Learning-Aided Computation Offloading for Trusted Collaborative Mobile Edge Computing

Yuqing Li, Xiong Wang, Xiaoying Gan, *Member, IEEE*, Haiming Jin, Luoyi Fu, and Xinbing Wang, *Senior Member, IEEE*

Abstract—Cooperative offloading in mobile edge computing enables resource-constrained edge clouds to help each other with computation-intensive tasks. However, the power of such offloading could not be fully unleashed, unless trust risks in collaboration are properly managed. As tasks are outsourced and processed at the network edge, completion latency usually presents high variability that can harm the offered service levels. By jointly considering these two challenges, we propose OLCD, an Online Learning-aided Cooperative offloaDing mechanism under the scenario where computation offloading is organized based on accumulated social trust. Under co-provisioning of computation, transmission, and trust services, trust propagation is performed along the multi-hop offloading path such that tasks are allowed to be fulfilled by powerful edge clouds. We harness Lyapunov optimization to exploit the spatial-temporal optimality of long-term system cost minimization problem. By gap-preserving transformation, we decouple the series of bidirectional offloading problems so that it suffices to solve a separate decision problem for each edge cloud. The optimal offloading control can not materialize without complete latency knowledge. To adapt to latency variability, we resort to the delayed online learning technique to facilitate completion latency prediction under long-duration processing, which is fed as input to queued-based offloading control policy. Such predictive control is specially designed to minimize the loss due to prediction errors over time. We theoretically prove that OLCD guarantees close-to-optimal system performance even with inaccurate prediction, but its robustness is achieved at the expense of decreased stability. Trace-driven simulations demonstrate the efficiency of OLCD as well as its superiorities over prior related work.

Index Terms—Mobile edge computing, multi-hop cooperative offloading, trust propagation, completion latency variability

1 Introduction

Recent years have witnessed the explosive growth of data generated at the network edge, mainly driven by the proliferation of Internet of Things [1]. The failure to guarantee low latency and location-awareness undercuts the ability of traditional cloud computing solutions [2]. Mobile edge computing (MEC) is emerging as a new compelling computing paradigm by pushing cloud computing capabilities closer to end users, underpinning a variety of computation-intensive yet latency-sensitive applications, such as face recognition, natural language processing and interactive gaming [3], [4], [5]. As a result, it is estimated that over 90 percent of the data will be stored or processed at the network edge [6]. However, the limited resources available to individual edge clouds (e.g., small cell base stations or WiFi access points) remain to be the biggest obstacle [7], [8]. Although there exist a few works for offloading computation tasks exceeding edge clouds' capacity to the remote cloud [9], [10], relying on a single edge cloud significantly limits MEC performance.

By exploiting cooperations among edge clouds, MEC enables resource-constrained edge clouds to help each other with computation-intensive tasks, catering to user heterogeneous demands [11], [12]. Cooperative offloading services often assisted by virtualization technologies realize more flexible workload and resource sharing within specific geographic regions. While having received significant attention recently, existing solutions mostly make a simplifying assumption that tasks can be offloaded only once rather than being offloaded further [8], [13]. In contrast, we focus on the multi-hop case that allows tasks to be offloaded multiple hops away. Compared to 1-hop case with limited number of physical neighbors, multi-hop offloading would be more promising in exploiting collaborative computing capabilities, since it can offload tasks to more powerful edge clouds that lie beyond 1-hop neighbors [14], [15].

Despite the clear advantages, trust risks in MEC collaboration create tremendous difficulties for fully reaping benefits of multi-hop offloading. Acting as both resource requester and provider, edge clouds are often owned and deployed by self-interested individuals (e.g., home/enterprise owners) concerned about limited computing resources. In practice, edge clouds expect to offload tasks to powerful neighbors, but as providers, they may refuse to support offloading services or deliberately throttle resources only to provide low-quality services, if possible. Such uncooperative strategic behaviors

• *Y. Li, X. Wang, H. Jin, L. Fu, and X. Wang are with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {liyuqing, wangxiongsjtu, jinhaiming, yiluofu, xwang8}@sjtu.edu.cn.*
• *X. Gan is with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China, and also with the National Mobile Communications Research Laboratory, Southeast University, Nanjing 211189, China. E-mail: ganxiaoying@sjtu.edu.cn.*

Manuscript received 28 Dec. 2018; revised 26 June 2019; accepted 3 Aug. 2019. Date of publication 14 Aug. 2019; date of current version 3 Nov. 2020. (Corresponding author: Xinbing Wang.)
Digital Object Identifier no. 10.1109/TMC.2019.2934103

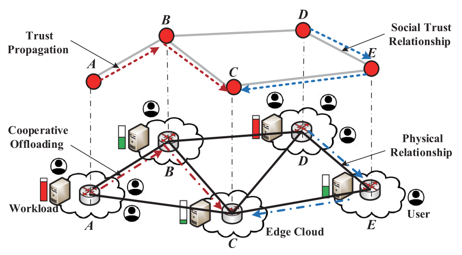


Fig. 1. An illustration of the trusted cooperative offloading for MEC.

in the long run, may provoke trust crisis of confidence in neighbors' cooperation, thereby suppressing efficiency of offloading services [16], [17]. To the best of our knowledge, [18] and [19] are the only results that explicitly build social trust relationships to identify trustworthy edge clouds. However, they typically consider either static trust-based interactions, or cooperative offloading confined within one hop. In this work, we will explore the benefits of *multi-hop cooperative offloading* that relies on *accumulated social trust relationships*.

We particularly identify two major challenges in designing an efficient trusted collaborative MEC paradigm, both of which are addressed in this paper.

The first major challenge comes from *variability of completion latency*, which can harm the offered service levels. As the basis of offloading decision making, completion latency has received increasing attention in performance evaluation of MEC services. An extensive body of research on latency minimization has emerged [8], [13], [14], [15], most of which implicitly assumes that completion latency can be determined in advance. Such practice leads to tractable analysis, but fails to capture latency variability, which is commonplace in practical distributed computing systems [20]. As tasks are outsourced and processed at the network edge, completion latency usually presents high variability mainly due to different workload levels or resource contention in virtualized environments [21]. This introduces a heavy burden on performing efficient cooperative offloading control.

The second challenge is on *co-provisioning of computation, transmission and social trust services*, which distinguishes the trusted cooperative offloading of interest from prior offloading solutions that manage only the former two services [13], [14], [15]. By exploiting cooperations among edge clouds, tasks are allowed to be offloaded multiple hops away to reduce computation latency but at the expense of increased trust risks and transmission latency. One critical issue of trusted cooperative offloading is how to strike a good balance among computing power, radio access and social trust resources, so as to optimize the quality of services. This brings new modeling requirements for incorporating interplay and interdependency among the management of these three resources.

By jointly considering the above challenges, we develop an online learning-aided cooperative offloading mechanism called OLCD. A salient contribution of our approach is that, multi-hop offloading is organized based on accumulated social trust, which, combined with predictive computing capabilities, provides users with high quality of services. We harness Lyapunov optimization to exploit spatial-temporal optimality of long-term cost minimization problem. OLCD is devoted to an efficient online offloading control policy by addressing the assignment problem for each edge cloud: how to assign available computing capabilities to tasks for the minimum system cost in terms of latency and trust risks? However, such offloading control can not materialize without complete latency knowledge. To adapt to latency variability, OLCD resorts to the delayed online learning technique to predict task completion latency, which is used for the basis of offloading decision making. By co-provisioning of computation, transmission and trust services, the optimal offloading decisions obtained via a reduced minimum cost maximum flow problem, are expected to achieve latency versus trust risk tradeoff. To encourage high-quality service offerings, accumulated trust update is performed by aggregating locally-generated results of completed tasks to yield global trust values for cooperative edge clouds.

Our main contributions are highlighted as follows.

- We propose OLCD, an online learning-aided cooperative offloading mechanism, where trust propagation is performed along multi-hop offloading path to explore collaborative computing capabilities. To the best of our knowledge, we're the first to explore the benefits of *multi-hop cooperative offloading in MEC* by taking into account both *completion latency variability and trust risks in collaboration*.
- Specifically, a long-term cost minimization problem with co-provisioning of computation, transmission and trust services is formulated. By gap-preserving transformation, we decouple the series of bidirectional offloading problems so that it suffices to solve a separate decision problem for each edge cloud. With the aim of minimizing the loss due to prediction errors over time, OLCD adapts the delayed online learning technique to facilitate completion latency prediction, which is fed as input to Lyapunov-based cooperative offloading control. An expected regret bound is proven as compared to the best static predictor. We theoretically prove that OLCD guarantees close-to-optimal system performance and robustness to prediction errors.
- We evaluate the performance of OLCD with real-world traces from Google cluster. Our results show that OLCD outperforms prior related work and approaches near-optimal performance, even under workload prediction errors. Moreover, it's suggested that high-intensity trust propagation can yield a performance boost in cost reduction.

In what follows, we describe the system model and formulate the trusted cooperative offloading problem in Sections 2 and 3. To proceed, we propose an online learning-aided multi-hop offloading mechanism in Section 4. Trace-driven simulations are performed in Section 5. In Section 6, we briefly review related work. Finally, we conclude the paper and future work in Section 7.

2 System Model

We describe the system model for each of components in the trusted cooperative offloading, as shown in Fig. 1. Table 1 lists the major notations and descriptions used in this paper.

2.1 Collaborative MEC System

Consider the trusted collaborative MEC system consisting of a set $N = \{1, \ldots, N\}$ of densely deployed edge clouds.

TABLE 1
Major Notations

Notation	Description
$\boldsymbol{I}, i, \boldsymbol{N}, n$	set and index of users and edge clouds
λ^i, γ^i	task size, computation intensity of task i
t_0^i, b^i, σ^i	arrival time, deadline and trust demand of task i
$\boldsymbol{N}^i, \boldsymbol{N}_n$	i's contributing edge cloud set, n's trusted neighbor set
d_n^i, φ	computation latency, backhaul transmission latency
E_n, B_{nm}	local computing capacity, offloading capacity
$\omega_{mn}(t), W_n^i$	trust value, cumulative trust value at n
$a_n^i(t), r_n^i(t)$	admission decision variable, processing rate
$\alpha_n^i(t), \beta_{nm}^i(t)$	local computing/offloading decision variables
$\alpha_n(t), \beta_n(t)$	workloads dispatched to the Edge-B and offloaded away
$Q_n^F(t), Q_n^B(t)$	the Edge-F and Edge-B task queues of n
ρ_d, ρ_s	weighting cost parameters of latency and social trust
$D_n^C(t), D_n^T(t)$	computation/transmission latency cost
$C(t), S_n(t)$	system cost, trust risk cost

Endowed with cloud-like computing capacity, each edge cloud is associated with an access point (e.g., small cell base station, WiFi access point) covering a dedicated local area, and serves a set $\boldsymbol{I} = \{1, 2, \ldots\}$ of user-generated computation-intensive tasks[1] in the area. Suppose edge clouds in the neighborhood (e.g., A, B and C in Fig. 1) are connected by backhaul links, which can be used to send task requests or responses between edge clouds [7]. By exploiting cooperations among edge clouds, tasks that arrive at one edge cloud can be either processed locally, or offloaded to non-local edge clouds via backhaul links for high-quality services.[2] Different from most existing one-hop offloading works, a task can be offloaded multiple hops away and finally fulfilled by powerful edge clouds. Notice that our mechanism is also compatible with edge cloud-to-cloud offloading strategy [9] (i.e., offloading edge clouds' unsatisfied computation tasks to the remote cloud) but along with high transmission latency and huge bandwidth costs. The system runs in a time-slotted fashion for making decisions, i.e., $t \in \boldsymbol{T} = \{0, \ldots, T-1\}$, where the slot (e.g., 1-5 minutes) is a much slower time scale than that of task arrival and offloading.

2.2 Social Trust Model

Of particular importance is that multi-hop cooperative offloading relies on accumulated social trust relationships to identify trustworthy edge clouds. We introduce social trust model captured by directed graph $\mathcal{G} = (\boldsymbol{N}, \boldsymbol{E})$, where $\boldsymbol{E} = \{(n, m) : e_{nm} = 1, \forall n, m \in \boldsymbol{N}\}$. $e_{nm} = 1$ if and only if edge clouds n and m have positive trust value with each other, where the trust value perceived by n is denoted by $\omega_{nm}(t) \in [0, 1]$, characterizing the confidence that n has in m's direct cooperation at slot t based on historical interactions. Here "direct" suggests that social trust relationships are built upon physical relationships,[3] i.e., two edge clouds that are physically unreachable have no trust relationships and that have positive mutual trust value are bound to be physical neighbors. Due to heterogeneity in edge clouds' computing capability, $\omega_{nm}(t)$ may not equal $\omega_{mn}(t)$. Let $\boldsymbol{N}_n = \{m | e_{nm} = 1, \forall m \in \boldsymbol{N} \setminus \{n\}\} \subseteq \boldsymbol{N}$ denote n's trusted neighbor set, capturing the set of edge clouds that it can directly interact with (i.e., physical neighbors with positive trust value). Consider the trusted MEC service provider who is responsible for dynamic trust management including trust value update and trusted neighbor set update. Specifically, add a new edge cloud to $\boldsymbol{N}_n$ when it builds trust relationship with n, and delete edge clouds from $\boldsymbol{N}_n$ if they miss positive social trust.

