demand from $s \in \mathcal{S}_j$ to any $h \in \mathcal{F}$ via a minimum-cost (*w.r.t.* $\sigma_j(p)$), delay-bounded path. If multiple min-cost paths exists for an $s$, one can always shift all demands onto one path to satisfy Lemma 1. ☐

**Remark:** The shifting does not violate any capacity, which is guaranteed by the dual costs as shown in Sec. IV-E. ☐

**Theorem 2.** *Program* (11) *is NP-hard.* ☐

*Proof.* From Lemma 1, solving (11) is equal to finding one path for each source, such that the total demand-weighted cost of all paths is minimized. With just one $s \in \mathcal{S}_j$ and one $h \in \mathcal{F}$, this is equal to finding a Delay-Constrained Least Cost (DCLC) path, which is NP-hard [32]. Theorem 2 follows. ☐

The best known algorithm for DCLC is a *fully polynomial-time approximation schemes (FPTAS)* [32] which guarantees a $(1+\epsilon)$-approximation of the DCLC path with an arbitrarily small $\epsilon$ and has time complexity polynomial to the input size and $1/\epsilon$. Utilizing this, we can find an approximate solution to Program (11), by finding an approximate DCLC path for each source $s \in \mathcal{S}_j$ to carry its entire demand. The approximation scheme is shown in Algorithm 2. DCLC-Path implements the FPTAS from [32] for finding an approximate DCLC path that is within $(1+\epsilon)$ times the minimum cost, while strictly satisfying the delay constraints on the paths. The subroutine runs in $O(|\mathcal{N}||\mathcal{L}|(\log\log\log|\mathcal{N}| + 1/\epsilon))$ time. Algorithm 2 thus runs in $O(\max_j\{|\mathcal{S}_j|\}|\mathcal{N}||\mathcal{L}|(\log\log\log|\mathcal{N}|+1/\epsilon))$ time. The following theorem is intuitive based on the FPTAS subroutine.

**Algorithm 2:** FPTAS to Program (11)

```
Input: Network $\mathcal{G}$, application $A_j$, costs $\{\sigma\}$, tolerance $\epsilon$
Output: Provisioning scheme $\psi_j = (P_j, f_j)$
1 Construct $\mathcal{G}'$ by adding node $\hat{h}$ that is connected from
    every edge node $h \in \mathcal{F}$ with a 0-cost 0-delay link;
2 for $s \in \mathcal{S}_j$ do
3     $p \leftarrow$ DCLC-Path$(\mathcal{G}', s, \hat{h}, \mathbf{D}_j, \{\sigma\}, \epsilon)$;
4     if $p$ does not exist then return $(\emptyset, \emptyset)$;
5     $P_j \leftarrow P_j \cup \{p\}$, and $f_j(p) \leftarrow B_j(s)$;
6 return $\psi_j = (P_j, f_j)$.
```

**Theorem 3.** *Algorithm 2 is an FPTAS to Program* (11). ☐

Inserting Algorithm 2 into Algorithm 1 (Line 2) yields a well-defined online mechanism, which we next analyze.

## *E. Analysis of the Online Mechanism*

**Theorem 4.** *Algorithm 1 combined with Algorithm 2 yields a $(2(1+\epsilon)\log\mu + 1)$-competitive algorithm.* ☐

*Proof.* We first prove feasibility. For any link or node resource, let us use $c$, $\xi$, $\sigma$, and $r$ to denote its bandwidth or capacity, utilization before allocation, cost, and the newly allocated resource for any time $T \in [T_j^{\mathsf{start}}, T_j^{\mathsf{end}}]$ when an arbitrary $A_j$ arrives; for type-$k$ node resource, we assume $c$ and $r$ are both scaled by $1/r_k^j$ such that they are of the same unit as bandwidth. Based on the way Algorithm 2 pushes the demands, we have $\min_s\{B_j(s)\} \leq r \leq \mathbf{B}_j$. If the capacity $c$ is exceeded for this resource, it means $\xi + r/c > 1$. By Assumption 2, $\xi > 1 - 1/\log\mu$. Then, $\sigma = c(\mu^\xi - 1) > c(\mu/2 - 1) = cFT_{\mathsf{max}}$. The cost on this resource is $r\sigma/c > rFT_{\mathsf{max}} \geq \gamma_j$ (by Assumption 1), which contradicts our acceptance criterion.

