An Integrated Top-down and Bottom-up Task Allocation Approach in Social Sensing based Edge Computing Systems

Daniel (Yue) Zhang, Dong Wang
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
Email: yzhang40@nd.edu, dwang5@nd.edu

Abstract—**With the advance of mobile computing, Internet of Things, and 5G networks, social sensing based edge computing (SSEC) systems have emerged as a new computation paradigm where people and their personally owned devices collect and process sensing measurements about the physical world at the edge of networks. In this paper, we focus on the task allocation problem in SSEC where rational edge devices are motivated by incentives to collectively accomplish the computation tasks in the system. Several unique challenges exist to solve this problem: (i) the edge devices often do not share the complete context information (e.g., CPU, memory usage) in the task allocation process due to privacy concerns; (ii) the edge devices are rational actors who may have competing objectives with the application; (iii) the application server and edge devices are usually owned by different entities, making the coordination in task allocation more challenging. This paper develops a novel integrated Top-Down and Bottom-Up (TDBU) task allocation framework to address these challenges. In particular, TDBU incorporates a *bottom-up* game-theoretic model that allows the edge devices to specify their task preferences in a way that maximizes their payoffs. It also incorporates a *top-down* control model that ensures the performance of the applications using control theory. The TDBU was implemented on a real-world edge computing testbed that consists of heterogeneous devices (Jetson TX1, TK1 boards, Raspberry Pi3). We compared the performance of TDBU with state-of-the-art baselines through a real-world social sensing application. The results showed that our solution significantly outperformed the baselines in various application settings.**

I. Introduction

In recent years, *social sensing* has emerged as a new sensing paradigm for collecting real-time measurements about the physical world using humans or the privately owned devices on their behalf [1]. Examples of social sensing applications include obtaining real-time situation awareness in the aftermath of a disaster using observations from online social media [2], monitoring the air quality of a city using inputs from people with portable sensors [3], and detecting real-time traffic abnormalities using mobile phone apps of drivers [4].

The emergence of edge computing pushes the frontier of computation, service, and data from the cloud to the edge of the networks [5], [6] and brings new opportunities for social sensing applications. By adopting edge computing, social sensing applications can offload the computation tasks to the edge devices (e.g., smart phones and nodes in the Internet of Things) as well as local edge servers (e.g., micro data centers and smart gateways) [7]. The benefits of running social sensing applications in the edge computing systems (referred to as Social Sensing based Edge Computing Systems or SSEC) are multi-fold: (i) it saves the overall network bandwidth for data transmission and reduces the risks of overloading the backend servers; (ii) it enables immediate responses to time-sensitive applications, allowing services to be delivered to the end users more efficiently; (iii) the massive computation power at edge devices is better utilized and the end users can also obtain rewards by executing the computation tasks on their devices.

A critical problem in the SSEC is *task allocation* where rational edge devices are motivated by incentives to collectively accomplish the computation tasks in the system. Existing task allocation schemes in edge computing can be classified into two major categories: top-down (centralized) approach and bottom-up (decentralized) approach. In a top-down based approach, a centralized decision maker (e.g., an edge server) is assumed to have full control and information of the tasks and the computation resources in the system [8]. It assigns tasks to edge devices based on their run-time status (e.g., CPU and memory usage) to meet the performance requirements of the application. The key limitation of the top-down approach is that it makes strong assumptions on the global knowledge of the system which is often not practical in SSEC systems [9]. In a bottom-up based approach, the privately owned edge devices make their own task allocation decisions in a distributed manner without the need for a centralized decision maker [10]. Such an approach gives control to the end users to maximize their payoffs. However, it is challenging for a bottom-up approach to maintain the desirable system performance due to the myopic view of the system information from individual edge devices and their rational nature [11]. Consequently, we found neither top-down nor bottom-up approaches can effectively address the task allocation problem in SSEC due to several unique challenges as discussed below.

Asymmetric Information Challenge: This challenge refers to the fact that neither the edge device nor the application has complete information (e.g., hardware specs, energy profile, resource utilization) about the entire SSEC system. For example, the application often does not have detailed information about the edge devices due to the privacy concerns of end users and the excessive overhead of synchronizing devices' status to the remote server. In contrast, the edge devices are fully aware of its own device status but they do not have the information of resources that are owned by the application (i.e., local edge servers and remote data centers). Due to the lack of holistic information of the SSEC system, it is difficult to enforce optimal task allocation decisions by either the application (top-down) or the edge devices (bottom-up).

Competing Objective Challenge: A SSEC application and the edge devices may have inconsistent and even competing objectives. From the application's perspective, it is important to ensure that the edge devices finish the allocated tasks in a timely fashion to meet the Quality of Service (QoS) requirements. In contrast, privately owned edge devices are assumed to be *rational actors* who are often less concerned about the QoS of the application but more concerned about their own costs (e.g., device's utilization, energy, memory usage) and payoff. Due to such competing objectives, task allocation performed on either side (application or edge device) could inevitably be suboptimal to the other.

Disparate Ownership Challenge: In SSEC, the computation and communication resources are often owned by different entities. For example, the edge devices are often owned by their end users while the underlying infrastructure (e.g., local edge servers, access points, and remote data centers) is often owned by the SSEC application. The challenge for top-down task allocation is that it cannot assume the privately owned edge devices are always available and fully cooperative in the task allocation process. In fact, the compliance issue from end users is a key concern in social sensing applications [11]. Similarly, the challenge for bottom-up task allocation is that it is inappropriate to assume the edge devices can directly control the infrastructure owned by the application. A task allocation scheme that explicitly considers the disparate ownership in SSEC has yet to be developed.