As shown in Fig. 1, social trust relationships among edge clouds can be leveraged to facilitate cooperative offloading. For example, since $e_{AB} > 0$ and $e_{BC} > 0$, computation workloads in edge cloud A can be offloaded to edge cloud B, and finally processed in lightly-loaded edge cloud C. Here physical neighbors A and C miss positive mutual trust (i.e., $e_{AC} = 0$) due to lack of good recent interactions, and thus the workload cannot be directly offloaded from A to C.

2.3 Multi-Hop Task Offloading via Trust Propagation

Mobile users randomly arrive at the system by submitting task requests with *diverse deadline and trust demands*.[4] Each of them connects to the edge cloud that covers its vicinity.[5] Formally, the task corresponding to user $i \in \boldsymbol{I}$ can be specified by a tuple $\left(\lambda^i, \gamma^i, t_0^i, b^i, \sigma^i\right)$, where $\lambda^i \in [\lambda^{\min}, \lambda^{\max}]$ is task size (in bits) that needs to be offloaded, and computation intensity $\gamma^i \in [\gamma^{\min}, \gamma^{\max}]$ is number of CPU cycles for processing one-bit task. Upon arriving at slot $t_0^i \in \boldsymbol{T}$, user i reports the desired deadline b^i and social trust σ^i, capturing the maximum completion latency and trust risks in collaboration that i can tolerate.

For every task, each associated edge cloud can either directly finish it (i.e., local computing), or forward it to another trusted edge cloud in $\boldsymbol{N}_n$ (i.e., cooperative offloading). Repeat this process and the multi-hop offloading path is formed finally. For any task $i \in \boldsymbol{I}$, we use variable $K \in \{0, 1, \ldots\}$ to capture how many hops i is offloaded in total. As shown in Fig. 2, user i first connects to local edge

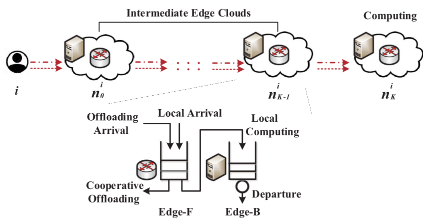


Fig. 2. Top: trust propagation along multi-hop offloading path. Bottom: online offloading control for any intermediate edge cloud (Edge-F: Front-end server, Edge-B: Back-end cluster).

1. In the rest of this paper, we will use "user" and "task" interchangeably.
2. We focus on cooperative offloading among edge clouds, where user requests are entirely offloaded from end devices to requesting edge clouds.
3. We consider the case with fixed physical connections among edge clouds, i.e., physical relationships remain the same. While taking into account trust risks in collaboration, both trust value and trusted neighbor set will be updated based on interaction results, i.e., social trust relationships are always changing.

4. Users usually have heterogeneous service requests regardless of whatever applications. Thus, we won't make any distinction among MEC applications. In particular, to satisfy user diverse deadline and trust demands has promoted the development of deadline-aware [22] and trust-aware [23] scheduling solutions.
5. We focus on the cooperative offloading case without user mobility. The effect of user mobility on offloading control will be discussed in Section 4.6.

cloud $n_0^i \in \boldsymbol{N}$, and then its task request is offloaded K hops to edge cloud n_K^i for processing. Let $\boldsymbol{N}^i$ denote the set of edge clouds contributing to the offloading service for user i, and $\pi^i = \{n_0^i, \ldots, n_K^i\}$ denote the permutation of all $K+1$ edge clouds along the offloading path. Under social trust relationships, *trust propagation* is performed along multi-hop offloading path. For simplicity, we denote the cumulative trust value of k-hop edge cloud n_k^i as $W_k^i = W_{n_k^i}^i, \forall k \in \{0, \ldots, K\}$. That is,

$$W_k^i = \begin{cases} [1-\partial]\omega_{n_{k-1}^i n_k^i}(t_k^i) \cdot W_{k-1}^i, & \text{if} \quad k > 0, \\ \omega_0^i(t_0^i), & \text{if} \quad k = 0, \end{cases} \tag{1}$$

where $\partial \in [0,1)$ is the decaying factor that decreases the trust value when the number of offloading hops increases,[6] t_k^i is the time that task i is assigned to edge cloud n_k^i, and $\omega_0^i(t_0^i)$ is the trust value between i and requesting edge cloud n_0^i at slot t_0^i.

Taking trusted cooperative offloading illustrated in Fig. 1 for example, user i submits task request to edge cloud A at slot t_0^i with trust value $\omega_A^i(t_0^i)$, and then the task is offloaded to edge cloud B at slot t_1^i and finally processed in edge cloud C at slot t_2^i. Hence, for this offloading service, offloading hop number $K=2$ and the total cumulative trust value $W_C^i = \omega_A^i(t_0^i)\omega_{AB}(t_1^i)\omega_{BC}(t_2^i)$.

2.4 Online Control for Edge Clouds

Given the multi-hop offloading path shown in Fig. 2, let's take a closer look at any intermediate edge cloud, and learn how cooperative offloading functions. Similar to cloud datacenters [24], each edge cloud consists of two parts, front-end server (Edge-F) and back-end cluster (Edge-B), where Edge-F is responsible for task admission and scheduling, and Edge-B utilizes provisioned computing resources to process tasks dispatched from Edge-F. These stochastic control processes, together with time-varying task arrivals, may bring about dynamics of workloads in Edge-F and Edge-B. We apply queueing theory to handle such dynamics.

We consider the distributed scenario where edge clouds coordinate their online control strategies in an autonomous way. Each edge cloud $n \in \boldsymbol{N}$ maintains two task queues, whose backlogs[7] $Q_n^F(t)$ and $Q_n^B(t)$ capture the amount of workloads queued in Edge-F and Edge-B at the beginning of slot t. The optimal cooperative offloading control cannot materialize without complete backlog information among edge clouds. Due to trust risks in collaboration, edge clouds would prefer to interact with their trusted neighbors. That is, edge clouds are expected to share backlog information with peers in trusted neighbor set. At each slot, edge clouds serve both interaction results and communication traffic data. Since the size of the former is usually small and constant, it is acceptable to assume that interaction results can be transmitted to edge clouds immediately and the associated overhead/delay can be ignored without affecting online control. Thus, we only consider communication traffic data later.

2.4.1 Online Control for Edge-F

1) *Task Admission:* At each slot, heterogenous task requests arrive at the system. Denote $\boldsymbol{I}(t) \subseteq \boldsymbol{I}$ as the set of tasks newly arrived at slot t. The instantaneous demand of user $i \in \boldsymbol{I}(t)$ for edge cloud n can be described as $A_n^i(t) = \lambda^i \mathbb{1}_{\{n_0^i = n\}}$. The amount of workloads admitted into the Edge-F of n is $a_n(t) = \sum_{i \in \boldsymbol{I}(t)} a_n^i(t)$, where $0 \leq a_n^i(t) \leq A_n^i(t)$ suggests that not all tasks are allowed in the system so as to prevent system overload. The task admission decisions for the whole system can thus be given by vector $\boldsymbol{a}(t) = \{a_n^i(t), \forall i \in \boldsymbol{I}(t), n \in \boldsymbol{N}\}$.

2) *Task Scheduling:* In addition to admitting locally arrived tasks, the Edge-F, under multi-hop offloading, also receives tasks offloaded from others. The next control is to determine which tasks are processed locally or offloaded to trusted neighbors, corresponding to *local computing* and *cooperative offloading*.

For each task $i \in \boldsymbol{Q}_n^F(t)$, we use $\alpha_n^i(t) = \{0,1\}$ to denote whether i is dispatched to edge cloud n's Edge-B. Obviously, $\alpha_n^i(t) = 1$ represents that i is finally offloaded to n and its multi-hop offloading path ends, and $\alpha_n^i(t) = 0$ otherwise. The amount of workloads dispatched from n's Edge-F to Edge-B is thus

$$\alpha_n(t) = \sum_{i \in \boldsymbol{Q}_n^F(t)} \lambda^i \alpha_n^i(t). \tag{2}$$

We introduce the binary variable $\beta_{nm}^i(t)$ to capture cooperative offloading decisions, where $\beta_{nm}^i(t) = 1$ represents that edge cloud n offloads task i to trusted neighbor $m \in \boldsymbol{N}_n$ at slot t, i.e., m will be added to i's offloading path, and $\beta_{nm}^i(t) = 0$ otherwise. The links between edge clouds are assumed to be error-free, since the wired backhaul links are typically reliable and only require simple channel coding with substantially lower complexity than computation-intensive tasks [15]. Thus, the backhaul transmission latency does not involve complicated encoding and decoding. According to [25], the transmission latency of the backhaul is proportional to the size of traffic data with scaling factor φ. Then we obtain the unprocessed workloads offloaded away from n, i.e.,

$$\beta_n(t) = \sum_{i \in \boldsymbol{Q}_n^F(t)} \sum_{m \in \boldsymbol{N}_n} \lambda^i \beta_{nm}^i(t). \tag{3}$$

The task scheduling decisions in the whole system can be given by $\boldsymbol{s}(t) = \{\alpha_n^i(t), \beta_{nm}^i(t), \forall i \in \boldsymbol{Q}_n^F(t), m \in \boldsymbol{N}_n, n \in \boldsymbol{N}\}$.

2.4.2 Online Control for Edge-B

The Edge-B processes tasks dispatched from Edge-F. The computing capability captured by processing rate largely depends on two aspects: workloads and computing resources. Let $\varsigma_n(t)$ denote edge cloud n's service limitation which is determined by available resources at slot t. We introduce net present value function [4] to expound the relationship between processing rate and workload/resource levels, which is proven to fit measurement results [26]. The processing rate (in CPU cycles per second) can be computed by

6. The decaying factor is introduced to control weights of trust value by the "distance" (i.e., offloading hop number) between edge clouds, making the weights assigned to short-distance neighbors larger than long-distance ones. Accordingly, cumulative trust value is monotonically decreasing with the distance, suggesting edge clouds would prefer short-distance neighbors to help with task processing, especially under trust risks. The effect of decaying factor on trusted cooperative offloading performance is further studied in Section 5.

7. We use $\boldsymbol{Q}_n^F(t)$ to denote the set of tasks associated with $Q_n^F(t)$. The same is with $\boldsymbol{Q}_n^B(t)$ and $Q_n^B(t)$.

$r_n(t) = \frac{1}{y} x^{\varsigma_n(t) - Q_n^B(t)}$, where $x \in (1, \infty)$ controls the skewness of the relationship between processing rate and workload/resource levels, and y captures the speed when Edge-B is fully loaded. Under the same MEC system structure, parameters x and y are identical to all edge clouds. The heterogeneity in computing capability primarily comes from differences in workload/resource levels. Intuitively, the less workloads and the more available resources, the larger processing rate will be.

Our study highlights the intriguing role of fair-share scheduling, which has gained growing attention recently. In particular, Rate Control Protocol (RCP) scheduling has been developed as an adaptive fair-share solution, where every router assigns the same rate to all requests and updates the rate approximately once per slot [27], [28]. Compared to priority-based scheduling, RCP guarantees that all task requests buffered in the same Edge-B share the same resources without preemption, thus making requests finish as quickly as possible while staying stable and fair among requests. For any task $i \in \boldsymbol{Q}_n^B(t)$, the amount of its workloads processed at slot t under RCP scheduling can be described as

$$r_n^i(t) = \min\left\{ \frac{r_n(t)}{\gamma^i |\boldsymbol{Q}_n^B(t)|}, \lambda^i(:,t) \right\}, \tag{4}$$

where $\lambda^i(:,t) = \lambda^i - \sum_{n\in\boldsymbol{N}} \sum_{\tau=0}^{t-1} r_n^i(\tau)$ denotes the residual workloads at the beginning of slot t. Users will leave the system as soon as their computation tasks are completely served, i.e., $\lambda^i(:,t) = 0$. Let $d_n^i \in (0, d^{C,\max}]$ denote the latency when task i is served in edge cloud n. Due to different workload levels or resource contention [21], the computation latency usually presents high variability, making it unknown until the task is finished. We thus incorporate online learning into multi-hop offloading mechanism in Section 4, to predict completion latency knowledge.