We next prove competitiveness. We first derive a lower bound on the online social welfare, which we denote as $S_{\mathsf{onl}}$, using the final costs of link and node resources. Consider a link or node resource, let $\sigma^-(T,j)$ and $\sigma^+(T,j)$ be its cost at time $T$ before and after accepting an SLA request $A_j$ respectively, and let $\sigma^*(T,j) = \sigma^+(T,j) - \sigma^-(T,j)$. Based on how the cost is computed, we have $\sigma^*(T,j) = c(\mu^\xi(\mu^{r/c} - 1)) = c(\mu^\xi(2^{(\log\mu)r/c} - 1))$. Since $r \leq \mathbf{B}_j \leq c/\log\mu$ by Assumption 2, we have $2^{(\log\mu)r/c} - 1 \leq (\log\mu)r/c$, and hence $\sigma^*(T,j) \leq \mu^\xi r\log\mu = (\sigma^-(T,j)r/c + r)\log\mu$. Now we sum up $\sigma^*(T,j)$ for all resources and all time slots (we use subscript to denote each link/node resource), and we have

$$\sum\nolimits_T \left(\sum\nolimits_l \sigma_l^*(T,j) + \sum\nolimits_n \sum\nolimits_k \sigma_{n,k}^*(T,j)\right)$$

$$\leq \log\mu\left(\gamma_j + \sum\nolimits_T \sum\nolimits_{p \in P_j} f_j(p)(|p| + \mathbf{K})\right) \leq 2\gamma_j\log\mu.$$

The first inequality above is due to the accepted SLA request having total cost no more than the bid $\gamma_j$; the second inequality is due to the left-hand-side of Assumption 1. Summing up for all accepted SLA requests (denoted as $\mathcal{A}_{\mathsf{onl}}$), we have

$$\sum_{A_j \in \mathcal{A}_{\mathsf{onl}}} \gamma_j \geq \frac{1}{2\log\mu} \sum_T \left(\sum_l \sigma_l^+(T,\mathbf{j}) + \sum_n \sum_k \sigma_{n,k}^+(T,\mathbf{j})\right), \tag{12}$$

where $\mathbf{j}$ is the last accepted SLA request index in $\mathcal{A}_{\mathsf{onl}}$.

We finally show that the offline optimal social welfare can be (upper) bounded by the same final costs. Let $\mathcal{A}_{\mathsf{opt}}$ be the set of requests accepted by the offline optimal solution, and let $\mathcal{A}_{\mathsf{diff}} \triangleq \mathcal{A}_{\mathsf{opt}} \setminus \mathcal{A}_{\mathsf{onl}}$. For any $A_j \in \mathcal{A}_{\mathsf{diff}}$, let $(P_j^{\mathsf{opt}}, f_j^{\mathsf{opt}})$ be its provisioning scheme in the optimal solution. The facts that $A_j$ is not accepted, and that Algorithm 2 computes a $(1+\epsilon)$-approximate minimum-cost provisioning scheme, yield that $\gamma_j \le (1+\epsilon)\sigma(P_j^{\mathsf{opt}}, f_j^{\mathsf{opt}})$ when $\sigma$ is computed at its arrival, which, because per-time slot costs monotonically increase with every arriving request, is further bounded by $(1+\epsilon)\sigma(P_j^{\mathsf{opt}}, f_j^{\mathsf{opt}})$ when $\sigma$ is computed using the final costs after arrival of the last request $A_{\mathbf{j}} \in \mathcal{A}_{\mathsf{onl}}$. Let $\sigma(P_j, f_j, T, \mathbf{j}) = \sum_k \left(\sigma^+_{h_p,k}(T,\mathbf{j})\frac{f_j(p)r_k^j}{c_k^n}\right) + \sum_{l\in p}\left(\sigma_l^+(T,\mathbf{j})\frac{f_j(p)}{b_l}\right)$ be the cost of $(P_j, f_j)$ at time $T$, using the final costs after accepting $A_{\mathbf{j}}$. Summing up for $A_j \in \mathcal{A}_{\mathsf{diff}}$, we have

$$\sum_{A_j\in\mathcal{A}_{\mathsf{diff}}}\gamma_j \le (1+\epsilon)\sum_{A_j\in\mathcal{A}_{\mathsf{diff}}}\sum_T \sigma(P_j^{\mathsf{opt}}, f_j^{\mathsf{opt}}, T, \mathbf{j})$$
$$\le (1+\epsilon)\sum\nolimits_T\Big(\sum\nolimits_l \sigma_l^+(T,\mathbf{j}) + \sum\nolimits_n\sum\nolimits_k \sigma^+_{n,k}(T,\mathbf{j})\Big), \tag{13}$$

where the second inequality is because the offline optimal solution cannot exceed any link bandwidth or node resource capacity constraint. Finally, combining Eqs. (12) and (13), we can conclude that $S^{\mathsf{onl}} = \sum_{A_j\in\mathcal{A}_{\mathsf{onl}}}\gamma_j \ge \sum_{A_j\in\mathcal{A}_{\mathsf{opt}}}\gamma_j/(2(1+\epsilon)\log\mu+1) = S^{\mathsf{opt}}/(2(1+\epsilon)\log\mu+1)$. The theorem follows from the definition of the competitive ratio in Definition 3. □

Corollary 1 is by-product of the above proof, showing that $F$ is a *conservativeness parameter* that can be tuned for over-provisioning (Sec. IV-F). Proof is omitted due to page limit.