To address the above challenges, this paper develops a novel integrated Top-Down Bottom-Up (TDBU) approach to solve the task allocation problem in SSEC. In particular, to meet the objective of the edge devices, we develop a bottom-up game theoretic approach in TDBU that allows edge devices to specify their preferred tasks in a way that maximizes their payoffs. To meet the objective of the application, we develop a top-down control mechanism in TDBU using stochastic control theory and online learning to regulate the task allocation process that ensures the QoS performance of the application. The TDBU scheme is novel because it allows the edge devices and the application to "meet in the middle" and reconcile their competing objectives using a holistic solution. In particular, the top down module i) observes the bottom-up task preference decisions from edge devices and decides the optimal task offloading strategy to ensure the overall system performance; and ii) leverages top-down incentive schemes to implicitly guide the edge devices to pick the tasks that they are most likely to finish in time. We implemented a system prototype of TDBU on a real-world edge computing platform that consists of Jetson TX1, TK1 boards, and Raspberry Pi3. The TDBU was evaluated using a real-world social sensing application: *Real-time Traffic Monitoring* [12]. We compared TDBU with the state-of-the-art task allocation schemes used in edge computing systems. The results show that our scheme achieves a significant performance gain in terms of meeting the objectives of both applications and edge devices.

II. Related Work

A. Social Sensing and Edge Computing

Social sensing has received a significant amount of attention due to the proliferation of low-cost mobile sensors and the ubiquitous Internet connectivity [13]. A large set of social sensing applications are sensitive to delay, i.e., have real-time requirements. Examples of such applications include intelligent transportation systems [4], environmental sensing [3], and disaster and emergency response [14]. Traditional social sensing applications push all the computation tasks to the remote servers/cloud, which can be quite ineffective, particularly for delay-sensitive applications, when the network bandwidth is limited and the communication latency is high [9], [15]. Edge computing systems complement traditional centralized social sensing solutions by offloading computation tasks to the edge devices to significantly reduce the communication costs and application latency [16]. A comprehensive survey of edge computing is given by Shi *et al.* [6].

B. Top-down (Centralized) Computation Allocation

The task allocation is an important problem in distributed and real-time systems and many top-down solutions have been developed to address this problem [8], [9], [17]. In these solutions, a centralized decision maker (e.g., back-end server) makes global task allocation decisions. For example, Davare *et al.* proposed a Mixed Integer Linear Programming based approach to meet the deadlines and minimize the end-to-end latency in hard real-time systems [8]. Wang *et. al* considered the assignment of mobile computational tasks over multiple cloudlets to optimize the overall computation efficiency [18]. Zhang *et. al* developed a heterogeneity-aware task allocation scheme that can map interdependent tasks to be assigned to devices with heterogeneous hardware and runtime environment that jointly minimizes task delay and energy consumption [19]. Most of the top-down schemes assume that edge devices are fully controlled and cooperative [7], [18]. Such assumption does not hold in the SSEC systems where the end users might refrain from providing necessary information to accomplish the tasks allocated by the top-down approaches [20].

C. Bottom-up (Decentralized) Computation Allocation

Considering the limitations of top-down task allocation schemes, bottom-up approaches have been developed to allow edge devices to select tasks in a distributed manner. For example, Liu *et al.* proposed a data offloading scheme using multi-item auction and congestion game approaches to decide the optimal strategy for edge devices to offload tasks to the cloud [21]. Chen *et al.* proposed a game-theoretic approach to achieve an efficient and decentralized computation offloading [22]. However, these approaches assume the edge devices are *cooperative* and cannot be applied to our SSEC system where edge devices are rational actors. The most relevant work was proposed by Zhang *et al.*, in which a bottom-up task allocation scheme (referred to as BGTA) was developed to allow non-cooperative edge devices to pick their preferred tasks using dynamic incentives [11]. However, this approach cannot address the unique challenges of asymmetric information and disparate ownership of SSEC discussed in the introduction.

III. Problem Formulation

A. System Model

A typical social sensing based edge computing (SSEC) system architecture is shown in Figure 1. In SSEC, a set of X privately owned edge devices $ED = \{E_1, E_2, ..., E_X\}$ perform the sensing and computation tasks near the data sources (i.e., social sensors). The application server AS (often built into a remote data center/cloud) provides a global service interface to all users of interest to the social sensing applications. A set of Y local edge servers $ES = \{ES_1, ES_2, ..., ES_Y\}$ (e.g., micro data centers, cloudlets, smart routers, or gateways) provide additional data storage and computation power in locations of close proximity to the edge devices. In SSEC, edge devices are commonly connected to the remote social sensing applications through these local edge servers that coordinate the task allocation of all edge devices that communicate with it [7]. Such a design provides local data processing capability to reduce the end-to-end latency and offer a generic communication interface between heterogeneous edge devices and the remote applications [6].

We assume both AS and ES are owned by the application manager who provides incentives to edge devices ED to reward their contributions in accomplishing the computation tasks.

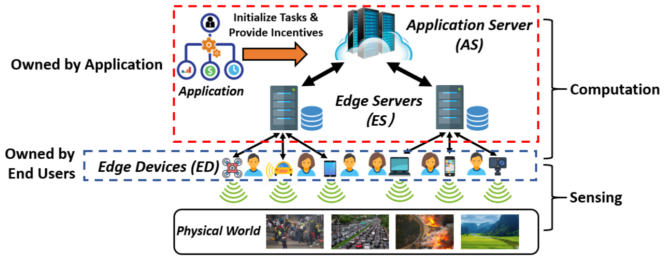


Figure 1. Social Sensing based Edge Computing System

B. Task Model

We adopt a frame-based task model [23] commonly used in the real-time system community where tasks are periodically initialized and have the same period and deadline. We use Ω to denote the common deadline of all the tasks in an application. Ω captures the user desired QoS of the application (i.e., when the tasks should be finished).

A social sensing application is assumed to have a set of Z tasks, $Task = \{\tau_1, \tau_2, ...\tau_Z\}$, which are initialized by the application at the beginning of each sensing cycle. A computation task for edge devices converts the raw sensor measurements into desired intermediate outputs. The computation results of the task are sent back to the edge server for further data processing and analysis. For example, consider a typical SSEC application where drivers' dash cameras are leveraged to monitor the road condition. The edge devices can perform data prepossessing by extracting visual and motion features from the raw camera data. Then the output features are sent to the local edge server for further analysis (e.g., object detection). Each task is associated with a 3-tuple: $\tau_z = \{VI_z, VO_z, c_{z,x}\}$ where VI_z is the data volume to be processed by task τ_z and VO_z is the size of the output. $c_{z,x}$ is the estimated worst-case execution time (WCET) if τ_z is assigned to the edge device E_x.