2.4.3 *Task Queues*

We adopt the convention that task scheduling and processing at slot t happen at the beginning of the slot, while task acceptance (i.e., admitting tasks arrived locally or offloaded from neighbors/Edge-F) happens at the end [29]. Accordingly, the queueing dynamics of task queues, Q_n^F and Q_n^B, associated with any edge cloud $n \in \boldsymbol{N}$ can be described as

$$Q_n^F(t+1) = \max\{Q_n^F(t) - \alpha_n(t) - \beta_n(t), 0\} + a_n(t) + \sum_{m\in\boldsymbol{N}_n} \beta_{mn}(t), \tag{5}$$

$$Q_n^B(t+1) = \max\left\{ Q_n^B(t) - \sum_{i\in\boldsymbol{Q}_n^B(t)} r_n^i(t),\ 0 \right\} + \alpha_n(t), \tag{6}$$

where $\beta_{mn}(t) = \sum_{i\in\boldsymbol{Q}_m^F(t)} \lambda^i \beta_{mn}^i(t)$ is the amount of workloads offloaded to n from trusted neighbor $m \in \boldsymbol{N}_n$. The first term on the right-hand-side (RHS) of (5) captures the unprocessed workloads in Edge-F at slot t after part of workloads are offloaded away or dispatched to Edge-B, and the last two terms describe the workloads arrived locally and offloaded from trusted neighbors. The first term on the RHS of (6) denotes the unprocessed workloads in Edge-B after part of workloads are processed.

3 Problem Formulation

By exploring collaborative computing potentials, the desired trusted cooperative offloading mechanism does the best effort to serve tasks while providing users with quality of service (QoS) guarantee. We focus on the QoS performance in terms of trust risks and completion latency (including transmission and computation latency). In particular, beneficial offloading, user trust demand and edge clouds' scheduling capacity constraints are respected.

3.1 Constraints of The Problem

1) *Scheduling Exclusion Constraint:* For task i queued in Edge-F, edge could n can either process it or offload it to one of trusted neighbors unless keeping it waiting in queue Q_n^F, i.e.,

$$\alpha_n^i(t) + \sum_{m\in\boldsymbol{N}_n} \beta_{nm}^i(t) \le 1, \forall i \in \boldsymbol{Q}_n^F(t), n \in \boldsymbol{N}. \tag{7}$$

2) *Beneficial Offloading Constraint:* The decision of edge cloud n that offloads task $i \in \boldsymbol{Q}_n^F(t)$ to trusted neighbor m (i.e., $\beta_{nm}^i(t) = 1$) is beneficial if offloading to m does not incur higher completion latency than local computing, i.e.,

$$d_m^i + \varphi\lambda^i \le d_n^i, \forall i \in \boldsymbol{Q}_n^F(t), m \in \boldsymbol{N}_n, n \in \boldsymbol{N}, \tag{8}$$

where $\varphi > 0$ is a coefficient representing the backhaul transmission latency for one-bit task, and d_n^i is the computation latency for task i when processed in n.

3) *Trust Demand Constraint:* Multi-hop offloading relies on social trust relationships. This constraint enforces that task i can be offloaded from edge cloud n to its neighbor m at slot t (i.e., $\beta_{nm}^i(t) = 1$), if the resulting cumulative trust value W_m^i for offloading to m satisfies user trust demand σ^i, i.e.,

$$W_m^i = [1 - \partial]\omega_{nm}(t) \cdot W_n^i \ge \sigma^i, \forall i \in \boldsymbol{Q}_n^F(t), m \in N_n, n \in \boldsymbol{N}. \tag{9}$$

Remark. Constraints (8) and (9) suggest that cooperative offloading control is performed under the concept of beneficial offloading and trusted offloading, which is vital to providing high QoS. On this basis, a tradeoff between completion latency and trust risks is established accordingly. It's true that 1-hop offloading is superior for low trust risk and low transmission latency, but that is not the whole story. Recall that MEC paradigm is proposed for computation-intensive tasks, i.e., computation latency is usually large, especially for resource-constrained edge clouds. In multi-hop case, however, tasks are more likely to be offloaded to powerful edge clouds lying beyond physical neighbors, thus realizing a huge decrease in computation latency only at the expense of slightly increased trust risks and transmission latency.

4) *Scheduling Capacity Constraint:* We highlight the limited scheduling capacity of edge clouds as follows:

$$\sum_{i\in\boldsymbol{Q}_n^F(t)} \alpha_n^i(t) \le E_n, \forall n \in \boldsymbol{N}, \tag{10}$$

$$\sum_{i\in\boldsymbol{Q}_n^F(t)} \beta_{nm}^i(t) \le B_{nm}, \forall m \in \boldsymbol{N}_n, n \in \boldsymbol{N}, \tag{11}$$

where (10) indicates at most E_n tasks are dispatched to Edge-B at one slot for guaranteed processing rate, and (11) specifies the limited offloading capacity of edge cloud m for n by placing an upper bound $B_{nm} \in [0, B]$ for the number of offloaded tasks.

5) *Stability Constraint:* An edge cloud n is stable only if it has a bounded time-averaged backlog [29], i.e.,

$$\bar{Q}_n = \limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\{Q_n^F(t) + Q_n^B(t)\} < \infty, \forall n \in \boldsymbol{N}. \quad (12)$$

3.2 Trusted Cooperative Offloading Problem

Taking trust relationships into account, we leverage cooperations among edge clouds to provide users with QoS guarantee using the offloading mechanism defined as Definition 1.

Definition 1 (Trusted Cooperative Offloading Mechanism). *A trusted cooperative offloading mechanism determines admission decisions* $\boldsymbol{a}(t) = \{a_n^i(t), \forall i \in \boldsymbol{I}(t), n \in \boldsymbol{N}\}$ *and scheduling decisions* $\boldsymbol{s}(t) = \{\alpha_n^i(t), \beta_{nm}^i(t),\ \forall i \in \boldsymbol{Q}_n^F(t), m \in \boldsymbol{N}_n, n \in \boldsymbol{N}\}$ *to provide high QoS performance in a best-effort manner. In particular,* $m \in \boldsymbol{N}_n$ *suggests that the cooperative offloading decisions* $\beta_{nm}^i(t)$ *are made based on social trust relationships.*

Under such desired mechanism, the instantaneous system cost with the co-provisioning of computation, transmission and social trust services, can be defined as

$$C(t) = \sum_{n\in\boldsymbol{N}} \big(\rho_d[D_n^C(t) + D_n^T(t)] + \rho_s S_n(t)\big), \quad (13)$$

where $\rho_d, \rho_s > 0$ denote the weighting cost parameters of latency and trust risk for online control, respectively. $D_n^C(t) = \sum_{i\in\boldsymbol{Q}_n^F(t)} d_n^i \alpha_n^i(t)$ describes the computation latency cost at slot t, establishing connections between local computing decisions $\alpha_n^i(t)$ for Edge-F and processing results d_n^i for Edge-B. $D_n^T(t) = \sum_{i\in\boldsymbol{Q}_n^F(t)} \sum_{m\in\boldsymbol{N}_n} \varphi\lambda^i\beta_{nm}^i(t)$ and $S_n(t) = \sum_{i\in\boldsymbol{Q}_n^F(t)} \sum_{m\in\boldsymbol{N}_n} \lambda^i\beta_{nm}^i(t)[1-\omega_{nm}(t) + \partial\omega_{nm}(t)]$ capture the transmission latency cost (for transmission among associated edge clouds) and trust risk cost (for managing security risks) when offloading tasks away. Notice that the transmission latency between users and requesting edge clouds is irrelevant to online offloading control later. For simplicity, we will omit it and use this form for transmission latency in the rest of paper.

With the goal of minimizing long-term system cost, the formal mathematical formulation of the Trusted Cooperative Offloading mechanism design problem (TCO) is given by

$$\text{TCO:} \quad \min_{\boldsymbol{a}(t),\boldsymbol{s}(t)} \quad \frac{1}{T}\sum_{t\in\boldsymbol{T}} C(t) \quad (14)$$

$$\text{s.t.} \quad \text{Constraints (7)-(12).} \quad (14a)$$

Challenges. There are two challenges that impede the derivation of optimal online control: (1) spatial-temporal coupled decisions: tasks can be offloaded across edge clouds and the processing duration is affected by scheduling decisions; (2) no complete offline information on completion latency: as the basis for offloading decision making, task completion latency usually presents high variability, especially under stochastic task arrivals. These challenges call for an online optimization approach that can efficiently perform multi-hop offloading with latency prediction knowledge.

4 Online Learning-Aided Cooperative Offloading Mechanism

To circumvent the above challenge from stochastic task arrivals, we develop a novel mechanism called OLCD for online offloading control by harnessing Lyapunov optimization, which relies only on current system information and past offloading control history, while not on any future information. Meanwhile, it asymptotically converges to the optimal solution with complete future information (i.e., task arrivals across all slots).

4.1 Lyapunov Optimization

Let $\boldsymbol{\Theta}(t) = [\boldsymbol{Q}^F(t), \boldsymbol{Q}^B(t)]$ be the aggregate queue vector. To start, we define the perturbed Lyapunov function [29], [30] as

$$L(\boldsymbol{\Theta}(t)) = \frac{1}{2}\|\boldsymbol{Q}^F(t) - \boldsymbol{\theta}\| + \frac{1}{2}\|\boldsymbol{Q}^B(t)\|, \quad (15)$$

where $\boldsymbol{\theta} = \theta_n \cdot \boldsymbol{1}^N$ with θ_n being perturbation parameters (to be specified later). We define the Lyapunov drift as $\Delta(\boldsymbol{\Theta}(t)) = \mathbb{E}\{L(\boldsymbol{\Theta}(t+1)) - L(\boldsymbol{\Theta}(t))|\boldsymbol{\Theta}(t)\}$ to capture expected changes in the quadratic function of backlogs over each slot. We incorporate system cost into Lyapunov drift, providing network stability and cost minimization jointly. At each slot, we try to minimize the drift-plus-penalty function greedily, i.e.,

$$\min \quad \Delta(\boldsymbol{\Theta}(t)) + V\mathbb{E}\{C(t)|\boldsymbol{\Theta}(t)\}, \quad (16)$$

where V is a tunable parameter weighting how much importance we stress on minimizing system cost. Intuitively, this objective function is rarely converged at the equilibrium state if $L(\boldsymbol{\Theta}(t))$ is too large. To keep $L(\boldsymbol{\Theta}(t))$ small, we "push" $Q_n^F(t)$ towards θ_n, which ensures that each Edge-F has certain amount of workloads to schedule, avoiding a waste of computing resources and social trust relationships. By carefully choosing the value of θ_n, it can be guaranteed that the Edge-F task queue always has enough requests whenever edge cloud n decides to schedule. The TCO problem with spatial-temporal coupled decisions is thus converted to per-slot optimization problem (16) subject to constraints (7), (8), (9), (10), and (11), which is solvable with only current information.

Lemma 1. *Denote* $\tilde{Q}_n^F(t) = Q_n^F(t) - \theta_n$ *and* $\tilde{\omega}_{nm}(t) = 1 - \partial\omega_{nm}(t)$. *Under any feasible control policy, we have*

$$\begin{aligned}
&\Delta(\Theta(t)) + V\mathbb{E}\{C(t)|\Theta(t)\} \\
&\le \sum_{n\in\boldsymbol{N}} \mathbb{E}\Bigg\{ \sum_{i\in\boldsymbol{I}(t)} a_n^i(t)\tilde{Q}_n^F(t)|\Theta(t) \Bigg\} + B_1 + B_2(t) \\
&+ \sum_{n\in\boldsymbol{N}} \mathbb{E}\Bigg\{ \sum_{i\in\boldsymbol{Q}_n^F(t)} \alpha_n^i(t)[\lambda^i Q_n^B(t) - \lambda^i\tilde{Q}_n^F(t) + V\rho_d d_n^i]|\Theta(t) \Bigg\} \\
&+ \sum_{n\in\boldsymbol{N}} \mathbb{E}\Bigg\{ \sum_{m\in\boldsymbol{N}_n} \sum_{i\in\boldsymbol{Q}_m^F(t)} \lambda^i\beta_{mn}^i(t)\tilde{Q}_n^F(t) - \sum_{i\in\boldsymbol{Q}_n^F(t)} \sum_{m\in\boldsymbol{N}_n} \lambda^i \cdot \\
&\qquad \beta_{nm}^i(t)[\tilde{Q}_n^F(t) - V\rho_d\varphi - V\rho_s\tilde{\omega}_{nm}(t)]|\Theta(t) \Bigg\},
\end{aligned} \quad (17)$$

where $B_1 = \frac{1}{2}\sum_{n\in \boldsymbol{N}}[(\lambda^{\max})^2(I^{\max} + [N-1]B^{\max})^2 + (\lambda^{\max})^2(E_n + [N-1]B^{\max}) + (\lambda^{\max}E_n)^2 + (\frac{Q_n^{B,\max} r_n^{\max}}{\lambda^{\min}\gamma^{\min}})^2]$ *is a finite constant, and* $B_2(t) = -\sum_{n\in \boldsymbol{N}}\sum_{i\in \boldsymbol{Q}_n^B(t)} r_n^i(t)Q_n^B(t)$ *is a known constant at time slot t since queue backlogs are known at slot t. The expectations are taken over randomness in system and policy.*

Proof. See Appendix A in the supplemental materials, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/2934103. □

Remark. The key idea of OLCD is to approximately minimize the upper bound of drift-plus-penalty, subject to constraints (7), (8), (9), (10), and (11). Note that in the RHS of (17), the offloaded workloads between two interacting edge clouds are counted bidirectionally, which is too complicated to address. By leveraging gap-preserving techniques [5], we first carry out the transformation to enable offloaded workloads counted in both directions combined into a single direction, so that it suffices to solve a separate decision problem for each edge cloud. We get the following new problem:

$$\begin{aligned}\min \mathbb{P}_1 = &\sum_{n\in \boldsymbol{N}}\mathbb{E}\left\{\sum_{i\in \boldsymbol{I}(t)} a_n^i(t)\tilde{Q}_n^F(t)|\Theta(t)\right\}\\ &+\sum_{n\in \boldsymbol{N}}\mathbb{E}\left\{\sum_{i\in \boldsymbol{Q}_n^F(t)}(\alpha_n^i(t)[\lambda^i Q_n^B(t) - \lambda^i\tilde{Q}_n^F(t) + V\rho_d d_n^i]\right.\\ &\quad -\sum_{m\in \boldsymbol{N}_n}\beta_{nm}^i(t)[\lambda^i\tilde{Q}_n^F(t) + \lambda^{\max}\lceil\tilde{Q}_n^F(t)\rceil^1\\ &\quad \left. - V\lambda^i\rho_d\varphi - V\lambda^i\rho_s\tilde{\omega}_{nm}(t)])|\Theta(t)\right\}\\ \text{s.t.}\quad &\text{Constraints (7)-(11)},\end{aligned} \tag{18}$$

where $\lceil z\rceil^1 = z$ if $z \leq 0$, and $\lceil z\rceil^1 = 0$ if $z > 0$.