**Corollary 1.** *If constant $F$ in Assumption 1 is scaled by $1/\varepsilon$ for $\varepsilon \ge 1$, the maximum capacity violation is $(1+\log\varepsilon)$ times, while the competitive ratio is scaled by $(\varepsilon+1)/2\varepsilon$.* □

**Theorem 5.** *Algorithm 1 is truthful, individually rational, budget-balanced, and computationally efficient.* □

*Proof.* We prove these properties in several steps.

*Truthfulness:* Algorithm 2 and payment $\pi_j = \sigma(P_j^*, f_j^*)$ (if accepted) are unrelated to the bid $\gamma_j$. If an AO $i$ bids truthfully as $\gamma_i = v_i$ and request $A_i$ is *accepted*, her utility is $u_i(1,\pi_i) = (v_j - \sigma(P_i^*, f_i^*)) \ge 0$ as in Eq. (4). If she instead bids $\gamma_i^+ > v_i$, $A_i$ is still accepted with the same $\pi_j = \sigma(P_j^*, f_j^*)$, leading to the same utility based on Eq. (4). This also holds if she bids lower and still gets accepted. Now, if the AO bids lower and gets rejected, her utility $u_i(1,\pi_i)$ becomes 0 as $\zeta_i$ becomes 0 in Eq. (4). Hence an AO who is accepted with a truthful bid cannot improve utility by bidding higher or lower.

Now, if an AO $j$ bids $\gamma_j = v_j$ and is *rejected*, her utility is 0, and we know $v_j < \sigma(P_j^*, f_j^*)$. If she raises bid to $\gamma_j^+ > \gamma_j$, she may still get rejected and receive 0 utility; but if she gets accepted, her utility will become $u_j(1,\pi_j) = (v_j - \sigma(P_j^*, f_j^*)) < 0$, thus lowering her utility from 0. If the AO bids lower, she still gets rejected and still receives 0 utility. Hence an AO who is rejected when bidding truthfully cannot improve her utility by either raising or lowering her bid as well. Combining the above proves the truthfulness property.

*Individual rationality and budget balance:* As long as an AO bids truthfully, its utility is always non-negative by the acceptance and payment rules. The payment is always non-negative, so the utility of the EP is also non-negative.

*Computational efficiency:* Though our formulation has $\mathbf{T}$ time slots, in practice only a linear number of (non-uniform) slots are needed, as costs only change upon request arrival or departure; hence the input size to Algorithm 2 is polynomial (rather than pseudo-polynomial in $\mathbf{T}$). Algorithm 2 runs in time polynomial to input size and $1/\epsilon$ [32]. This, combined with Algorithm 1, yields a polynomial-time complexity. □

## F. Centralized and Synchronous Applications

So far, Algorithm 2 and subsequently Theorem 4 only apply to asynchronous applications. In this subsection, we show how to extend the results to centralized and synchronous applications, by addressing Constraints (3) and (2), respectively.

*1) Centralized Applications:* For a centralized application, we need to pick exactly one edge node $h \in \mathcal{F}$ for each SLA request $A_j$, and route demands from all sources of $A_j$ to this node. We first present the following observation:

**Observation 1.** *Lemma 1 holds for a centralized application.*

Based on Constraint (3), an optimal one-shot solution is to find an edge node, which has the minimum traffic-weighted cost of a delay-bounded reverse path tree. As such, we change Algorithm 2 from finding individual paths to any edge node, to finding such a tree to each $h \in \mathcal{F}$. Using the same DCLC FPTAS from [32], we can approximate the minimum-cost tree of each $h \in \mathcal{F}$, and then pick one with the minimum cost. Adding this to Algorithm 1 gives the same competitive ratio, and with at most $|\mathcal{F}|$ times the complexity.