C. Cost and Incentive Model

In order to motivate edge devices to perform computation tasks, it is often necessary for SSEC to provide incentives (either monetary or non-monetary) to end users to reward their participation and compensate for their costs (e.g., energy) [11], [24]. We first define a few terms in our model.

DEFINITION 1. Edge Cost (π_x): it refers to the energy cost that edge device ED_x have to pay to finish the computation tasks.

From the end users' perspective, energy consumption is a major concern considering the fact that most of the edge devices are running on battery. In this paper, we adopt a relatively simple energy model that is sufficient for demonstrating how TDBU can take energy into consideration during the task allocation process[1]. Given the WCET $c_{z,x}$, the energy consumed by executing τ_z on edge device E_x is computed as

$$e_{z,x} = Power_x \times c_{z,x}, \tag{1}$$

where $Power_x$ is the average power consumption of edge device E_x and is calculated by

$$Power_x = Power_{comp,x} + Power_{trans,x} \tag{2}$$

where $Power_{comp}$ is the power consumption for computation and $Power_{trans}$ is the power for data transmission and is proportional to the transferred data size.

DEFINITION 2. Task Reward r_z: it refers to the incentives that the application provides for compensating the edge device for completing each social sensing task. We assume the server has a fixed budget (denoted as η_C) for the social sensing application.

DEFINITION 3. Edge Payoff $u_{z,x}$: it refers to the overall benefit the edge device E_x receives by executing a task τ_z. It defines the individual gain as a function of both cost and

[1]The TDBU framework can be readily extended to more complicated energy models, e.g., supporting multiple voltage/frequency levels. The details are omitted due to page limit.

reward of executing tasks on an edge device. More details on the edge payoff are given in Section IV.

Our model also assumes edge devices are not malicious (e.g., output fake computation results) or lazy (i.e., intentionally postpone execution time). We discuss how to deal with situations where such assumptions do not hold in Section VII.

D. Objectives

Based on the above definitions, assumptions and system models, we formally define the objectives of TDBU as follows.

Server (Application) Objective: From the server's perspective, the objective is to satisfy the predefined QoS requirement, characterized by the End-to-End delay of tasks.

DEFINITION 4. End-to-end delay of task ($\mathcal{L}_z$): the total amount of time taken for a unit of sensor measurement (e.g., a video frame of a road) to be processed (by both the edge device and edge server) and turned into final data analysis result (e.g., inferred traffic congestion index).

Formally, the server's objective is to

$$\text{minimize: } \sum_{z=1}^{Z} \delta_z, \quad \text{s.t. } \sum_{z=1}^{Z} r_z \ \leq \eta_C \tag{3}$$

where r_z is the incentive for task τ_z. δ_z is a binary variable: $\delta_z = 1$ if task τ_z misses the deadline (i.e., $\mathcal{L}_z > \Omega$) and $\delta_z = 0$ otherwise.

Edge Objective: From the perspective of edge devices, the objective is to maximize their own payoffs. The edge devices are often constrained by their physical distances to the task (e.g., many tasks in social sensing are location-aware and is associated with the sensing range of the device). We assume an edge device E_X collects and process sensor data within its sensing range η_{D_x} [3]. More details on setting up the sensing range are discussed in Section VI.

Let $dist_{x,z}$ denote the distance between E_X and task τ_z. S_x represents all the tasks allocated to E_x. We formally define the edge objective as:

$$\begin{aligned} \text{maximize: } & \sum_{z \in S_x} u_{z,x}, \forall 1 \leq x \leq X \\ \text{s.t. } & dist_{x,z} \leq \eta_{D_x}, \forall 1 \leq x \leq X, \ \forall 1 \leq z \leq Z \end{aligned} \tag{4}$$

It is proven that the task allocation problem for heterogeneous distributed systems is in general NP-hard [18], [25]. The problem becomes more challenging in our multi-objective formulation (Equations (3) and (4)) where objectives of the application and edge devices are potentially conflicting and additional constraints on costs and devices are imposed. In the next section, we present the TDBU to address the problem.

IV. Solution

An overview of TDBU is given in Figure 2. The TDBU scheme consists of two major components: i) a game-theoretic Bottom-up Task Preference (BUTP) module that allows edge device to pick their preferred tasks based on their own payoffs, and ii) a Top-Down Optimal Control (TDOC) module that ensures the objectives of both server and edge are satisfied. We elaborate these components below.

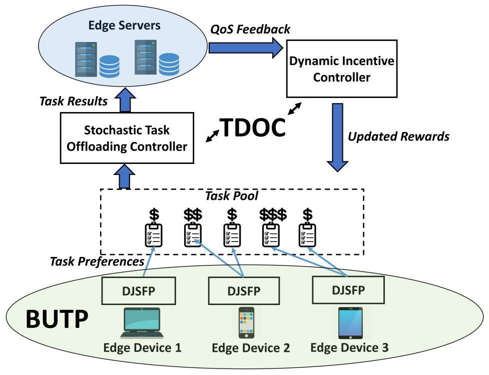


Figure 2. Overview of TDBU

A. Game-Theoretic Bottom-Up Task Preference Module

We first develop a BUTP scheme by addressing the objectives of edge devices via game theory. A key design philosophy of TDBU is to allow edge devices to express their task preferences in the allocation process. This design i) allows an edge device to selfishly identify the strategy that maximizes their own payoffs, and ii) hides the details of device status from the server and effectively preserves the privacy of end users.