Lemma 2. *Any feasible control algorithm that produces the optimal solution to* $\mathbb{P}_1$ *will also be a solution that achieves the minimum of the RHS of (17) within a constant gap.*

Proof. See Appendix B in the supplement, available online. □

Remark. Lemma 2 provides us with the convenience to consider only problem $\mathbb{P}_1$ hereafter, where offloading decisions are further decoupled for individual edge clouds. Actually, such practice facilitates the implementation of distributed and autonomous scenario, where at any slot, each edge cloud adjusts its own task admission and scheduling strategies for the minimum system cost.

To optimally solve $\mathbb{P}_1$ can not materialize without complete latency knowledge. That is, each edge cloud (or rather, Edge-F) should have a global latency knowledge for all queued tasks when processed in associated edge clouds, i.e., $\{d_m^i, \forall m \in \boldsymbol{N}_n \vee \{n\}, i \in \boldsymbol{Q}_n^F(t)\}$, which serves as the basis for online offloading control. In practice, however, as tasks are outsourced and processed at the network edge, completion latency usually presents high variability due to varying workload and resource levels, especially under stochastic task arrivals [21]. Hence, those *offloading solutions that focus on deterministic latency may not be feasible*. Instead, we adopt a learning-aided cooperative offloading approach that first harnesses online learning techniques to acquire latency prediction knowledge, and on this basis conducts online control policy.

To encourage high-quality service offerings, we further develop an accumulated trust update policy along with the performance-related incentive. Such social trust update, combined with latency prediction and online queue-based control, constitutes the key ideas of OLCD. If online control acts as core strategy based on predicted latency, then surely trust update guarantees the sound operation of cooperative offloading.

4.2 Latency Prediction Policy

Taking latency variability into account, the desired latency prediction policy is implemented based on delayed online learning.

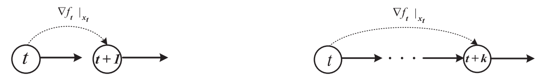
4.2.1 Preliminaries

1) *Completion Latency Prediction:* Our study highlights the intriguing relationship between latency variability and workload levels. Intuitively, for any task $i \in \boldsymbol{I}$, the completion latency in edge cloud $n \in \boldsymbol{N}$ is determined by the processing rate that n can provide. Under resource contention, if we know the workload to process in the Edge-B[8] of edge cloud n, $Q_n^B(t)$, we can compute the available processing rate $r_n^i(t)$ according to (4). Notice that OLCD is specially designed to serve computation-intensive task requests, whose processing often takes some amount of time to complete, from minutes to hours or even days [3]. As a result, task completion latency associated with edge cloud n will be affected by its workload levels for future consecutive slots. The problem is that edge clouds don't have a global knowledge of workload levels at all slots. To acquire completion latency knowledge, we seek the online learning technique to predict future workload information.

2) *Online Learning and Delayed Online Learning:* In online learning, a learner interacts with an environment over a sequence of consecutive rounds [31]. In round t, the learner chooses an action x_t from action space $\boldsymbol{X}$, and the environment delivers a cost function f_t. Thus the learner incurs loss $f_t(x_t)$. The online algorithm aims at minimizing the total loss over timespan T. One popular approach is to apply Online Gradient Descent (OGD) method to solve the loss minimization problem. Specifically, after choosing the tth action x_t, OGD computes the gradient $\nabla f_t|_{x_t}$ of the loss function at x_t, and chooses action $x_{t+1} = x_t - \eta\nabla f_t|_{x_t}$ in the subsequent round. OGD has proven useful for solving large-scale learning problems, and there has been much work on extending to parallel and distributed systems [32]. Notice that online learning assumes the loss function $f_t(x_t)$ (or feedback) is delivered and applied before choosing next action in round $t+1$.

But in our latency prediction policy, the processing duration makes task completion latency $\hat{d}_n^i$ predicted at slot t highly rely on workload levels for future consecutive slots (or rounds), i.e., $Q_n^B(t+1), Q_n^B(t+2), \ldots,$ where actual

8. In the following, we use "workload level" instead of "workload to process in the Edge-B" when there is no confusion.



(a) Online learning (b) Delayed online learning

Fig. 3. Impact of delayed feedback on learning process.

workload $Q_n^B(t+k)$ cannot be observed until slot $t+k$ comes. Such delayed feedback greatly undermines efficiency of online learning models that target immediate feedback. To this end, we employ Delayed Online Gradient Descent (DOGD) method, where loss function is not necessarily delivered before choosing next action and can be applied only after a delay of several slots. The impact of delayed feedback on learning process is depicted in Fig. 3. DOGD first proposed by Zinkevich is designed for distributed systems with delays between gradient computations and corresponding feedbacks [33], [34]. Formally, each slot t has a non-negative integer delay d_t. The feedback from t is delivered at the end of $t+d_t-1$ and can be applied at $t+d_t$. In standard online learning, $d_t = 1$ for all t.

4.2.2 DOGD-Based Computation Latency Prediction

The prediction model is parameterized by a vector $\boldsymbol{T}_t = \{t+1, \ldots, t+b^{\max}\}$, where prediction window size $b^{\max} = \max_{i \in \boldsymbol{I}} b^i$ is specified to accommodate user diverse demand in deadline. At slot t, each edge cloud $n \in \boldsymbol{N}$ predicts workload information, $\hat{\boldsymbol{Q}}_n^B(t) = \{\hat{Q}_n^B(\tau), \forall \tau \in \boldsymbol{T}_t\}$, where actual workload $Q_n^B(\tau)$ can be observed at the beginning of slot τ. Such learning window model approximates practical scenarios and is also used in [32]. Hence, workload prediction for $\boldsymbol{T}_t$ turns out to be *a set of delayed online learning processes*, where for each slot $\tau \in \boldsymbol{T}_t$, the feedback from t is delivered at the end of $\tau - 1$ and can be applied at slot τ.

We implement the DOGD method to provide effective prediction on future workload levels. To capture workload fluctuation, we assume $Q_n^{B,\max} = \max_{t \in \boldsymbol{T}} Q_n^B(t)$, which is provided by each edge cloud $n \in \boldsymbol{N}$ based on prior experience. To bound the sub-optimality gap in overall loss due to imperfect prediction for slot $\tau \in \boldsymbol{T}_t$, we construct a loss function

$$f_{n\tau}(\hat{Q}_n^B(\tau)) = \left|\hat{Q}_n^B(\tau) - Q_n^B(\tau)\right|, \tag{19}$$

which is a convex function on $\hat{Q}_n^B(\tau) \in [0, Q_n^{B,\max}]$. A loss minimization problem for any edge cloud n over all slots can be characterized as

$$\min_{\hat{Q}_n^B(\tau) \in [0, Q_n^{B,\max}]} \sum_{t \in \boldsymbol{T}} \sum_{\tau \in \boldsymbol{T}_t} f_{n\tau}(\hat{Q}_n^B(\tau)). \tag{20}$$

Due to the delayed feedback of $Q_n^B(\tau)$, loss function $f_{n\tau}(\hat{Q}_n^B(\tau))$ is not given before choosing the next predicted workload $\hat{Q}_n^B(\tau+1)$. The natural generalization of OGD to this delayed setting is to process loss functions and apply their gradients once they are delivered. Specifically, at any slot t, each edge cloud n makes a workload prediction $\hat{Q}_n^B(\tau)$ for each future slot $\tau \in \boldsymbol{T}_t$ based on the feedback that it has observed from t, and suffers the loss $f_{n\tau}(\hat{Q}_n^B(\tau))$. The update rule for each slot $\tau \in \boldsymbol{T}_t$ is as follows:

$$\hat{Q}_n^B(\tau+1) = \hat{Q}_n^B(\tau) - \eta_n \nabla f_{nt}|_{Q_n^B(t)}, \tag{21}$$

where step size η_n is typically set proportional to $\frac{1}{\sqrt{T+D}}$ (e.g., $\frac{Q_n^{B,\max}}{\sqrt{T+D}}$) with $D = \sum_{t \in \boldsymbol{T}} \sum_{\tau \in \boldsymbol{T}_t} d_\tau = \frac{1}{2} T d^{\max}[1 + d^{\max}]$ denoting the sum of delays over all slots.

Algorithm 1. Latency Prediction Algorithm at Slot t

```
1  for Each edge cloud $n \in \boldsymbol{N}$ do
2    $\eta_n \leftarrow \frac{Q_n^{B,\max}}{\sqrt{T+D}}$;
3    Observe actual computation workloads $Q_n^B(t)$ and share
       this information with trusted neighbors in $\boldsymbol{N}_n$;
4    for Each slot $\tau \in \boldsymbol{T}_t$ do
5      Derive predicted workload $\hat{Q}_n^B(\tau)$ by (21);
6      $\hat{r}_n(\tau) \leftarrow \frac{1}{\alpha} \beta^{\varsigma_n(\tau) - \hat{Q}_n^B(\tau)}$;
7    for Each task $i \in \boldsymbol{Q}_n^F(t)$ do
8      for Each edge cloud $m \in \boldsymbol{N}_n \vee \{n\}$ do
9        $\tilde{\lambda}^i \leftarrow \lambda^i$; $s \leftarrow t$;
10       while $\tilde{\lambda}^i > 0$ do
11         $\tilde{r}_n^i(s) \leftarrow \min\{\frac{\hat{r}_n(s)\gamma^{\min}}{\gamma^i Q_n^B(s)}, \tilde{\lambda}^i\}$;
12         $\tilde{\lambda}^i \leftarrow \tilde{\lambda}^i - \tilde{r}_n^i(s)$; $s \leftarrow s+1$;
13       $\hat{d}_m^i \leftarrow s - t$.
```

As shown in Algorithm 1, our latency prediction policy employs DOGD to compute future workload levels for prediction window, $\hat{\boldsymbol{Q}}_n^B(t) = \{\hat{Q}_n^B(\tau), \forall \tau \in \boldsymbol{T}_t\}$, where $\hat{Q}_n^B(\tau)$ is predicted by minimizing $f_{n\tau}(\hat{Q}_n^B(\tau))$ (line 5) based on the feedback that it has observed from t (line 3). For better cooperative offloading control, each edge cloud is expected to share workload information with its trusted neighbors. Given prediction $\hat{Q}_n^B(\tau)$ for all slots $\tau \in \boldsymbol{T}_t$, we can obtain the available processing rate that edge cloud $n \in \boldsymbol{N}$ can provide, $\hat{\boldsymbol{R}}_n(t) = \{\hat{r}_n(\tau), \forall \tau \in \boldsymbol{T}_t\}$ (line 6). With collective rate knowledge $\{\hat{\boldsymbol{R}}_m(t), \forall m \in \{n\} \vee \boldsymbol{N}_n\}$, edge cloud n can thus estimate the corresponding computation latency for each task $i \in \boldsymbol{Q}_n^F(t)$ when processed in associated edge clouds, i.e., $\hat{\boldsymbol{d}}_n^i = \{\hat{d}_m^i, \forall m \in \{n\} \vee \boldsymbol{N}_n\}$ (lines 7-13). In the following, we assume that completion latency knowledge (or more precisely, future workload levels) can be predicted accurately. The case with prediction errors will be discussed later.