*2) Synchronous Applications:* Regarding synchronous applications, we have the following observation:

**Observation 2.** *Any provisioning scheme for a synchronous application is the convex combination of up to $|\mathcal{F}|$ provisioning schemes for a centralized application with the same inputs.*

Combining Observations 1 and 2, it is clear that by the same argument as in the proof of Lemma 1, we can also shift all demands to be processed by only one edge node that has the minimum-cost delay-bounded path tree, and hence:

**Observation 3.** *Lemma 1 holds for a synchronous application.*

In other words, the algorithm for synchronous applications is exactly the same as the algorithm for centralized applications. By the same argument as above, the modified algorithm yields the same performance bound as the original algorithm.

## G. Discussions

**Tunable parameter.** Our main theorems are based on Assumptions 1 and 2, which may not hold in practice. Meanwhile, Corollary 1 shows that parameter $F$ can be regarded as a tunable *conservativeness parameter*, defining how conservative the EP is on estimating how much requests overlap in time and across different resources. The competitive analysis above assumes a worst-case scenario, where all requests are fully overlapping in time and must utilize the same resources. In reality, based on historical data, the EP can estimate bounds on the inter-arrival time of requests, as well as resources required by a typical request, and set a different (potentially much smaller) $F$ value than suggested by Assumptions 1 and 2. Note that by Corollary 1, setting a smaller $F$ value could lead to potential capacity violations, which must be handled carefully by rejecting the violating requests. In evaluation, we will show that setting a smaller $F$ based on the specific network scenario usually increases actual social welfare over the default value.

**Implications on offline mechanism design.** Comparing our mechanism to the *offline* provisioning algorithm in [5], they bear similar structures including an exponential cost function w.r.t. resource utilization. However, the offline algorithm in [5] is an FPTAS (for a slightly different objective), while our mechanism only achieves a polylogarithmic competitive ratio, due to that accommodated requests can no longer be adjusted or rejected in the online setting. Meanwhile, although our mechanism cannot achieve the same approximation guarantee as an offline FPTAS, it in fact provides *the best known theoretical guarantee for a* ***truthful offline mechanism*** for edge resource provisioning with a general edge network topology. Specifically, trivially turning an FPTAS into a mechanism was shown to violate truthfulness [39], and the design of truthful FPTAS mechanisms has so far been successful only on simple Knapsack problems and variants [39], with little success on more complex problems such as provisioning and orchestration. Our mechanism provides a simple and efficient offline mechanism with theoretical guarantee, when coupled with an arbitrary, bid-independent ordering of requests.

## V. Performance Evaluation

### A. Evaluation Method

We evaluated our mechanism with simulations. Each network scenario had 20 mesh-connected APs, with 5 edge nodes each attached to one AP. Links were generated in the Waxman model [40] with $\alpha = \beta = 0.6$, each with 1.2 Gbps capacity, and delay uniformly in $[10, 50]$ ms. Each edge node had $\mathbf{K} = 3$ types of resources (*e.g.*, CPU, GPU, memory), with normalized capacities uniformly in [3, 10] Gbps (in unit of in-coming data processing). In each experiment, 1000 SLA requests arrived in *continuous time*, following a Poisson distribution with rate $\lambda_{\mathsf{arr}}$, and the requested service times had rate $\lambda_{\mathsf{srv}}$. We used the ratio $\lambda = \lambda_{\mathsf{arr}}/\lambda_{\mathsf{svr}}$ to characterize the overall arrival and service process, and varied $\lambda$ to evaluate under different loads. By default $\lambda = 300$. Each request was randomly assigned to be asynchronous, synchronous, or centralized. Each request had 5 to 10 data sources, and a delay bound uniformly in $[25, 75]$ ms. Data rate of each source was uniformly in $[3, 10]$ Mbps. $F$ was computed based on Assumption 1, and valuations followed a normal distribution $N(\frac{\gamma_{\max}+\gamma_{\min}}{2}, \frac{\gamma_{\max}-\gamma_{\min}}{2})$ truncated by $[\gamma_{\min}, \gamma_{\max}]$, where $\gamma_{\max}$ and $\gamma_{\min}$ were upper and lower bounds of $\gamma_j$ respectively for each request in Assumption 1.