In BUTP, edge device competes for the computation tasks by playing a non-cooperative game. The high-level game protocol of the BUTP is as follows:

1) At the beginning of a sensing cycle, the application defines Z tasks and the reward r_z for each task τ_z.
2) Each edge device selfishly picks a task that has the highest payoff for itself. We refer to one round of this task allocation process as an "iteration".
3) Within each iteration, we assume each edge device is myopic and only picks one task at a time (this is referred to as "singleton property" that reduces the strategy space from O(2^Z) to O(Z)) [26].
4) Keep iterating until all edge devices reach a consensus.
5) EAch device sends outputs of the executed task to the edge server to claim rewards.

The above protocol follows the rule of *singleton weighted congestion game* where the Pure Strategy Nash Equilibrium is *guaranteed* to exist [27]. This is crucial for edge devices to make mutually satisfactory task allocation decisions. The convergence property of BUTP is evaluated in Section VI-C.

Given the above game protocol, we formally derive a key element in the BUTP: the payoff function. The payoff function depends on three factors: 1) the *cost* for processing a selected task, 2) the *reward* associated with the completion of a task, and 3) a *penalty function* that the server applies to penalize delayed computation results.

We first define the cost as:

$$\pi_{z,x} = \begin{cases} e_{z,x}, & distx, z \leq \eta_{D_x} \\ \infty, & distx, z > \eta_{D_x} \end{cases} \tag{5}$$

where $e_{z,x}$ is the energy consumed by running task τ_z on E_x as discussed in Section III. $dist_{x,z}$ is the physical distance between E_x and the sensing location of task τ_z. We assume an edge device has a sensing radius of η_{D_x} within which it is capable of collecting sensing data.

Given the cost function above, each edge device tries to find the optimal game strategy that minimizes its own cost. This will result in no tasks being picked for execution (since picking no tasks incurs zero cost at each edge device). However, such task allocation results obviously conflict with the objective of the application. Therefore, the reward is provided to incentivize end users to contribute their resource on edge devices. The reward is assigned based on an *initial reward* r_z and a *reward penalty function* $l_{z,x,y}$ defined as:

$$l_{z,x,y} = \begin{cases} \dfrac{\lambda \times \Omega}{\Omega - (td(x,y) + c_{z,x})}, & td(x,y) + c_{z,x} < \Omega \\ \infty, & td(x,y) + c_{z,x} \geq \Omega \end{cases} \tag{6}$$

where $td(x,y)$ denotes the transmission delay from edge device E_x to edge server ES_y. $c_{z,x}$ is the WCET of τ_z and Ω is the task deadline defined in Section III. λ is a parameter that will be tuned via a dynamic incentive controller (discussed in IV-B). The intuition of this reward penalty function is to penalize a "lazy but greedy" edge device that targets at obtaining high rewards by picking many tasks but fails to finish them. Specifically, the closer the finishing time of a task is to the deadline, the higher the penalty is.

Assume that device E_x picks task τ_z in an iteration and $N(z)$ is the number of devices that pick τ_z. Based on the reward, penalty and cost, we define the *payoff function* $u_{z,x}$ of edge device E_x for finishing task τ_z as:

$$u_{z,x} = \begin{cases} Exp(\dfrac{r_z}{\pi_x \times l_{z,x,y'}}) = \dfrac{r_z \times (\Omega - c_{z,x} - td(x,y'))}{\lambda \times \Omega \times N(z) \times e_{z,x}} \\ 0, \quad \pi_{z,x} = \infty \end{cases} \tag{7}$$

where $Exp(\cdot)$ calculates the expected payoff, which is simply the original payoff divided by the number of device that pick the same task.y' refers to the local edge server that is assigned to each edge device E_X. We assume y' is the local edge server that is closest to E_x as discussed in Section III.

To ensure that each edge device makes its best decision towards its rationally selfish objective, our goal is to find a Nash Equilibrium for BUTP. The Nash Equilibrium exists in a non-cooperative game where each player is assumed to know the equilibrium strategies of all other players and no player has anything to gain by only changing his/her own strategy [28].

A unique challenge in finding the Nash Equilibrium in BUTP is that an edge device does not know the strategies of other devices to make the best response of its own (similar as an auction scenario where each bidder would not share her own valuation of an item beforehand). However, a nice property of BUTP is that each edge device knows all information in the payoff function (Equation (7)) except the number of devices that pick the task (i.e., $N(z)$ in Equation (7)). Therefore, to derive the best response, each edge device only needs to estimate $N(z)$, which represents an aggregated preference of all edge device toward a task. We develop a Distributed Joint Strategy Fictitious Play (DJSFP) algorithm to achieve this goal (see Algorithm 1).

Algorithm 1 DJSFP Algorithm For Payoff Maximization

```
1: Initialize: $S \leftarrow newArray[X]$, $EH \leftarrow newArray[Z][X]$, $sig \leftarrow False$, $converge \leftarrow False$
2: while $converge \neq True$ or $iterCount \leq maxIter$ do
3:     $iterCount \leftarrow iterCount + 1$
4:     initialize $N(z) = 1, \forall 1 \leq z \leq Z$
5:     find $br_x \leftarrow$ strategy for $E_x$ based on maximizing Equation (7)
6:     send $br_x$ to local edge server, receive $N(z)$, $\forall 1 \leq z \leq Z$ of all tasks and convergence signal $sig$
7:     $converge \leftarrow sig$
8:     update $EH_{z,x} \leftarrow \mu EH_{z,x} + (1-\mu)N(z)$, $S[x] \leftarrow br_x$
9: end while
10: Return $S[x]$
```

In Algorithm 1, $EH_{z,x}$ is the empirical histogram that each edge device E_x used to predict $N(z)$ in the next sensing cycle. $\mu \in (0,1]$ is a decay factor that controls the importance of more recent observations of $N(z)$.

BUTP alone will not guarantee QoS objective of the application since its objective is to allow edge devices can find the task set that selfishly maximizes their own payoffs. In the next subsection, we show how top-down control is applied to bridge the competing objectives of the edge and the server.