4.2.3 Regret Analysis

We next analyze the performance of Algorithm 1 in predicting future workloads $\hat{\boldsymbol{Q}}_n^B(t)$ by computing the regret bound. Let $\boldsymbol{Q}_n^{B,*}(t) = \{Q_n^{B,*}(\tau), \forall \tau \in \boldsymbol{T}_t\}$ be the best static predictor in hindsight obtained by the strategy in [35] with full knowledge of workloads. We have

$$\text{Regret}_n^T(DOGD) = \sum_{t \in \boldsymbol{T}} \sum_{\tau \in \boldsymbol{T}_t} \Big[f_{n\tau}\Big(\hat{Q}_n^B(\tau)\Big) - f_{n\tau}(Q_n^{B*}(\tau)) \Big]. \tag{22}$$

The following theorem upper-bounds the overall regret.

Theorem 1. *The regret of DOGD in Algorithm 1 in predicting future workloads with respect to the best static prediction strategy that uses $\boldsymbol{Q}_n^{B,*}(t), \forall t \in \boldsymbol{T}$, is upper bounded by*

$$\begin{aligned} \text{Regret}^T(DOGD) &= \sum_{n \in \boldsymbol{N}} \text{Regret}_n^T(DOGD) \\ &\le \sum_{n \in \boldsymbol{N}} \frac{b^{\max}}{2\eta_n} + \eta_n \left[\frac{T b^{\max}}{2} + 2D \right]. \end{aligned} \tag{23}$$

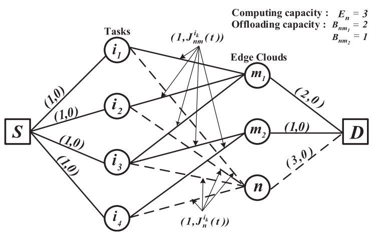


Fig. 4. An example of reduction of the optimal task scheduling instance problem to MCMF.

Proof. See Appendix C in the supplemental materials, available online. □

4.3 Online Queue-Based Control Policy

Given latency prediction knowledge, OLCD employs online queue-based control policy to solve per-slot joint admission and scheduling subproblem with a carefully designed objective specified for individual edge clouds. The solutions for all subproblems constitute a feasible solution to the original control problem $\mathbb{P}_1$.

4.3.1 Task Admission Control

Task admission decisions can be made by minimizing the first term of the RHS of (18). Since admission decisions of different edge clouds are independent, we can concurrently obtain $\boldsymbol{a}_n(t) = \{a^i_n(t), \forall i \in \boldsymbol{I}(t)\}$ by solving

$$\begin{aligned} \min_{\boldsymbol{a}_n(t)} \quad & \sum_{i \in \boldsymbol{I}(t)} a^i_n(t)[Q^F_n(t) - \theta_n] \\ \text{s.t.} \quad & 0 \le a^i_n(t) \le A^i_n(t). \end{aligned} \tag{24}$$

The optimal solution can thus reduce to a simple threshold rule

$$a^i_n(t) = \begin{cases} A^i_n(t), & Q^F_n(t) \le \theta_n, \\ 0, & \text{otherwise.} \end{cases} \tag{25}$$

Remark. For edge cloud n, newly arrived task i will be admitted into the system with the increase of workload $A^i_n(t)$ when backlog $Q^F_n(t)$ is no larger than threshold θ_n; otherwise it will be rejected for system stability. The intuitive behind no task admission is that, current task arrivals go beyond the response capability of Edge-F, and the best way to avoid long waits for scheduling is to seek for another trusted edge cloud that covers its vicinity to associate.

4.3.2 Task Scheduling Control

In OLCD, each Edge-F is responsible for scheduling tasks to potential edge clouds with high service performance. By gap-preserving transformation, scheduling decisions (including local computing and cooperative offloading) of different edge clouds are independent from each other. For edge cloud n, decisions on $\alpha^i_n(t)$ and $\beta^i_{nm}(t)$ can be determined by solving

$$\begin{aligned} \min_{\boldsymbol{s}_n(t)} \quad & \sum_{i \in \boldsymbol{Q}^F_n(t)} (\alpha^i_n(t)[V\rho_d d^i_n + \lambda^i Q^B_n(t) - \lambda^i \tilde{Q}^F_n(t)] \\ & + \sum_{m \in \boldsymbol{N}_n} \beta^i_{nm}(t)[V\lambda^i\rho_d\varphi + V\lambda^i\rho_s\tilde{\omega}_{nm}(t) \\ & - \lambda^i\tilde{Q}^F_n(t) - \lambda^{\max}\lceil\tilde{Q}^F_n(t)\rceil^1]) \\ \text{s.t.} \quad & \text{Constraints (7)-(11).} \end{aligned} \tag{26}$$

The scheduling decision problem in (26) is essentially an assignment problem: how to assign available computing capabilities (i.e., local computing resources or social trust relationships) to tasks for minimum system cost under constraints (7), (8), (9), (10), and (11)? Such objective function contains two parts: one for local computing $J^i_n(t) = \alpha^i_n(t)[V\rho_d d^i_n + \lambda^i Q^B_n(t) - \lambda^i\tilde{Q}^F_n(t)]$ involving computation latency, and one for a set of cooperative offloading $J^i_{nm}(t) = \beta^i_{nm}(t)[V\lambda^i\rho_d\varphi + V\lambda^i\rho_s\tilde{\omega}_{nm}(t) - \lambda^i\tilde{Q}^F_n(t) - \lambda^{\max}\lceil\tilde{Q}^F_n(t)\rceil^1]$, $\forall m \in \boldsymbol{N}_n$ taking transmission latency and trust risk into account. By provisioning computation, transmission and trust services, the optimal solution to (26) is expected to achieve latency versus trust risk tradeoff. Moreover, constraints (7), (10) and (11) focus on a balanced scheduling plan, while (8) and (9) suggest the prerequisites on beneficial offloading and trusted offloading.

Consider an instantaneous assignment problem for edge cloud $n \in \boldsymbol{N}$ with trusted neighbor set $\boldsymbol{N}_n = \{m_1, m_2, \ldots\}$ and queued task set $\boldsymbol{Q}^F_n(t) = \{i_1, i_2, \ldots\}$. Such problem can be reduced to a minimum cost maximum flow (MCMF) problem [36], where constraints (7), (8), (9), (10), and (11) guarantee tasks are properly assigned. Let $\mathcal{G}_{nt} = (\boldsymbol{Q}^F_n(t) \vee \boldsymbol{N}_n \vee \{n, S, D\}, \mathcal{E})$ denote the flow network graph for edge cloud n at slot t, where $\mathcal{E}$ is the set of edges, and vertices n, S, D represent local edge cloud, source and destination nodes. There are $|\boldsymbol{Q}^F_n(t)|$ edges connecting S to all nodes $i_k \in \boldsymbol{Q}^F_n(t)$ with capacity 1 since every task can be offloaded to at most one edge cloud at one slot. There are also $|\boldsymbol{N}_n|$ edges connecting all nodes $m_l \in \boldsymbol{N}_n$ to D with capacity B_{nm_l} under offloading capacity constraint (11). Similarly, we connect n to D, whose capacity is set to E_n due to limited processing capacity (10). For each task i_k, we add an edge from node i_k to all nodes $m_l \in \boldsymbol{N}_n$ which satisfy (8) and (9), indicating that the selected neighbors must be trusted enough and can provide lower latency, which is vital to realizing QoS guarantee for cooperative offloading. The capacity and cost of each edge are set to 1 and $J^{i_k}_{nm_l}(t)$. Since tasks can also be processed locally, we add an edge from all nodes i_k to node n with capacity 1 and cost $J^{i_k}_n(t)$. We set the cost of all other edges in $\mathcal{E}$ to 0. Consequently, by finding the minimum-cost maximum flow in $\mathcal{G}_{nt}$, edge cloud n is expected to serve task requests at slot t in a best-effort manner with the minimum system cost in (26).

Take trusted cooperative offloading system illustrated in Fig. 1 for example. At slot t, with the help of edge clouds B and E (i.e., $\boldsymbol{N}_n = \{m_1, m_2\}$), D needs to serve 4 task requests queued in Edge-F (i.e., $\boldsymbol{Q}^F_n(t) = \{i_1, i_2, i_3, i_4\}$). Each task i_k is associated with one desired trust value. The corresponding flow network graph $\mathcal{G}_{nt}$ is shown in Fig. 4. In particular, for task i_1, only edge cloud B satisfies its trust demand while providing lower latency. Hence, node i_1 can transfer flow to nodes m_1 and n with the cost $J^{i_1}_{nm_1}(t)$ and $J^{i_1}_n(t)$. The capacities for two edges are both 1.

By reducing to the MCMF problem on $\mathcal{G}_{nt}$, we can leverage any algorithm for that problem to determine the optimal offloading decisions in polynomial time. One of the well-known techniques is the Successive Shortest Path (SSP) algorithm proposed by Edmonds and Karp [37].

4.3.3 Queue Update Control

The queue backlogs associated with each edge cloud $n \in \boldsymbol{N}$, $Q_n^F(t)$ and $Q_n^B(t)$, can be updated according to (5) and (6), with the above task admission and scheduling control strategies.

4.4 Accumulated Trust Update Policy

We establish a connection between cooperative offloading design and social trust management, which mainly involves two aspects: trusted neighbor set update (specified earlier) and trust value update. Intuitively, edge clouds may hesitate to interact with less trusted or unknown ones due to trust risks in MEC collaboration. To encourage high-quality service offerings and combat uncooperative strategic behaviors, accumulated trust value update policy along with the intuition of "performance-related incentive" is necessary [38]. Specifically, the MEC service provider collects the locally-generated service result once a task is completed, and dynamically aggregates it to yield the global trust values for all associated cooperative edge clouds. As a result, edge clouds are motivated to build good trust relationships for increased chances of being assisted and selected in future offloading.

4.4.1 Service Valuation for A Single Interaction

For any task $i \in \boldsymbol{I}$, the provided offloading service corresponds to the process from being admitted into the system at slot t_0^i to being served completely at slot $t^i = t_K^i$ after multiple offloading hops. All edge clouds who contribute to the offloading service (i.e., $\boldsymbol{N}^i$) can be easily located along the offloading path $\pi^i = \{n_0^i, \dots, n_K^i\}$. Given i's deadline demand b^i and actual completion latency $d^{i,*} = t^i - t_0^i$, the provider assigns a performance-related rating $Score^i \in [0, 1]$ to this service offering. Intuitively, the smaller latency $d^{i,*}$ is, the higher rating $Score^i$ would be. One simple example is as follows:

$$Score^i = \begin{cases} 1 - \frac{d^{i,*}}{2b^i}, & 0 \leq d^{i,*} < 2b^i, \\ 0, & d^{i,*} \geq 2b^i. \end{cases} \tag{27}$$

4.4.2 Aging-Based Trust Update

At the end of each slot t, the MEC service provider aggregates current and historical interaction results to update global trust values among all associated edge clouds. Clearly, recent interactions can reflect edge clouds' future cooperative performance more accurately than earlier interactions [19]. Hence, in calculating social trust value between any cooperative edge clouds, we weight the rating for each interaction according to its age, i.e., number of time slots that have passed since it happened. That is,

$$\omega_{nm}(t) = \frac{1}{\mathcal{I}_{nm}(t)} \sum_{k=1}^{\mathcal{I}_{nm}(t)} Score_k \chi^{t_k}, \tag{28}$$

where $\mathcal{I}_{nm}(t)$ denotes the number of interactions between edge clouds n and m until slot t, and $Score_k = Score^i$ is the rating of the kth interaction associated with user i. The aging coefficient $\chi \in (0, 1)$ is used to control the weights of interaction ratings by their ages denoted by t_k, making the weights assigned to recent interactions heavier than older ones. The trust values between two edge clouds that have no previous interactions are set to 0. Notice that the above is just one common way to evaluate social trust between edge clouds. Actually, there are many other ways for trust evaluation in the literature.

4.5 Performance Analysis

OLCD contains three main components: a latency predictor, an online queue-based controller and a social trust manager. All of them work online together and influence each other. The complete OLCD that enables autonomous coordination among edge clouds is summarized in Algorithm 2. At any slot $t \in \boldsymbol{T}$, each edge cloud employs online queue-based control in a distributed manner. The Edge-F is responsible for task admission (lines 1-4) and scheduling (lines 9-14), any of which that realizes the optimized workload allocation is an optimal online control profile for per-slot minimization problem $\mathbb{P}_1$. Specifically, for each queued task, Edge-F applies Algorithm 1 to predict completion latency in associated edge clouds, and then determines scheduling decisions via SSP. Meanwhile, Edge-B processes tasks dispatched from Edge-F (lines 6-8). As the central trust manager, MEC service provider is expected to perform accumulated trust update to encourage high-quality service offerings (lines 16-21). If online scheduling acts as core strategy based on latency prediction results, then surely trust update guarantees the sound operation of cooperative offloading.

