Modeling Real-Time Task Assignment for Mobile Crowdsourcing in Opportunistic Networks

Haruumi Imamura, Kazuya Sakai, *Member, IEEE,* Min-Te Sun, *Member, IEEE,* Wei-Shinn Ku, *Senior Member, IEEE,* and Jie Wu, *Fellow, IEEE*

Abstract—Opportunistic network-based mobile crowdsourcing (MCS) outsources location-based human tasks to a crowd of workers, where workers with mobile devices opportunistically have contact with the server. While a number of task assignment algorithms have been proposed for different objectives, real-timeness is not considered. In this paper, we are interested in real-time MCS (RT-MCS), in which tasks can be generated at any time step, and task assignment is performed in real-time. We first model an abstract RT-MCS and then instantiate the real-time task assignment problem for opportunistic network-based RT-MCS. A generic real-time task assignment (RTA) algorithm is designed based on the principle of the greedy approach, where each task is assigned to the best worker with the highest expected completion probability. To understand the fundamental performance issues, we formulate closed-form solutions for task completion probability as well as delay. In addition, we identify the critical condition that illuminates the busy state and the not-busy state of an RT-MCS. Furthermore, the analytical and simulation results demonstrate that our analysis yields close approximation of simulation results.

Index Terms—Mobile crowdsourcing, MCS, real-time mobile crowdsourcing, task assignment, opportunistic networks.

1 Introduction

With the spread of smart devices with sensing and communication capabilities, mobile crowdsourcing (MCS) [1]–[3] has become a popular application in city as well as community areas, where a user/server called *a requester* outsources location-based human tasks, such as photo taking [4], [5], park reservation [6], bus arrival time prediction [7], indoor navigation [8]–[11], and WiFi signal characterization [12]. The users who are willing to work on these tasks for pay are called *workers*. Opportunistic network-based (ON-based) MCS is one type of MCS where communications among mobile users and access points/servers are opportunistic. For example, MCS for disaster recovery [13], [14] is of this type, in which network infrastructure is not readily available and human contacts play the essential role to form a network.

An ON is a special type of wireless ad hoc network, where wireless links are often disconnected and no stable end-to-end communication link is readily available. In such a network, communication opportunities among the server and workers are limited. That is, the server can communicate with each worker only at a contact, i.e., the event in which the server and a worker are within the communication range. This opportunistic nature of ONs forces us to exploit limited communication opportunities in task assignment algorithm designs. To be specific, the task assignment and collection delay must be considered when the server assigns tasks to workers.

A number of works [13], [15]–[22] have been devoted to designing task assignment algorithms in ON-based MCS. In these works, a server (or a requester) assigns human tasks to a crowd of workers with different design goals, such as maximizing a task completion rate and minimizing makespan. In addition, quality-awareness [17], expertise awareness [15], incentive mechanisms [23], and priority-awareness [22] are also important factors for designing task assignment algorithms. The existing task assignment problems are *batch-based*. That is, a set of tasks is given at the beginning, then the server (or the requester) assigns tasks to workers in an online or offline fashion, and eventually task processing is performed within an *episode*. In other words, the real-timeness of MCS is not considered, i.e., new tasks are never generated during an episode. While the real-time task assignment problem for mobile crowdsourcing has been studied in [24], the nature of opportunistic networks is not considered.

Therefore, in this paper, we are interested in real-time task assignment for ON-based real-time MCS (RT-MCS). The contributions of this paper are as follows.

- First, we introduce a framework of real-time mobile crowdsourcing (RT-MCS), where tasks are spontaneously generated at each time step. Then, ON-based RT-MCS is instantiated with the probability distributions that characterize the randomness of the system. In addition, we formulate a real-time task assignment problem for RT-MCS, where the server assigns tasks in its task queue to a crowd of workers in real time. As performance metrics, the delay as well as completion rate of tasks are quantified.
- Second, we define the optimal task assignment with respect to the probability of tasks being completed by their dead-

- *Haruumi Imamura is with the Department of Electrical Engineering and Computer Science, Tokyo Metropolitan University, 6-6 Asahigaoka, Hino, Tokyo 191-0065, Japan. E-mail: imamura-haruumi@ed.tmu.ac.jp*
- *Kazuya Sakai is with the Department of Electrical Engineering and Computer Science, Tokyo Metropolitan University, 6-6 Asahigaoka, Hino, Tokyo 191-0065, Japan. E-mail: ksakai@tmu.ac.jp*
- *Min-Te Sun is with the Department of Computer Science and Information Engineering, National Central University, Taoyuan 320, Taiwan. E-mail: msun@csie.ncu.edu.tw*
- *Wei-Shinn Ku is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA. E-mail: weishinn@auburn.edu*
- *Jie Wu is with the Department of Computer and Information Science, Temple University, 1925 N. 12th St. Philadelphia, PA 19122. E-mail: jiewu@temple.edu*

line. Then, we propose a real-time task assignment (RTA) algorithm based on the principle of the greedy strategy that approximates the optimal assignment. The key idea of the proposed algorithm is that each task is assigned to the best worker with the highest probability of processing the task.