B. Top-Down Optimal Control Module

In this subsection, we design a Top-Down Optimal Control (TDOC) module to complement the BUTP module to meet the QoS objective of the application. The TDOC module consists of two control mechanisms: 1) a Lyapunov optimization based stochastic task offloading controller to maximize the QoS while explicitly considering the payoffs of edge devices; 2) an online-learning based dynamic incentive controller to adjust the payoff of each edge device to address the system dynamics (e.g., deadline miss rate, device status) using feedback control.

The reason for the top-down control is that the server cannot count on the selfish edge device to perform optimal task allocation to meet the QoS requirements of the application. In particular, the E2E delay is composed of both the delay on the edge as well as the edge servers. Therefore, the server has to ensure the task results are offloaded to the right edge server so QoS objective is met. This will be addressed by the stochastic task offloading controller. On the other hand, the payoff function of the edge devices may not align well with the server's objective. This will be addressed by the dynamic incentive controller component.

1) Stochastic Task Offloading Controller via Lyapunov Optimization: Once the edge devices make their task allocation decisions and execute the selected tasks, the results of the tasks will be sent to the local edge servers in a real-time manner. From the perspective of the local edge server, the arrival time of the computation results from edge devices can be highly dynamic and unpredictable, thus treated as "random events". The server has to make dynamic decisions on where the task result should be offloaded to. A naïve solution is to simply allow edge devices to offload task results to the nearest local edge server (i.e., y' defined in Equation (7)). However, this solution is not optimal because the local edge server can be overloaded if the volume of offloading data from edge devices is large. This will cause the excessive E2E delay of tasks.

On the other hand, this top-down task offloading control can cause edge devices to suffer some loss. For example, if the server asks an edge device to offload to an edge server that is far away, it will not only increase the reward penalty, but also induces higher energy cost of the edge device. In TDOC, we design a Stochastic Task Offloading Controller by leveraging Lyapunov optimization technique to manage the task offloading on the edge servers. This technique not only guarantees QoS, but also minimizes the loss on the edge.

We first formally define the *edge loss* as the deficit of the payoff after the control action being applied as compared to the expected payoff before the control action. We assume SSEC is a time-slotted system and the control decisions can be applied in each time slot $t, 1 \le t \le T$. We use $p_{edge}(t)$ to denote the edge loss at time slot t

We further define an *edge server delay* $d_{z,y}(t)$ that measures the accumulated delay if incoming execution result of task τ_z at time slot t is assigned to the edge server ES_y. Formally, $d_{z,y}(t) = DAT_z + c_{z,y}$ where $c_{z,y}$ is the worst-case execution time if the offloaded computation result of τ_z is processed at ES_y, and DAT_z is the data arrival time that captures the total amount of time it takes from a task τ_z being generated till the edge server receives the data.

The goal is to take control actions (i.e., deciding task offloading destination and admission control) such that time-average edge loss $p_{edge}(t)$ is minimized while the QoS objective of the application is satisfied. Formally, we have:

$$\begin{aligned} &Minimize: \quad \lim_{t\to\infty} \frac{1}{T}\sum_{t=1}^{T} E(p_{edge}(t));\\ &Subject\ to: \quad \lim_{t\to\infty} \frac{1}{T}\sum_{t=1}^{T} E(d_{z,y}(t)) \le \Omega; \end{aligned} \tag{8}$$

This objective can be achieved through the Lyaponuv optimization [29], which is a principled framework to control a dynamic and unpredictable system to a stable status. In particular, we define a Lyapunov function $L(t)$ to measure the deficiency of the system towards the QoS objective:

$$L(t) = \frac{1}{2}\sum_{y=1}^{Y} Q_y(t)^2 \tag{9}$$

where $Q_y(t)$ is a delay backlog function (often called "virtual queue" in control theory) that can be represented as:

$$Q_y(t+1) = max[Q_y(t) + d_{z,y}(t) - \Omega, 0], Q_y(0) = 0 \tag{10}$$

It is easy to show that the constraint in Equation (8) can be converted to to $\lim_{t\to\infty} \frac{1}{T}\sum_{t=1}^{T} E(Q_y(t)) = 0, \forall 1 \le y \le Y$. This requires controlling the virtual queues to be *mean rate stable* [30]. To keep the queue stable while minimizing edge loss, we define and minimize a *drift-plus-penalty expression*:

$$V \times p_{edge}(t) + \Delta L(t) \tag{11}$$

where V is a scalar that can be tuned to place more emphasis on edge loss minimization or QoS requirement. $\Delta L(t)$ is a *one-shot Lyapunov drift function*: $\Delta L(t) = L(t+1) - L(t)$.

Lemma. *The drift-plus-penalty expression in Equation* (11) *is upper bounded by* $UB = Vp_{edge}(t) + \sum_{y=1}^{Y} Q_y(t) \times d_{z,y}(t) + C$*, where C is a constant.*

Proof: from Equation (10), the following equality holds

$$\begin{aligned} &Q_y(t+1) \le Q_y(t) + d_{z,y}(t)\\ \Longrightarrow &\frac{1}{2}Q_y(t+1)^2 \le Q_y(t)^2 + d_{z,y}(t)^2 + 2Q_y(t) \times d_{z,y}(t)\\ \Longrightarrow &\Delta L(t) = \frac{1}{2}\sum_{y=1}^{Y} Q_y(t+1)^2 - \frac{1}{2}\sum_{y=1}^{Y} Q_y(t)^2\\ &\le \frac{1}{2}\sum_{y=1}^{Y} d_{z,y}(t)^2 + \sum_{y=1}^{Y} Q_y(t) \times d_{z,y}(t) \end{aligned}$$

Let $C = max\{\frac{1}{2}\sum_{y=1}^{Y} d_{z,y}(t)^2\} = \frac{1}{2Y}\Omega^2$ (we assume tasks are preempted upon missing deadlines). The lemma is proved.

To satisfy our objective in (8), our controller observes the current delay backlog $Q_y(t)$ and incoming task results at every time slot t, and then picks a control decision to greedily minimize the upper-bound UB. The controller has the following control decisions:

- Admission Control ($A^0(t)$) - decide if a task's results to be admitted.
- Server Offloading Control ($A^1(t)$) - decide which edge server for task offloading.