Algorithm 2. OLCD at Slot t

```
1  for Each task $i \in \boldsymbol{I}(t)$ do
2      for Each edge cloud $n \in \boldsymbol{N}$ do
3          if $n_0^i = n$ then
4              Derive admission decision $a_n^i(t)$ by (25);
5  for Each edge cloud $n \in \boldsymbol{N}$ do
6      for Each task $i \in \boldsymbol{Q}_n^B(t)$ do
7          Derive processing rate $r_n^i(t)$ by (4);
8          $\lambda^i(:,t) \leftarrow \lambda^i(:,t) - r_n^i(t)$;
9      for Each task $i \in \boldsymbol{Q}_n^F(t)$ do
10         for Each edge cloud $m \in \boldsymbol{N}_n \vee \{n\}$ do
11             Apply Algorithm 1 to obtain estimated latency $\hat{d}_m^i$;
12         Apply SSP to obtain decisions $\alpha_n^i(t)$ and $\beta_{nm}^i(t)$;
13         if $\alpha_n^i(t) = 1$ then
14             $\lambda^i(:,t) \leftarrow \lambda^i$;
15     Update $Q_n^F(t)$, $Q_n^B(t)$ according to (5), (6);
16 for Each task $i \in \boldsymbol{I}$ do
17     if $\lambda^i(:,t) = 0$ then
18         Update trust value $\omega_{nm}(t), \forall n, m \in \boldsymbol{N}^i$ by (28);
19         $\boldsymbol{I} \leftarrow \boldsymbol{I} \setminus \{i\}$.
20 for Each edge cloud $n \in \boldsymbol{N}$ do
21     Update trusted neighbor set $\boldsymbol{N}_n$;
```

Complexity. Take OLCD implemented at slot t as an example. The running time of task admission is $O(NI(t))$. There are $O(N)$ iterations for online predictive scheduling, within each of which OLCD first considers $v_1 = |\boldsymbol{N}_n| + 1$ candidate scheduling strategies for each task queued in n's Edge-F, predicts computation latency for each strategy using $O(|\boldsymbol{T}_t| + v_1|\boldsymbol{Q}_n^F(t)|)$-complexity Algorithm 1 and applies SSP to obtain scheduling control with running time $\min\{O(v_2^2 f^*), O(v_2^3 c^*)\}$. Here $v_2 = v_1 + 2 + |\boldsymbol{Q}_n^F(t)|$ is the total number of vertices on $\mathcal{G}_{nt}$, f^* is the derived maximum flow and c^* is the corresponding minimum cost. While for tasks queued in Edge-B, the amount of residual workloads is computed with complexity of $O(|\boldsymbol{Q}_n^B(t)|)$. Therefore, the overall complexity of online scheduling is $\min\{O(N[v_1|\boldsymbol{T}_t| + v_1^2|\boldsymbol{Q}_n^F(t)| + v_2^2 f^*]),\ O(N[v_1|\boldsymbol{T}_t| + v_1^2|\boldsymbol{Q}_n^F(t)| + v_2^3 c^*])\}$. The running time of calculating trust values for associated cooperative edge clouds is $O(I\sum_{n\in\boldsymbol{N}}\sum_{m\in\boldsymbol{N}_n}\mathcal{I}_{nm}(t))$. Thus, OLCD can find a near-optimal solution in polynomial time.

OLCD always exhibits obvious dynamic characteristics, especially under stochastic task arrivals. Next, we prove that even in the dynamic case, OLCD achieves close-to-optimal system cost while guaranteeing system stability and robustness against prediction errors. Before that, we define the perturbation parameter as

$$\theta_n \overset{\Delta}{=} 2\lambda^{\max}(E_n + [N-1]B^{\max}), \tag{29}$$

which can be easily determined since it only requires the knowledge of maximum coefficient of transmission latency cost, maximum number of tasks dispatched to the Edge-B and maximum number of tasks offloaded away, and requires no statistical knowledge of system dynamics, e.g., $Q_n^F(t)$ and $Q_n^B(t)$. Such feature is desirable for practical implementations.

Theorem 2. *Suppose* $Q_n^F(0) = \theta_n, Q_n^B(0) = 0, \forall n \in \boldsymbol{N}$. *If admission decisions* $\boldsymbol{a}(t)$, *scheduling decisions* $\boldsymbol{s}(t)$ *and queue updates are performed by Algorithm 2 with* $V > 0$, *we obtain the following properties of OLCD:*

$$\lim_{T\to\infty}\sup\frac{1}{T}\sum_{t=0}^{T-1}\sum_{n=1}^{N}\mathbb{E}\{Q_n^F(t) + Q_n^B(t)\} \le \theta_n + \frac{B_1 + V[C^{\max} - C^*]}{\epsilon}, \tag{30}$$

$$\lim_{T\to\infty}\sup\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{C_{\mathrm{OLCD}}(t)\} \le C^* + \frac{B_1}{V}, \tag{31}$$

where B_1 *is defined as in Lemma 1,* C^* *is the optimal system cost of TCO in (14),* $C^{\max} = N\rho_d(Ed^{C,\max} + [N-1]B\varphi\lambda^{\max}) + [N-1]NB\rho_s\lambda^{\max}$ *is the largest system cost, and* $\epsilon > 0$ *is a constant denoting the long-term computation surplus achieved by some stationary strategy.*

Proof. See Appendix D in the supplemental materials, available online. □

Remark. This theorem specifies a $[O(1/V), O(V)]$ tradeoff between cost optimality and queueing delay. According to Little's law, the average queueing delay including transmission and computation latency, is proportional to queue backlogs. The long-term queue backlog bound in (30) indicates that the overall average queue backlog grows linearly with V. OLCD asymptotically achieves the optimal cost performance of the offline problem by letting $V \to \infty$. However, the optimal system cost is achieved at the expense of larger transmission and computation latency. Since larger Edge-F and Edge-B queues are required to stabilize the system, the convergence is postponed.

With the above time-averaged system performance, we next consider more realistic scenario. What happens when the scheduling decisions are made based on predicted workloads $\hat{Q}_n^B(t)$ that differ from actual workloads $Q_n^B(t)$? The following theorem shows that OLCD is robust against workload prediction errors.

Theorem 3. *Suppose there exists a constant* ξ *such that at all slots* $t > 0$, $|\hat{Q}_n^B(t) - Q_n^B(t)| \le \xi$ *holds. Under OLCD, we have*

$$\lim_{T\to\infty}\sup\frac{1}{T}\sum_{t=0}^{T-1}\sum_{n=1}^{N}\mathbb{E}\{Q_n^F(t) + Q_n^B(t)\} \le \theta_n + \frac{B_3 + V[C^{\max} - C^*]}{\epsilon}, \tag{32}$$

$$\lim_{T\to\infty}\sup\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{C_{\mathrm{OLCD}}(t)\} \le C^* + \frac{B_3}{V}, \tag{33}$$

where $B_3 = B_1 + \xi\sum_{n\in\boldsymbol{N}}[\lambda^{\max}E_n + \frac{Q_n^{B,\max}r_n^{\max}}{\lambda^{\min}\gamma^{\min}}]$.

Proof. See Appendix E in the supplemental materials, available online. □

Remark. Comparing (31) and (33), we conclude that with inaccurate workload prediction, larger V is desired to achieve the same average system cost as that with accurate information. However, such practice may result in higher average queue backlogs, which can be observed by comparing (30) and (32). Therefore, OLCD works even with inaccurate workload prediction but its robustness is achieved at the expense of a decreased stability.

4.6 Discussions

Discussions about End Devices' Computing Capability. Our work focuses on cooperative offloading or resource sharing among edge clouds. Existing researches mostly make a simplifying assumption that user requests are entirely offloaded from end devices to requesting edge clouds [4], [5], [7]. In practice, however, end devices usually possess increasing computing capability and can execute complex computation tasks. Accordingly, a combination of local device computing and networked resource sharing empowers users with multiple task execution approaches, including local mobile computing, D2D offloading, direct edge cloud offloading and D2D-assisted edge cloud offloading. It would be worthwhile to further study how to motivate efficient cooperations among end devices. Since mobile devices are carried or owned by users, it is promising to leverage intrinsic social ties among users as cooperation incentive. We believe that this will serve as the cornerstone for socially-motivated collaborative MEC systems.

Discussions about User Mobility. In this work, we mainly talk about a trusted cooperative offloading mechanism within specific geographic regions. By leveraging cooperations among edge clouds, networked resource sharing can be realized via multi-hop offloading. In this case, user mobility does not affect task offloading among edge clouds since the correspondence between users and requesting edge clouds remains the same. Either considering user mobility or not has no effect on offloading control. Hence, we choose the simplified version, i.e., static scenario without user mobility. But if users are allowed to connect to other edge clouds, that's another story. When any user moves across different areas, its service usually needs to be migrated to follow it so that the benefits of cooperative offloading are maintained. Moreover, transmission latency between the user and the edge cloud that host its service

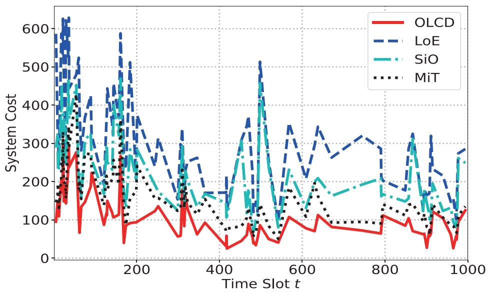


Fig. 5. System cost versus T.

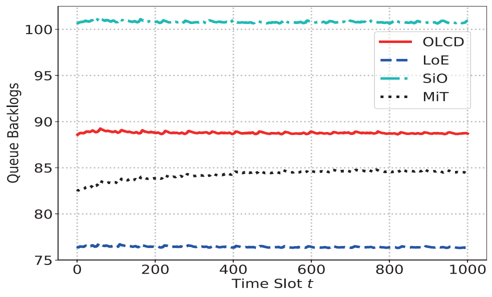


Fig. 6. Queue backlogs versus T.

may also decrease. A key challenge lies in when and where to migrate the service, which has been studied in earlier works [39]. Notice that a full analysis of dynamic service migration under user mobility effect is out of the scope of this work. It surely will be interesting for future research to study user mobility-driven cooperative offloading.

5 Performance Evaluation

5.1 Simulation Setup

We envision a MEC system deployed in a commercial complex where business tenants deploy edge clouds to serve employees. We simulate a $300\ \mathrm{m} \times 300\ \mathrm{m}$ commercial complex with 100 edge clouds[9] distributed by homogeneous Poisson Point Process, which is commonly used in previous studies [8]. Unless otherwise specified, each edge cloud is connected to 7 neighbors on average with initial trust value uniformly distributed in [0.9, 1], and the trust decaying factor ∂ is set to 0.05. Suppose both local computing capacity and offloading capacity are uniformly distributed in [3, 6]. Inspired by [4], we set the skewness parameter $x = 1.04$, processing speed of fully loaded edge clouds $y = 200$, and service limitation $\varsigma_n(t)$ uniformly distributed in [120, 150]. The backhaul transmission latency coefficient φ is set to $0.1\ \mathrm{sec/Kb}$ [25].

Consider the realistic job trace from Google cluster [41] as computation-intensive tasks. For each job trace, the task is specified by a tuple including time in seconds since the start of data collection, consumptions of CPU and memory, and task type. Here, the task type chosen from $\{0, 1, 2, 3\}$ is determined by conducting workload characterizations. Notice that our study highlights the role of tasks with diverse trust demands. It is acceptable to regard that different task types correspond to heterogeneous trust demands $\{0, 0.2, 0.4, 0.6\}$, ranging from completely public applications (i.e., $\sigma^i = 0$) to privacy-sensitive ones (i.e., $\sigma^i = 0.6$). We adjust tasks' inter-arrival time based on Poisson distribution to accommodate stochastic arrival and the deadlines are uniformly distributed in [10,15]. The expected number of CPU cycles and expected input data size per-task are $50M$ and $0.25\ \mathrm{Mb}$ with γ^i uniformly distributed in [100, 300]. Motivated by the fact that the popularity follows heavy-tailed distribution, we use Zipf distribution to capture unbalanced workloads [42], and each user is assigned to its nearest edge cloud with trust value 1.

9. To force edge clouds deployed by different tenants, the scale of commercial complex should be large enough ($> 100{,}000\ \mathrm{ft}^2$ [8]). Only in this way can the potential of cooperations among edge clouds be exerted fully. According to the 2012 statistics of IREM [40], a typical commercial complex within an area around $135{,}710\ \mathrm{ft}^2$ was occupied by 10 tenants on average. Hence, it is reasonable to assume that there are 100 edge clouds deployed in a $300\ \mathrm{m} \times 300\ \mathrm{m}$ commercial complex ($968{,}751.938\ \mathrm{ft}^2$).

We implement the OLCD mechanism for $T = 1000$ slots and compare it with three benchmarks: (1) Local Execution (LoE) [13]: edge clouds only process locally arrived tasks with the highest social trust; (2) Single-hop Offloading (SiO) [18]: trusted cooperative offloading is confined within one hop, which achieves lower completion latency while guaranteeing an acceptable social trust; (3) Multi-hop offloading ignorant of Trust (MiT) [15]: multi-hop offloading is enabled to minimize completion latency regardless of trust relationships among edge clouds. All algorithms are implemented through Python code in Anaconda Simulator, and evaluated on a machine with Windows 64 bits, 3.2 GHz Intel Core $i5$ Processor, and 16 GB 1600 MHz DDR3 Memory.