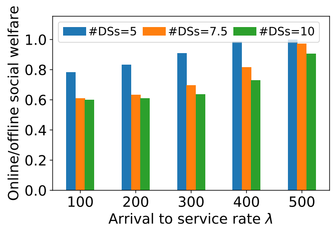
As even the one-shot problem is NP-hard, we compared our mechanism to upper bounding and heuristic solutions in Table II. SAP, RS and NS will accept any request that they have enough resources for upon arrival. We emphasize that *none of these algorithms (or any other known ones) except EA either guarantees* ***truthfulness*** *or has a* ***competitive ratio*** for QoS-aware provisioning in a general edge network. As such, we mainly evaluated in terms of a) *social welfare*, and b) *efficiency (running time)*. We set $\epsilon = 0.5$ in our mechanism. LPs were solved by the simplex method using Gurobi [41]. Simulations were done on a Linux PC with Octa-Core 3.6GHz CPU and 64GB memory. Each experiment was repeated for 20 times with random inputs to average-out random noise.

| | |
|---|---|
| **EA** | Our online mechanism in Algorithms 1 and 2. |
| **SAP** | *Single-Application Provisioning (SAP)* algorithm for **centralized** applications in [5], with high complexity, applied to each SLA request as an online heuristic. |
| **RS** | *Random Selection* heuristic where a random candidate edge node is selected for each SLA request, and then shortest path routing is repeatedly applied to every data source until sufficient bandwidth is provisioned. |
| **NS** | *Nearest Selection* heuristic where edge node with the minimum maximum distance from all data sources is selected, and then shortest path routing is repeatedly applied to every data source until enough bandwidth. |
| **ODA** | *Offline Delay-Agnostic* algorithm solving an edge-flow formulation (LP) that is the *delay-agnostic* version of the *linear relaxation* of Program (7), outputting an **upper bound** of the optimal social welfare. |

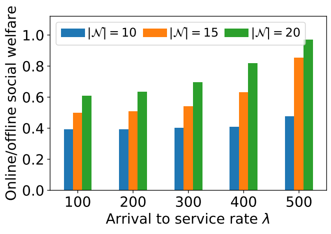
TABLE II: Implemented Algorithms

### B. Evaluation Results

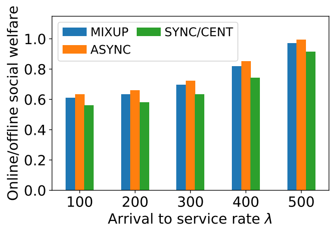
*1) Competitive Ratio Analysis:* In Fig. 2, we show results validating the competitiveness of our mechanism. We show the *online-offline ratio* as the ratio between the social welfare of an online algorithm (*e.g.* EA), and the upper bound of offline social welfare obtained by ODA. In all figures, increased load leads to higher online-offline social welfare ratio, since more competition helps fill the resource gap left by poor decisions early on. Fig. 2(a) shows that fewer data sources per request lead to higher online-offline ratio. This coincides with our competitive analysis based on $F$ in Assumption 1. In Fig. 2(b), more network nodes lead to improved online-offline ratio, due to increased resources and thus decreased relative load in the network. In Fig. 2(c), we show the impact of application types. The MIXUP algorithm processes each SLA request using its original type, while ASYNC and SYNC/CENT each regards all SLA requests as having the same type indicated by the algorithm name (note that synchronous and centralized cases have the same algorithm). ASYNC can achieve a better online-offline ratio than MIXUP or SYNC/CENT, due to flexibility in load balancing across edge nodes. Finally, in Fig. 2(d), we scale parameter $F$ by different factors $Fs$. Observe that $Fs = 1$, meaning $F$ fully satisfies Assumptions 1–2, is too conservative and leads to low social welfare, while $F$ scaled by $Fs = 10^{-4}$ can be too aggressive in over-provisioning and lowers social welfare as well. A proper $F$ can thus be picked based on historical requests to balance between conservativeness and over-provisioning. Based on this, in comparison with heuristics below, we scale $F$ by $Fs = 0.01$.



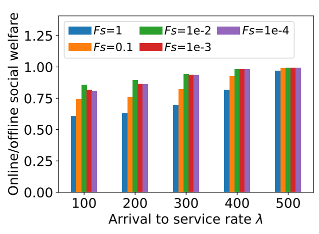
(a) Varying average # data sources



(b) Varying # network nodes $|\mathcal{N}|$

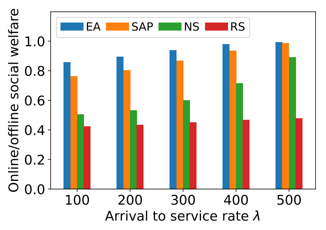


(c) Varying application types

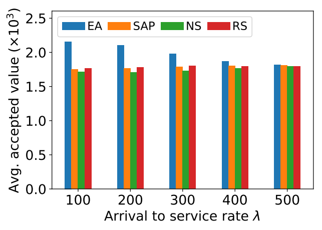


(d) Varying $F$ scaling factor $Fs$

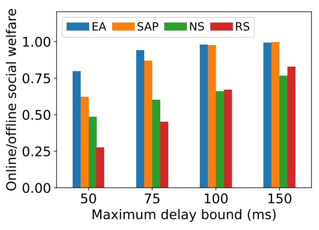
Fig. 2: Competitiveness analysis: online to offline social welfare ratio with varying load.