Let $A^0_{x,y,z}(t)$ denote the control strategy where the task τ_z from edge device E_x is assigned to ES_y at time slot t, and $A^1_{x,z}(t)$ denote that the task τ_z from E_x has not been admitted for processing.

The edge loss $\Delta p_{edge}(t)$, given a control decision, becomes:

$$p_{edge}(t) = \begin{cases} u^{new}_{z,x} - u^{old}_{z,x}, \text{ if apply } A^0_{x,y,z}(t)\\ u^{old}_{z,x}, \text{ if apply } A^1_{x,z}(t) \end{cases} \tag{12}$$

where $u^{old}_{z,x}$ and $u^{new}_{z,x}$ denote the payoff expected by edge device E_x and the new payoff if the task results been sent to ES_y. Note that the edge servers alone cannot derive $p_{edge}(t)$ due to asymmetric information. Each edge device is required to send to the edge server necessary information (i.e., $e_{z,x}, u^{old}_{z,x}$) to compute $p_{edge}(t)$ after finishing executing tasks.

The delay backlog $Q_y(t)$ becomes:

$$Q_y(t) = \begin{cases} Q_y(t) + c_{z,y}, \text{ if apply } A^0_{x,y,z}(t)\\ Q_y(t), \text{ if apply } A^1_{x,z}(t) \end{cases} \tag{13}$$

At each time slot, the application enumerates all control options and find the one that minimizes UB. The complexity of this process is $O(Z)$. We refer to more details in [30].

2) Dynamic Incentive Controller via Online Learning: The BUTP module ensures that the edge devices can achieve the maximum payoff given the reward assignment of each task. As shown in the payoff function Equation (7), the definition of the reward penalty function plays a critical role with respect to whether a task is picked by an edge device for execution. Static reward penalty functions are shown to be suboptimal for the application to satisfy the QoS requirements [11]. On one hand, if we set the reward penalty too stringent (by setting λ too large in Equation (6)), edge devices will only compete for the tasks with high initial rewards without considering whether they can finish the tasks or not. On the other hand, if we set the reward penalty too leisure (by setting λ too small), no edge devices would compete for the time-consuming tasks.

In TDOC, we develop a simple and effective Dynamic Incentive Controller for the server to dynamically update the reward penalty function in order to meet the QoS objective of the application via an online learning framework. In particular, we set a control loss function ϕ as the average complexity of tasks (characterized by the input size of a task VI_z) that meets the deadlines as compared to the average complexity score of the tasks that missed the deadlines:

$$\phi = \frac{\sum_{z=1}^{N} VI_z \times (1-\delta_z)}{\sum_{z=1}^{Z}(1-\delta_z)} - \frac{\sum_{z=1}^{Z} VI_z \times \delta_z}{\sum_{z=1}^{Z}\delta_z}, \sum_{z=1}^{Z}\delta_z \neq 0 \; or \; Z \tag{14}$$

where δ_z is defined as whether a task τ_z meets the deadline or not (in Section III). Based on the loss function ϕ and a learning parameter η, the reward penalty is updated via a classical online learning Exponential Weights Algorithm (EWA) [31] as:

$$\lambda^{new} = \lambda^{old} \times \mathrm{e}^{-\eta\phi}, \tag{15}$$

V. System Implementation and Experiment Platform

This section presents the hardware setup used in our experiments and our implementation of the TDBU framework.

A. Hardware Platform

We use three PC workstation with Intel E5-2600 V4 processor and 16GB of DDR4 memory as the local edge servers. The edge servers also coordinate edge devices to synchronize their strategies after each iteration of the BUTP algorithm.

The hardware platform consists of 15 edge devices: 2 Jetson TX1 and 3 Jetson TK1 boards from Nvidia, and 10 Raspberry Pi3 Model B boards. The detailed hardware set up is given in [20]. These edge devices have heterogeneous energy profiles and computation capabilities and are commonly used in portable computers, UAVs, and autonomous vehicles. All devices and the edge server are connected via a wireless router. The data communication is achieved using TCP sockets.

B. Energy Profiling

Energy data are needed in our system. We use FLUKE AC/DC current clamps to monitor real-time current signal of each edge device and capture the current values using a National Instruments USB-6216 Data Acquisition (DAQ) system. We then multiply the current values with the default voltage (12 V for TK1, 19 V for TX1 and 5 V for Pi3) to obtain the power consumption by each edge device.

VI. Evaluation

In this section, we study the performance of our TDBU scheme using the SSEC system described in V. We first discuss the baselines for comparison and then present the evaluation results using a real-world social sensing application.

A. Baselines

- **Bottom-Up Game-theoretic Task Allocation (BGTA)**: A pure bottom-up edge computing task allocation scheme that allows edge devices to selfishly compete for tasks to maximize their own payoffs [11].
- **Top-Down Allocation (TDA)**: A top-down task allocation scheme that uses Mixed Integer Linear Programming (MILP) to minimize the deadline miss rate [8]. The MILP requires the server to estimate the status (e.g., background utilization and frequency) of the edge devices. In our experiment, we dynamically adjust the estimation of the background utilization and choose the setting that works best for MILP.
- **Top-Down Allocation with Full Information (Ideal)**: An ideal case of TDA where the server has full context information of all edge devices.
- **Congestion (COG)**: A congestion game based edge computing task allocation scheme where the reward of a task is monotonically decreasing as more edge devices claiming that task [32].
- **Greedy-Max Reward (GMXR)**: A greedy computation allocation scheme where an edge device always picks the tasks with the highest reward.

Note that the "Ideal" baseline assumes the server has full information of the edge device. This is not practical in the SSEC system and unfair to TDBU and other baselines that assumes incomplete information of the edge devices at the server. we treat the performance of "Ideal" as an upper-bound and investigate how close the performance of TDBU and other schemes can go compared to the upper-bound.