5.2 Evaluation Results

5.2.1 Run-Time Performance

We first illustrate the performance comparison of system cost, queue backlogs and satisfaction ratio in terms of timespan T.

Fig. 5 presents that our proposed OLCD achieves the lowest system cost with high speed of convergence. The intuitive is that in OLCD, tasks are often offloaded multiple hops away and finally fulfilled by powerful edge clouds. While in LoE, tasks can only be processed locally. Under limited computing capacity, this practice may result in high computation latency, even with the lowest trust risks and transmission latency. That's why the system cost in LoE is the highest. Due to failure of fully exploiting computing capabilities of cooperative edge clouds, one-hop SiO is inferior to multi-hop MiT and OLCD in reducing system cost. Compared to OLCD, MiT ignores social trust relationships among edge clouds when making offloading decisions, thus leading to higher trust risk cost. The results reveal what benefit do multi-hop offloading underlying social trust relationships bring to MEC system.

From Fig. 6, we can observe that the queue backlogs under these mechanisms converge to steady-state levels with almost no ripples. That is, both convergence and system stability are maintained. Under perturbed Lyapunov optimization, $Q_n^F(t)$ is "pushed" towards θ_n to avoid edge cloud n from wasting computing resources and potentials of social trust relationships. In addition, the stochastic control processes will also lead to changes in $Q_n^B(t)$ accordingly. That's why the queue backlogs fluctuate around one fixed value.

Fig. 7 shows that OLCD is superior to others in guaranteeing high satisfaction ratio denoted by the proportion of offloaded tasks that are able to meet the deadlines. In general,

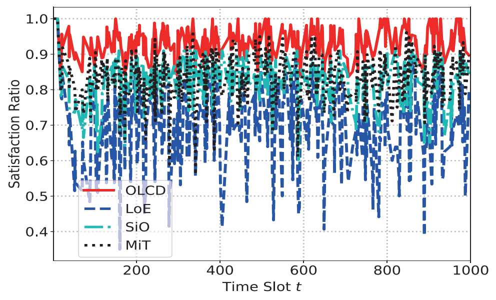


Fig. 7. Satisfaction ratio versus T.

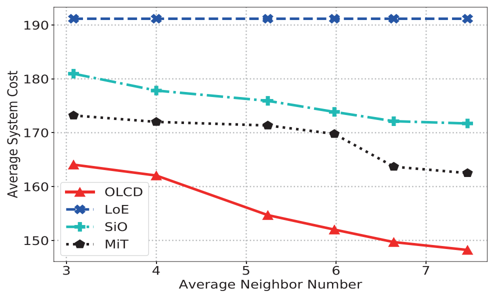


Fig. 8. Average system cost versus average neighbor number.

TABLE 2
Average Running Time for the OLCD Mechanism versus Benchmarks

Algorithms	Average Running Time
OLCD	201.59 sec
LoE	80.31 sec
SiO	190.62 sec
MiT	962.28 sec

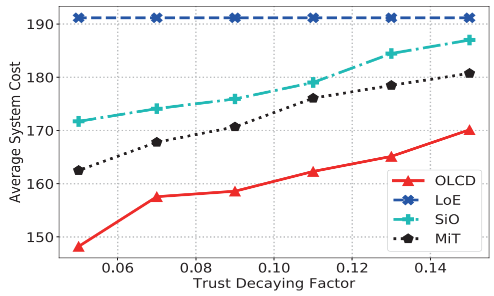


Fig. 9. Average system cost versus trust decaying factor ∂.

low satisfaction ratio suggests poor offloading controls and under-utilized collaborative computing potentials. LoE achieves the lowest satisfaction ratio. That's because tasks in LoE can only be processed locally, and it is hard for resource-constrained edge clouds to fulfill all tasks alone, especially under unbalanced workloads. Compared to multi-hop MiT and OLCD, the satisfaction ratio of single-hop SiO is lower since it fails to fully exploit computing resources of multi-hop neighbors. MiT achieves similar satisfaction ratio to OLCD with small ratio gap, since offloading control in MiT is performed regardless of social trust among cooperative edge clouds and tasks are likely to be assigned to untrusted edge clouds, increasing the risks of missing deadlines. It indicates that historic interactions can partly reflect the quality of current offloading service offering.

Table 2 presents that these mechanisms share different average running time. Since there is no need for LoE to interact with neighbors in local computing, LoE is very fast and the other three solutions are relatively slow. Compared to OLCD, MiT is inferior because without trust demand constraint, there exist more complex interactions between edge clouds in multi-hop offloading. Taking Figs. 5 and 6 together, we observe that OLCD provides a good trade-off between system performance and computational overhead, which serves near-optimal system cost and queue backlog with relatively low running time.

5.2.2 Trust Service Performance

Fig. 8 demonstrates the comparison of average system cost in terms of *trust propagation intensity* captured by average neighbors per edge cloud. As expected, trust propagation intensity has no effect on LoE since tasks in LoE can only be processed locally. As the average neighbor number increases, tasks are more likely to be offloaded to powerful edge clouds, facilitating exploring collaborative computing potentials. Thus, system cost in other mechanisms decreases. Without one-hop restrictions like SiO, OLCD and MiT are particularly influenced by trust propagation intensity under multi-hop offloading. It's suggested that edge cloud should be encouraged to build social trust relationships since high-intensity trust propagation always yields a performance boost.

Trusted cooperative offloading services can't fully exert its function with restrained trust propagation length, which is closely related to trust decaying factor ∂. The larger ∂ is, the shorter trust propagation would be. Fig. 9 illustrates how average system cost changes with ∂. It can be seen that the overall trend is increasing, except for LoE independent of trust propagation. The average system cost in OLCD is lower than that in SiO and MiT, and the cost gap decreases with ∂. When there is little decaying (e.g., $\partial = 0.05$), tasks are likely to be offloaded multiple hops away under less limitation on trust propagation, thus giving full play to the superiorities of cooperative offloading. With more powerful edge clouds involved, the system cost is decreased, especially for multi-hop OLCD and MiT. Compared to MiT, OLCD achieves lower system cost since it can realize optimal online control for multi-hop offloading, even under restrained propagation length. As ∂ increases, edge clouds will be conservative with cooperative offloading, and cooperating with indirectly connected neighbors always means high trust risks for them. Hence, multi-hop offloading has little impact on system cost and the cost gap is small.

Fig. 10 shows the average ratio of completion latency cost and trust risk cost with varying maximum offloading hops. As the number of hops increases, tasks are more likely to be offloaded to powerful edge clouds, yielding lower computation latency, but at the expense of high social trust risks and communication latency. That's why latency cost ratio has the trends to decrease, but to increase in trust risk cost ratio, i.e., there exists a tradeoff between completion latency cost and trust risk cost. This result reconfirms the benefits in exploiting the co-provisioning of computation, transmission and trust services in MEC.

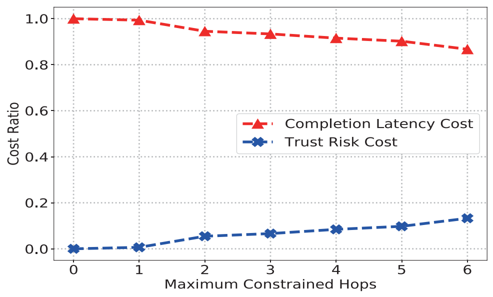


Fig. 10. Latency cost and trust risk cost versus maximum hops.

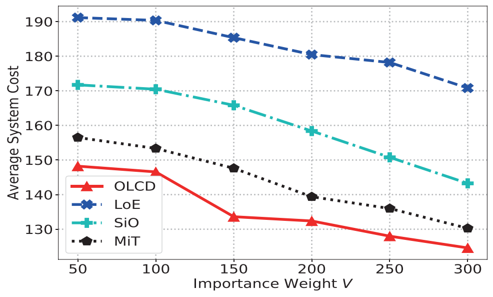


Fig. 11. Average system cost versus importance weight V.

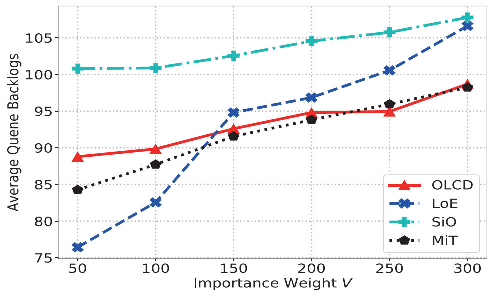


Fig. 12. Average queue backlogs versus importance weight V.

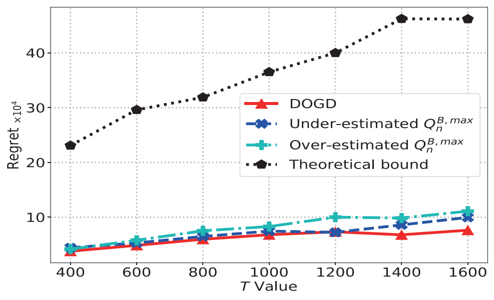


Fig. 13. Impact of estimated $Q_n^{B,\max}$ on the regret of DOGD.

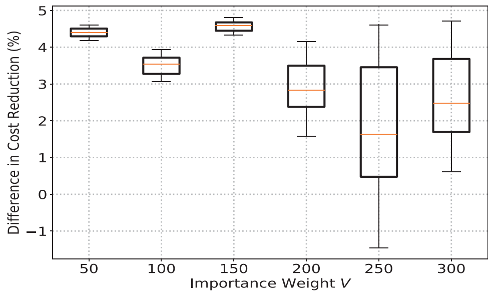


Fig. 14. Differences in cost increase versus V.

5.2.3 Impact of Parameters

We next show the comparison of system cost and queue backlogs in terms of importance weight V, which is introduced to capture the heterogeneity in how much emphasis put on cost minimization. From Fig. 11, we can see that as V increases, the system cost under these mechanisms decreases and OLCD achieves the lowest cost. The intuition is that high V typically means highlighting cost reduction more in online offloading control. OLCD is superior in achieving a trade-off between completion latency and trust risks. Specifically, the offloading cost in LoE greatly decreases with V. That's because with no interactions or impact from other edge clouds, LoE is more sensitive to V in balancing queue stability and offloading cost. As shown in Fig. 12, the average queue backlogs under these mechanisms increase with V, since compared to cost reduction, more attention is paid to queue stability. Combining these two figures, we can see that as V increases, the lower system cost is realized at the expense of larger queue backlogs. It's indicated that there exists a $[O(1/V), O(V)]$ trade-off between system cost and queue backlogs, which is consistent with Theorem 2.

Fig. 13 illustrates the impact of estimation $Q_n^{B,\max}$ on the regret of DOGD for delayed online learning. In particular, we deploy Algorithm 1 using $Q_n^{B,\max}$, $0.8Q_n^{B,\max}$ and $1.2Q_n^{B,\max}$ to compute the learning rate. We observe that the regrets in all cases are close, indicating both over-estimation (i.e., the case for $1.2Q_n^{B,\max}$) and under-estimation (i.e., the case for $0.8Q_n^{B,\max}$) of $Q_n^{B,\max}$ have a small effect on the regret of DOGD. Moreover, all regrets are always less than the theoretical bound given in Theorem 1.

5.2.4 Effectiveness of Workload Prediction

We further examine the impact of prediction errors on offloading service performance. At each slot, we add a random error ($\pm 30\%$, uniformly distributed) to the amount of predicted workloads queued in all edge clouds and then conduct OLCD on such error dataset. For any edge cloud, exact workload information for one slot is known only when that slot comes. Using results on the original datasets as baseline, we present the difference in average system cost increase and average queue backlog increase due to the injected prediction errors with varying importance weight V. As shown in Fig. 14, we observe that prediction errors result in the increased cost within the range of -1.5 to 5.5 percent. For all V we experiment with, the difference in average queue backlog increase is between -6 and 6 percent, as illustrated in Fig. 15. Combining these two figures coincides with our analysis about Theorem 3, where V can be leveraged to ensure robustness of OLCD to workload prediction errors.