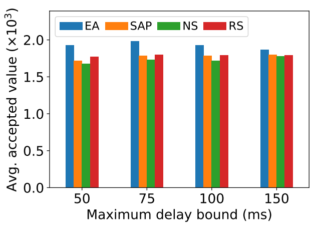
(a) Social welfare with varying load



(b) Average value with varying load



(c) Social welfare with varying maximum delay bound



(d) Average value with varying maximum delay bound

Fig. 3: Comparison between EA and heuristics, with varying load and delay bound.

*2) Comparison with Heuristics:* In Fig. 3, our mechanism is compared to heuristic solutions under varying loads and delay bounds. In Fig. 3(a), both EA and SAP heuristic constantly outperform NS and RS heuristics in terms of social welfare ratio. In Fig. 3(b), EA achieved higher average value per accepted request than all other heuristics, indicating serving SLA requests with high valuations. Average value decreases with increasing load, which corresponds to increasing social welfare in Fig. 3(a); this is because more requests are accepted when there are sufficient in-coming requests to make up for poor rejection decisions early on. Other heuristics serve SLA requests whenever there are available resources, and hence the average accepted values are lower and similar to each other (close to average valuation of all requests). The advantage of EA diminishes as load increases, when all algorithms' social welfare approaches the delay-agnostic upper bound (ODA).

In Figs. 3(c)–(d) the impact of delay bound is examined, where the delay bound of an SLA request is uniformly in $[25, \text{Max delay bound}]$ ms. All algorithms achieve higher social welfare as delay bound grows, as more requests become feasible when more edge nodes satisfy the delay bounds. EA holds the best performance especially when the delay bound is tighter, with $28\%$ higher social welfare at a maximum delay bound of $50$ms over SAP (the second best), suggesting its crucial advantage for requests with stringent QoS requirements. Their results converge at higher delay bounds since both can achieve almost the optimal social welfare (ODA upper bound). Keep in mind, though, that SAP can be **orders of magnitude** (such as $4000\times$) **slower** than EA as shown in Table III.

*3) Scalability:* Table III shows the running time comparison between two well-performing algorithms: EA and SAP. Both achieved high social welfare as in Fig. 3 although EA generally outperformed. As in Table III, SAP's running time was around $2000\times$ to $4000\times$ higher than EA in the given settings. We found this gap to further grow with larger arrival rate, more nodes, or a smaller $\epsilon$. This made it difficult to compare EA and SAP in larger-scale settings due to SAP's high running time. The result is not surprising, since SAP was designed as an offline algorithm and has a high complexity as shown in [5], several orders higher than EA asymptotically. The result precludes SAP to be used in situations where requests come with short intervals and need immediate accommodation.

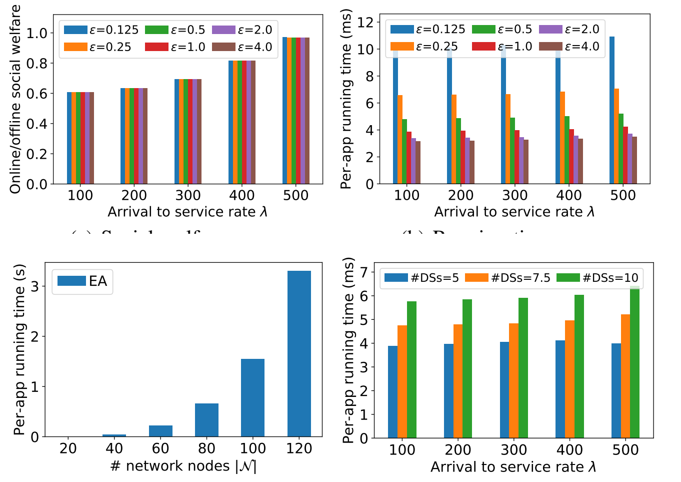
| $\lambda$ | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|
| **EA** | 5.62 | 5.66 | 5.68 | 5.62 | 5.55 |
| **SAP** | 10110.29 | 11658.24 | 14152.57 | 17681.29 | 21985.57 |

TABLE III: Running time per request (ms) for EA and SAP.