B. Case Study: Real-time Traffic Monitoring

We evaluate the performance of TDBU and the baselines through a case study of a real-world social sensing application: *Real-time Traffic Monitoring (RTM)* [12]. In this application, a set of drivers are tasked to use their dash cameras or smartphones to take videos of the traffic in front of their vehicles and extract relevant image features. These features can be then sent to nearby edge servers (e.g., Road-Side-Units) to further calculate the congestion rate of the road. In our experiment, we collected the video data using dash cameras from two vehicles. The data contains a total of 15 videos. We divided the application into 100 sensing cycles and each sensing cycle processes video clips of 6 seconds. To emulate the various data volumes of each raw video data collected, we randomly sample each video clip between 1 to

15 image frames per second. A traffic monitoring task for edge devices performs feature extraction (optical flow and HOG) to the original raw video clip data (one task per video clip). For each task, we assign a reward (normalized to [0, η_C]) based on the data size. The extracted feature is further processed by the edge server to infer the local traffic condition. We examine a total of three "virtual streets" to emulate the sensing area of interest (e.g., frequently congested street or a crossroad in real-world), where each virtual street has an edge server deployed. In the following experiments, we randomly assign tasks and edge devices to the three virtual streets and assume the sensing range η_{D_x} of an edge device is within its assigned street. We emulate extra communication latency (randomly selected between 50-100ms) for devices and servers that are not within the same street. We repeat the experiments 100 times to generate the results discussed below.

1) Quality of Service (Application (Server) Side): In the first set of experiments, we focus on how the objective of the application is achieved. In particular, we evaluate the deadline hit rate (DHR) and end-to-end (E2E) delay of all the compared task allocation schemes. The DHR results are shown in Figure 3. We use all 15 edge devices and gradually increase the deadline constraints. We observe that TDBU has significantly higher DHRs than all the baselines except "Ideal" (i.e., the upper bound of performance), and is the first one that reaches 100% DHR as the deadline increases. We attribute such performance gain to our optimal top-down control module of TDBU that ensures the delay is constrained at the local edge servers. We also observe that TDBU significantly outperforms the BGTA scheme, the state-of-the-art bottom-up baseline. This demonstrates the importance of top-down control that guarantees the QoS of the server.

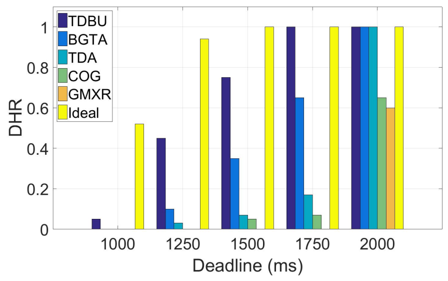


Figure 3. Deadlines vs. DHR

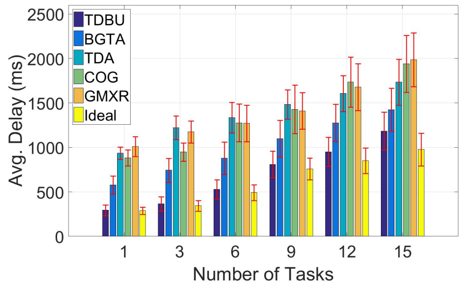


Figure 4. E2E Delay (Ω = 2s)

Figure 4 summarizes the E2E delays of all the schemes as the number of tasks varies. We show the average delays and the 90% confidence bounds. We observe that our TDBU scheme has the least E2E delay and tightest confidence bounds compared to the baselines. The results further demonstrate the effectiveness of TDBU for meeting real-time QoS requirements of the application. The performance gain of the TDBU is achieved by designing top-down policies (i.e., dynamic incentives) to push the edge devices to pick the tasks they can finish quickly to claim more rewards.

2) Payoff (Edge Device Side): In the second set of experiments, we focus on how the objectives of edge devices are achieved. Figure 5 shows the results of the normalized payoffs gained by the edge devices. We observe that TDBU has the highest payoff compared to all the baselines. This is because TDBU allows edge devices to maximize their rewards via achieving the Nash Equilibrium. Compared to other game-theoretic schemes (i.e., COG and BGTA), the TDOC module also allows TDBU to ensure the constrained E2E delay *after* the task results have been sent to the edge servers. This allows the TDBU to achieve higher deadline hit rates than BGTA as shown in Figure 3 and hence reduces the chance of rewards being penalized in the case of a deadline miss. We also note that TDBU also outperforms the "Ideal" baseline in terms of the edge device payoff. The reason is that "Ideal" scheme only focuses on the overall performance of the application without considering the benefits of edge devices.

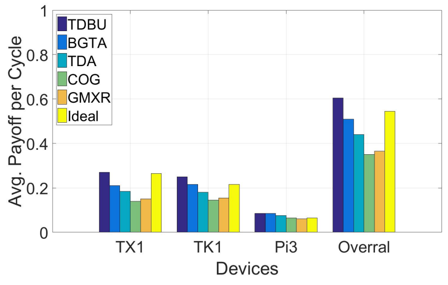
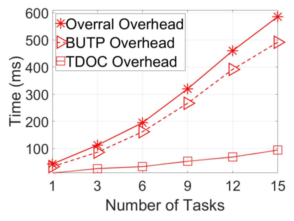


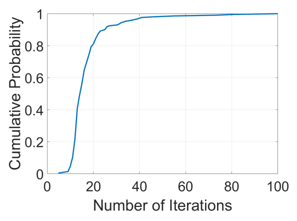
Figure 5. Normalized Payoffs for Edge Devices

C. Allocation Overhead and Convergence

We show the task allocation overhead, in terms of BUTP overhead, TDOC overhead as well as the overall overhead in Figure 6(a). We observe the BUTP scheme is a major contributor to the overhead in the TDBU. This is because the TDOC scheme has a small complexity of O(Z) while BUTP requires multiple iterations for edge devices to compete for their preferred tasks. But we also observe that the BUTP overhead is sub-linear as the number of tasks grows. This is due to the singleton property of the BUTP protocol that significantly reduces the search space for task strategies (discussed in Section IV). We also attribute the efficiency of the TDBU to the quick convergence of the BUTP algorithm (see Figure 6(b)).