6 Related Work

Cooperative MEC. The emerging MEC paradigm (a.k.a. fog computing [1], cloudlet [43]) offers the possibility for supporting mobile applications such as augmented reality and online gaming [3], [4], [5]. Compared with traditional data

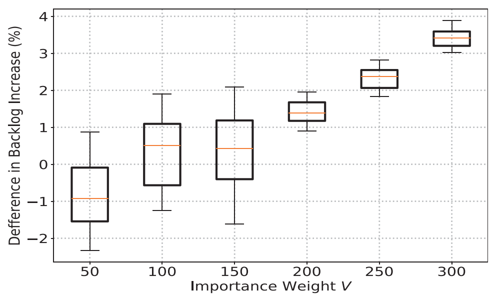


Fig. 15. Differences in backlog increase versus V.

centers, however, edge clouds are much more limited in resources, making it generally impossible for them to process all tasks alone, especially for computation-intensive and latency-sensitive tasks. One traditional method is to allow edge clouds to offload part or all workloads to the remote cloud, but it may struggle to meet user demand for low latency and relying on a single edge cloud significantly limits MEC performance. Instead, Xiao et al. [13] proposed a novel cooperation strategy, where edge clouds can help each other jointly offload workloads. Such cooperative offloading is the main focus of this paper. Chen et al. [8] studied peer offloading for energy-constrained MEC in the cases of centralized and autonomous coordination. However, these works assume tasks can be offloaded only once. Clearly, a multi-hop approach is more promising in exploiting collaborative computing capabilities [14]. Lyu et al. [15] presented a distributed optimization for cost-effectiveness multi-hop offloading decisions. The major advantage of our work is to capture both latency variability and trust risks in collaboration, producing new challenges for multi-hop offloading design.

Trust Risks in Collaboration. Edge clouds are often deployed by self-interested individuals, which may cause crisis of trust or even security risks. Chen et al. [8] studied how to avoid edge clouds' refusal to participate under limited resources. Given high correlation between user mobility and service migration, He et al. [17] considered user location privacy committed by untrusted MEC provider. To address this, building trust-based interaction model to identify trustworthy edge clouds is necessary. Chen et al. [18] proposed static trust network to manage trust risks in MEC collaboration. Lin et al. [19] studied dynamic reputation-based node selection in fog-assisted online gaming. There also has been extensive research dedicated to monetary incentive mechanisms for cooperative offloading [44], [45]. The focus of our work is not to design yet another monetary incentive mechanism, but rather, to develop a multi-hop offloading mechanism built upon accumulated social trust relationships, which essentially provides edge clouds with non-monetary incentive. In general, trusted edge clouds are more likely to execute tasks with high QoS. Actually, our non-monetary incentive method can work in conjunction with existing monetary ones. It would be an interesting future work to integrate these two methods into hybrid incentive [46] and study the reward sharing issue aligned with fairness criterion, where edge clouds are rewarded by their contributions to service offerings.

Latency Variability. Many services that rely on distributed computing resources usually present high variability in latency. In cloud computing, Zhu et al. [20] characterized the latency variability problem, and Qiu et al. [21] exploited latency variability to reduce latency. However, prior works on latency minimization in MEC typically assume that completion latency can be determined in advance [8], [9], [16], [18] or tasks are executed independently of each other [10], which fails to capture resource contention in computation offloading. Li et al. [4] considered a joint assignment optimization to maximize the proportion of offloading tasks that can meet deadlines. To break this barrier, we resort to the delayed online learning technique to facilitate task completion latency estimation, which is fed as input to online offloading policy.

7 Conclusion

In this paper, we study the online trusted cooperative offloading problem in MEC system, taking trust risks in collaboration and completion latency variability into explicit consideration. We develop a novel online learning-aided offloading mechanism for trusted collaborative MEC. To accommodate latency variability, we harness the delayed online learning technique to predict latency knowledge, which serves as the basis for multi-hop offloading decision making. Both theoretical analysis and trace-driven simulations validate the effectiveness of our mechanism.

Acknowledgments

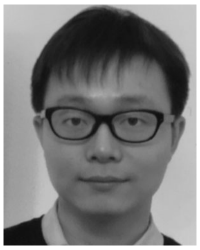
This work was supported in part by the National Key R&D Program of China 2017YFB1003000, 2018YFB1004705, 2018YFB2100302, and NSFC China (No. 61672342, 61671478, 61602303, 61532012, 61822206, 61829201), in part by the Science and Technology Innovation Program of Shanghai (Grant 17511105103, 18510761200), in part by the Open Research Fund of National Mobile Communications Research Laboratory, Southeast University Grant 2018D06, and in part by the Shanghai Key Laboratory of Scalable Computing and Systems.

References

[1] M. Chiang and T. Zhang, “Fog and IoT: An overview of research opportunities,” *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.
[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[3] Y. Sun, S. Zhou, and J. Xu, “EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks,” *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.
[4] T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, and H. Wang, “On efficient offloading control in cloud radio access network with mobile edge computing,” in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Jun. 2017, pp. 2258–2263.
[5] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, “Online resource allocation for arbitrary user mobility in distributed edge clouds,” in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Jun. 2017, pp. 1281–1290.
[6] R. Kelly, “Internet of Things data to top 1.6 zettabytes by 2020,” Apr. 2015. [Online]. Available: https://campustechnology.com/articles/2015/04/15/internet-of-things-data-to-top-1-6-zettabytes-by-2020.aspx
[7] T. He, H. Khamfroush, S. Wang, T. L. Porta, and S. Stein, “It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources,” in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, Jul. 2018, pp. 365–375.

[8] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[9] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[10] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 207–215.
[11] A.-C. Pang, W.-H. Chung, T.-C. Chiu, and J. Zhang, "Latency-driven cooperative task computing in multi-user fog-radio access networks," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Jun. 2017, pp. 615–624.
[12] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.
[13] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2017, pp. 1–9.
[14] M. Wang, H. Jin, C. Zhao, and D. Liang, "Delay optimization of computation offloading in multi-hop ad hoc networks," in *Proc. IEEE Int. Conf. Commun. Workshops*, May 2017, pp. 314–319.
[15] X. Lyu, C. Ren, W. Ni, H. Tian, and R. P. Liu, "Distributed optimization of collaborative regions in large-scale inhomogeneous fog computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 574–586, Mar. 2018.
[16] L. Pu, X. Chen, J. Xu, and X. Fu, "D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3887–3901, Dec. 2016.
[17] T. He, E. N. Ciftcioglu, S. Wang, and K. S. Chan, "Location privacy in mobile edge clouds: A chaff-based approach," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2625–2636, Nov. 2017.
[18] L. Chen and J. Xu, "Socially trusted collaborative edge computing in ultra dense networks," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, Oct. 2017, Art. no. 9.
[19] Y. Lin and H. Shen, "CloudFog: Leveraging fog to extend cloud gaming for thin-client MMOG with high quality of service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 431–445, Feb. 2017.
[20] J. Zhu, Z. Zheng, and M. R. Lyu, "DR2: Dynamic request routing for tolerating latency variability in online cloud applications," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, Jun. 2013, pp. 589–596.
[21] Z. Qiu, J. F. Perez, and P. G. Harrison, "Variability-aware request replication for latency curtailment," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.
[22] Z. Zheng and N. B. Shroff, "Online multi-resource allocation for deadline sensitive jobs with partial values in the cloud," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.
[23] X. Li, H. Ma, W. Yao, and X. Gui, "Data-driven and feedback-enhanced trust computing pattern for large-scale multi-cloud collaborative services," *IEEE Trans. Serv. Comput.*, vol. 11, no. 4, pp. 59–62, Jul. 2018.
[24] Y. Yao, L. Huang, A. B. Sharma, L. Golubchik, and M. J. Neely, "Power cost reduction in distributed data centers: A two-time-scale approach for delay tolerant workloads," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 200–211, Jan. 2014.
[25] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, Aug. 2016.
[26] H. Wang, R. Shea, X. Ma, F. Wang, and J. Liu, "On design and performance of cloud-based distributed interactive applications," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 37–46.
[27] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *IEEE Access*, vol. 36, no. 1, pp. 59–62, Jan. 2006.
[28] C. -H. Tai, J. Zhu, and N. Dukkipati, "Making large scale deployment of RCP practical for real networks," in *Proc. IEEE 27th Conf. Comput. Commun.*, Apr. 2008, pp. 2180–2188.
[29] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. San Mateo, CA, USA: Morgan & Claypool, 2010.
[30] Y. Li, W. Dai, J. Bai, X. Gan, J. Wang, and X. Wang, "An intelligence-driven security-aware defense mechanism for advanced persistent threats," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 646–661, Mar. 2019.
[31] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107–194, Feb. 2012.
[32] X. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2017, pp. 1–9.
[33] K. Quanrud and D. Khashabi, "Online learning with adversarial delays," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, Dec. 2015, pp. 1270–1278.
[34] P. Joulani, A. GyMörgy, and C. Szepesvári, "Online learning under delayed feedback," in *Proc. 30th Int. Conf. Mach. Learn.*, Jun. 2013, pp. III-1453–III-1461.
[35] N. Chen, A. Agarwal, A. Wierman, S. Barman, and L. L. H. Andrew, "Online convex optimization using predictions," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2015, pp. 191–204.
[36] L. Kazemi and C. Shahabi, "GeoCrowd: Enabling query answering with spatial crowdsourcing," in *Proc. 20th Int. Conf. Advances Geographic Inf. Syst.*, Nov. 2012, pp. 189–198.
[37] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, pp. 248–264, 1972.
[38] X. Li, H. Ma, W. Yao, and X. Gui, "A trust-based framework for fault-tolerant data aggregation in wireless multimedia sensor networks," *IEEE Trans. Depend. Sec. Comput.*, vol. 9, no. 6, pp. 785–797, Nov. 2012.
[39] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.
[40] Institute of Real Estate Management, "Trends in office buildings operations," 2012. [Online]. Available: https://www.irem.org/File%20Library/IREM%20Store/Document%20Library/IESamples/12Samples/2012OfficeBuildTrends.pdf.
[41] [Online]. Available: https://code.google.com/p/googleclusterdata/, Accessed: 2019.
[42] M. E. J. Newman, "Power laws, pareto distributions and zipf's law," Contemporary Physics, vol. 46, no. 5, pp. 323–351, Sept. 2005.
[43] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, Oct. 2016.
[44] D. Zhao, X.-Y. Li, and H. Ma, "How to crowdsource tasks truthfully without sacrificing utility: Online incentive mechanisms with budget constraint," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 1213–1221.
[45] M. Zeng, Y. Li, K. Zhang, M. Waqas, and D. Jin, "Incentive mechanism design for computation offloading in heterogeneous fog computing: A contract-based approach," in *Proc. IEEE Int. Conf. Commun.*, May 2018, pp. 1–6.
[46] Y. Wang, X. Jia, Q. Jin, and J. Ma, "QuaCentive: A quality-aware incentive mechanism in Mobile Crowdsourced Sensing (MCS)," *The J. Supercomput.*, vol. 72, no. 8, pp. 2924–2941, Aug. 2016.

Yuqing Li received the BS degree in communication engineering from Xidian University, Xi'an, China, in 2014, and is currently working toward the PhD degree in electronic engineering at Shanghai Jiao Tong University, Shanghai, China. Her current research interests include edge/mobile computing, social aware networks, privacy/security, and network economics.

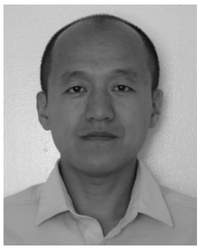


Xiong Wang received the BE degree in electronic information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2014, and is currently working toward the PhD degree in electronic engineering at Shanghai Jiao Tong University, Shanghai, China. His current research interests include crowdsourcing, data mining, resource allocation, and mobile computing.

Xiaoying Gan received the PhD degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, in 2006. She is currently an associate professor in the Department of Electronic Engineering, Shanghai Jiao Tong University. From 2009 to 2010, she worked as a visiting researcher at the California Institute for Telecommunications and Information Technology, University of California San Diego. Her research interests include network economics and wireless resource management. She is a member of the IEEE.

Haiming Jin received the BS degree from Shanghai Jiao Tong University, Shanghai, China, in 2012, and the PhD degree from the University of Illinois at UrbanaChampaign (UIUC), Urbana, IL, in 2017. He is currently a tenure-track assistant professor with the John Hopcroft Center for Computer Science and the Department of Electronic Engineering, Shanghai Jiao Tong University. Before this, he was a post-doctoral research associate with the Coordinated Science Laboratory, UIUC. His research interests include crowd and social sensing systems, reinforcement learning, and mobile pervasive and ubiquitous computing.

Luoyi Fu received the BE degree in electronic engineering from Shanghai Jiao Tong University, China, in 2009 and the PhD degree in computer science and engineering from the same university, in 2015. She is currently an assistant professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. Her research of interests are in the area of social networking and big data, connectivity analysis, and random graphs.



Xinbing Wang (SM'12) received the BS degree in automation from Shanghai Jiao Tong University, Shanghai, China, in 1998, the MS degree in computer science and technology from Tsinghua University, Beijing, China, in 2001, and the PhD degree with a major in electrical and computer engineering and minor in mathematics from North Carolina State University, Raleigh, in 2006. Currently, he is a professor with the Department of Electronic Engineering, Shanghai Jiao Tong University. His research interests include resource allocation and management in mobile and wireless networks, cross-layer call admission control, and congestion control over wireless ad hoc and sensor networks. He has been a member of the technical program committees of several conferences including ACM MobiCom 2012, ACM MobiHoc 2012, and IEEE INFOCOM 2009-2013. He is a senior member of the IEEE.