Fig. 4 shows impact of several scalability factors. Parameter $\epsilon$ defines trade-off between accuracy and complexity of one-shot FPTAS. While $\epsilon$ largely impacts running time as in Fig. 4(b), its impact on social welfare is negligible in Fig. 4(a). There are two reasons: 1) the theoretical bound $(1+\epsilon)$ of the FPTAS is very conservative, and 2) even if a request runs into a higher-than minimum cost and gets rejected, the unallocated resources can be used by future requests to make up for the poor decision. Hence, a loose $\epsilon$ value (*e.g.*, $1.0$) is enough for high efficiency without sacrificing social welfare. Fig. 4(c) examines running time with varying network sizes. With up to 120 network nodes (APs), EA still makes a decision in 3 seconds per request, fast enough for practical scenarios. Fig. 4(d) shows that the number of data sources has a limited impact on efficiency. Other factors such as number of edge nodes or resource types are similar and hence omitted.

## VI. Conclusions

We proposed an online mechanism for edge SLA provisioning with QoS guarantee, which achieves competitive social welfare while ensuring truthfulness of application owners' bids, individual rationality, budget balance, and computational efficiency. The mechanism does not assume application owners' knowledge regarding the edge network, and allows the edge provider to fully decide the provisioning of computing and



(c) Running time vs. # APs (d) Running time vs. # data sources

Fig. 4: Scalability factors of the EA algorithm.

network resources. It achieves the best known competitive ratio for both online and offline truthful edge resource provisioning mechanism design. Simulations showed that our mechanism achieved superior performance over delay-agnostic and heuristic solutions in practical settings.

## References

[1] "Pokémon GO." [Online]. Available: https://pokemongolive.com/en/

[2] "Workrooms | VR for Business Meetings - Oculus." [Online]. Available: https://www.oculus.com/workrooms/

[3] J. Korluk, "Edge computing and digital marketing: creating personalized experiences," 2019. [Online]. Available: https://aimarketingspot.com/edge-computing-digital-marketing-creating-personalized-experiences/

[4] R. Yu, G. Xue, Y. Wan, J. Tang, D. Yang, and Y. Ji, "Robust Resource Provisioning in Time-Varying Edge Networks," in *ACM Mobihoc*, 2020.

[5] R. Yu, G. Xue, and X. Zhang, "Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective," in *IEEE INFOCOM*, 2018, pp. 1–9.

[6] Amazon, "Amazon EC2 Spot Instances." [Online]. Available: https://aws.amazon.com/ec2/spot

[7] Q. Wang, K. Ren, and X. Meng, "When Cloud Meets eBay: Towards Effective Pricing for Cloud Computing," in *IEEE INFOCOM*, 2012, pp. 936–944.

[8] H. Roh, C. Jung, W. Lee, and D.-Z. Du, "Resource Pricing Game in Geo-Distributed Clouds," in *IEEE INFOCOM*, 2013, pp. 1519–1527.

[9] W. Shi, C. Wu, and Z. Li, "An Online Auction Mechanism for Dynamic Virtual Cluster Provisioning in Geo-Distributed Clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 677–688, mar 2017.

[10] Z. Huang, S. M. Weinberg, L. Zheng, C. Joe-Wong, and M. Chiang, "Discovering Valuations and Enforcing Truthfulness in a Deadline-Aware Scheduler," in *IEEE INFOCOM*, 2017, pp. 1–9.

[11] L. Zhang, Z. Li, and C. Wu, "Dynamic Resource Provisioning in Cloud Computing: A Randomized Auction Approach," in *IEEE INFOCOM*, 2014, pp. 433–441.

[12] W. Shi, L. Zhang, C. Wu, Z. Li, and F. C. Lau, "An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing," *ACM SIGMETRICS PER*, vol. 42, no. 1, pp. 71–83, jun 2014.

[13] M. Siew, K. Guo, D. Cai, L. Li, and T. Q. S. Quek, "Let's Share VMs: Optimal Placement and Pricing across Base Stations in MEC Systems," in *IEEE INFOCOM*, 2021, pp. 1–9.

[14] Y.-H. Hung, C.-Y. Wang, and R.-H. Hwang, "Optimizing Social Welfare of Live Video Streaming Services in Mobile Edge Computing," *IEEE Trans. Mob. Comput.*, vol. 19, no. 4, pp. 922–934, apr 2020.

[15] A. Kiani and N. Ansari, "Toward Hierarchical Mobile Edge Computing: An Auction-Based Profit Maximization Approach," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 2082–2091, 2017.

[16] S. Gao, C. Courcoubetis, and L. Duan, "Distributed Double Auctions for Large-Scale Device-To-Device Resource Trading," in *ACM Mobihoc*, 2020, pp. 281–290.

[17] Y. Abdulsalam and M. S. Hossain, "COVID-19 Networking Demand: An Auction-based Mechanism for Automated Selection of Edge Computing Services," *IEEE Trans. Netw. Sci. Eng.*, vol. 14, no. 8, pp. 1–1, 2020.