(a) Task Allocation Overhead



(b) Convergence of BUTP

Figure 6. Overhead and Convergence Analysis

VII. Conclusion

This paper presents the TDBU framework to addresses three fundamental challenges in solving the task allocation problem for delay-sensitive social sensing applications in edge computing systems, namely *asymmetric information*, *competing objective* and *disparate jurisdiction*. We have implemented TDBU on a real-world edge computing platform that includes Nvidia Jetson TK1, TX1, and Raspberry Pi3 boards. The evaluation results from a real-world social sensing application demonstrate that TDBU achieves significant performance gains in terms of meeting both the objectives of applications and edge devices compared to the state-of-the-art baselines.

Acknowledgment

This research is supported in part by the National Science Foundation under Grant No. CNS-1831669, CBET-1637251, CNS-1566465 and IIS-1447795, Army Research Office under Grant W911NF-17-1-0409, Google 2017 Faculty Research Award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] D. Wang, B. K. Szymanski, T. Abdelzaher, H. Ji, and L. Kaplan, "The age of social sensing," *arXiv preprint arXiv:1801.09116*, 2018.

[2] D. Wang, L. Kaplan, H. Le, and T. Abdelzaher, "On truth discovery in social sensing: A maximum likelihood estimation approach," in *Proc. ACM/IEEE 11th Int Information Processing in Sensor Networks (IPSN) Conf*, Apr. 2012, pp. 233–244.

[3] L. Wang, D. Zhang, A. Pathak, C. Chen, H. Xiong, D. Yang, and Y. Wang, "Ccs-ta: Quality-guaranteed online task allocation in compressive crowdsensing," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2015, pp. 683–694.

[4] Y. Zhang, Y. Lu, D. Y. Zhang, S. Lanyu, and D. Wang, "Risksens: A multi-view learning approach toidentifying risky traffic locations in intelligent transportation systems using social and remote sensing," to appear in Big Data (Big Data), 2018 IEEE International Conference on. IEEE, 2018, accepted.

[5] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*. IEEE, 2016, pp. 1–8.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[8] A. Davare, J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, "Classification, customization, and characterization: Using milp for task allocation and scheduling," *Systems Research*, 2006.

[9] D. Y. Zhang, C. Zheng, D. Wang, D. Thain, X. Mu, G. Madey, and C. Huang, "Towards scalable and dynamic social sensing using a distributed computing framework," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 966–976.

[10] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[11] D. Y. Zhang, Y. Ma, Y. Zhang, S. Lin, X. S. Hu, and D. Wang, "A real-time and non-cooperative task allocation framework for social sensing applications in edge computing systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 316–326.

[12] S. Kamijo, Y. Matsushita, K. Ikeuchi, and M. Sakauchi, "Traffic monitoring and accident detection at intersections," *IEEE transactions on Intelligent transportation systems*, vol. 1, no. 2, pp. 108–118, 2000.

[13] D. Wang, T. Abdelzaher, and L. Kaplan, *Social sensing: building reliable systems on unreliable data*. Morgan Kaufmann, 2015.

[14] D. Wang, T. Abdelzaher, L. Kaplan, and C. C. Aggarwal, "Recursive fact-finding: A streaming approach to truth estimation in crowdsourcing applications," in *The 33rd International Conference on Distributed Computing Systems (ICDCS'13)*, July 2013.

[15] M. T. Al Amin, T. Abdelzaher, D. Wang, and B. Szymanski, "Crowd-sensing with polarized sources," in *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 67–74.

[16] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[17] Y. Zhang, D. Zhang, N. Vance, and D. Wang, "Optimizing online task allocation for multi-attribute social sensing," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2018, pp. 1–9.

[18] L. Wang, L. Jiao, D. Kliazovich, and P. Bouvry, "Reconciling task assignment and scheduling in mobile edge clouds," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–6.

[19] D. Y. Zhang, T. Rashid, X. Li, N. Vance, and D. Wang, "Heteroedge: Taming the heterogeneity of edge computing system in social sensing," to appear in Internet-of-Things Design and Implementation (IoTDI), 2019 ACM/IEEE Third International Conference on. ACM, 2019, accepted.

[20] D. Y. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 243–259.

[21] D. Liu, L. Khoukhi, and A. Hafid, "Decentralized data offloading for mobile cloud computing based on game theory," in *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*. IEEE, 2017, pp. 20–24.

[22] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, no. 5, pp. 2795–2808, 2016.

[23] A. Allavena and D. Mossé, "Scheduling of frame-based embedded systems with rechargeable batteries," in *Workshop on Power Management for Real-time and Embedded Systems (co-located with RTAS 2001)*.

[24] X. Zhang, Z. Yang, W. Sun, Y. Liu, S. Tang, K. Xing, and X. Mao, "Incentives for mobile crowd sensing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 54–67, 2016.

[25] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, pp. 927–930.

[26] I. Milchtaich, "Congestion games with player-specific payoff functions," *Games and economic behavior*, vol. 13, no. 1, pp. 111–124, 1996.

[27] H. Ackermann, H. Röglin, and B. Vöcking, "Pure nash equilibria in player-specific and weighted congestion games," in *WINE*. Springer, 2006, pp. 50–61.

[28] X. Vives, "Nash equilibrium with strategic complementarities," *Journal of Mathematical Economics*, vol. 19, no. 3, pp. 305–321, 1990.

[29] M. Johansson and A. Rantzer, “Computation of piecewise quadratic lyapunov functions for hybrid systems,” in *Control Conference (ECC), 1997 European*. IEEE, 1997, pp. 2005–2010.

[30] M. J. Neely, “Stochastic network optimization with application to communication and queueing systems,” *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[31] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge university press, 2006.

[32] R. Johari and J. N. Tsitsiklis, “Network resource allocation and a congestion game,” in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 2. Citeseer, 2003, pp. 769–778.