[18] D. Zhang, L. Tan, J. Ren, M. K. Awad, S. Zhang, Y. Zhang, and P.-J. Wan, "Near-Optimal and Truthful Online Auction for Computation Offloading in Green Edge-Computing Systems," *IEEE Trans. Mob. Comput.*, vol. 19, no. 4, pp. 880–893, 2020.

[19] X. Wang, J. Ye, and J. C. Lui, "Decentralized Task Offloading in Edge Computing: A Multi-User Multi-Armed Bandit Approach," in *IEEE INFOCOM*, may 2022, pp. 1199–1208.

[20] Y. Li, W. Dai, X. Gan, H. Jin, L. Fu, H. Ma, and X. Wang, "Cooperative Service Placement and Scheduling in Edge Clouds: A Deadline-Driven Approach," *IEEE Transactions on Mobile Computing*, vol. 21, no. 10, pp. 3519–3535, oct 2022.

[21] P. Kayal and J. Liebeherr, "Distributed Service Placement in Fog Computing: An Iterative Combinatorial Auction Approach," in *IEEE ICDCS*, 2019, pp. 2145–2156.

[22] J. He, D. Zhang, Y. Zhou, and Y. Zhang, "A Truthful Online Mechanism for Collaborative Computation Offloading in Mobile Edge Computing," *IEEE Trans. Ind. Informatics*, vol. 16, no. 7, pp. 4832–4841, 2020.

[23] Y. Zhan and J. Zhang, "An Incentive Mechanism Design for Efficient Edge Learning by Deep Reinforcement Learning Approach," in *IEEE INFOCOM*, 2020, pp. 2489–2498.

[24] Z. Wang, L. Gao, and J. Huang, "Socially-Optimal Mechanism Design for Incentivized Online Learning," in *IEEE INFOCOM*. IEEE, may 2022, pp. 1828–1837.

[25] Y. Jiao, P. Wang, D. Niyato, and K. Suankaewmanee, "Auction Mechanisms in Cloud/Fog Computing Resource Allocation for Public Blockchain Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 1975–1989, sep 2019.

[26] M. H. Hajiesmaili, L. Deng, M. Chen, and Z. Li, "Incentivizing Device-to-Device Load Balancing for Cellular Networks: An Online Auction Design," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 2, pp. 265–279, feb 2017.

[27] H. Qiu, K. Zhu, N. C. Luong, C. Yi, D. Niyato, and D. I. Kim, "Applications of auction and mechanism design in edge computing: A survey," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, pp. 1034–1058, 6 2022.

[28] D. Porter, S. Rassenti, A. Roopnarine, and V. Smith, "Combinatorial auction design," *Proceedings of the National Academy of Sciences*, vol. 100, no. 19, pp. 11 153–11 157, sep 2003.

[29] J. O. Ledyard, "Optimal combinatoric auctions with single-minded bidders," in *Proc. ACM EC*. ACM Press, 2007, p. 237.

[30] N. Nisan and A. Ronen, "Algorithmic Mechanism Design," *Games and Economic Behavior*, vol. 35, no. 1-2, pp. 166–196, apr 2001.

[31] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A Survey on the Edge Computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[32] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 656–669, jun 2008.

[33] B. Lucier, R. Paes Leme, and E. Tardos, "On Revenue in the Generalized Second Price Auction," in *WWW*, 2012, pp. 361–370.

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

[35] C. Chekuri and S. Khanna, "On Multidimensional Packing Problems," *SIAM J. Comput.*, vol. 33, no. 4, pp. 837–851, jan 2004.

[36] ——, "A Polynomial Time Approximation Scheme for the Multiple Knapsack Problem," *SIAM J. Comput.*, vol. 35, no. 3, pp. 713–728, jan 2005.

[37] B. Awerbuch, Y. Azar, and S. Plotkin, "Throughput-competitive On-Line Routing," in *IEEE FOCS*, 1993, pp. 32–40.

[38] N. Buchbinder and J. (Seffi) Naor, "The Design of Competitive Online Algorithms via a Primal—Dual Approach," *Found. Trends® Theor. Comput. Sci.*, vol. 3, no. 2–3, pp. 93–263, mar 2009.

[39] P. Briest, P. Krysta, and B. Vöcking, "Approximation Techniques for Utilitarian Mechanism Design," *ACM STOC*, pp. 39–48, 2005.

[40] B. M. Waxman, "Routing of Multipoint Connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, 1988.

[41] "Gurobi Optimizer." [Online]. Available: http://www.gurobi.com/products/gurobi-optimizer