- Third, we derive the critical condition that clarifies the busy state and the not-busy state for a given set of system parameters. In the busy-state, newly generated tasks at the server cannot be completed by their deadline with a high probability. To understand the fundamental performance issues of RT-MCS, we formulate closed-form approximate solutions to estimate task completion probability and delay from system parameters.
- Fourth, we conduct extensive computer simulations with different system parameters to compare numerical and simulation results. The performance evaluation demonstrates that our models closely approximate the simulation results.

The rest of this paper is organized as follows. Related works are reviewed in Section 2. In Section 3, the problem of the real-time task assignment for ON-based RT-MCS is formulated. In Section 4, the RTA algorithm is proposed. The analyses of the expected completion probability and delay of completed tasks are provided in Section 5. In Section 6, the results of computer simulations are presented. Section 7 concludes this paper.

2 Related Work

2.1 Mobile Crowdsourcing

A number of mobile crowdsourcing (MCS) methods have been proposed so far. For example, photo crowdsourcing [4], [5] recruits the workers who are willing to take photos at points of interest in a city area for pay. Zhou et al. [25] propose a greedy algorithm for photo selection performed by the server during the server-to-requester stage. In [26], Hamrouni et al. propose an MCS framework for event reporting which handles the photo selection problem and eliminates wrong reports. In [27], an edge computing-based photo crowdsourcing is proposed for real-time 3D reconstruction for 5G multi-access edge computing environments. In a park reservation application [6], human power is used to monitor and find available parking slots in urban areas. In WiFi signal characterization [12], channel states at points of interest are surveyed to optimize the configurations of wireless access points. In [28], Barrón et al. present a mobile crowdsourcing data hub platform for urban infrastructure maintenance. Han et al. [23] propose a quality-aware incentive mechanism that identifies an appropriate price and assigns tasks to a group of workers with reasonable qualities. Yan et al. [29] design privacy-preserving data aggregation for an MCS with multiple requesters. Zhang et al. [30] invent an online task assignment algorithm as well as its priority-aware extension for a spatio-temporal MCS.

Different MCS methods rely on different mobility models, such as geometric-based and opportunistic network-based (ON-based) models. In geometric-based MCS [5], [31], an urban area is divided into grids, and the semi-Markov model characterizes the worker's mobility, i.e., the transitions among grids. On the contrary, in ON-based MCS, a network is constructed from opportunistic contact events among the server (or requesters) and workers. In this paper, we are interested in ON-based MCS.

2.2 Task Assignment for ON-based Mobile Crowdsourcing

The most closely related works to this paper are task assignment problems in ON-based MCS [13], [16]–[21], where an ON is used as the underlying network model. In [16], Xiao et al. propose an offline task assignment (FTA) algorithm and an online task assignment algorithm (NTA). In FTA, all the assignment decisions are made at the beginning; in NTA, a decision is made at every contact event. Xiao et al. [18] propose the average makespan sensitive online task assignment (AOTA) and the largest makespan sensitive online task assignment (LOTA). The former tries to shorten the average delay by the greedy strategy; the latter tries to reduce the worst-case delay. Mizuhara et al. [13] introduce the collaborative task assignment problem, where each task must be processed by more than one worker at the same time. In [17], Karaguchi et al. design the quality-aware task assignment (QA-TA) algorithm, in which not only makespan (or delay), but also the quality of processed tasks is considered. QA-TA applies the optimal stopping, which is a well-known statistical technique, to approximate the optimal assignment. Yucel et al. [19] present algorithms for a matching problem with coverage-aware preferences of requesters and profit-based preferences of workers in a budget constrained opportunistic MCS. In [20], Yucel et al. introduce the preference-aware task assignment problem in opportunistic MCS, considering the uncertainty in worker trajectories and capacity constraints of workers. Yucel et al. [21] design the metric that measures the utility of users for completing tasks in specific regions and propose protocols based on the metric. Sakai et al. [22] propose online and offline priority-aware tasks assignment (PNTA and PFTA) algorithms for ON-based MCS, in which some emergency tasks are prioritized to be processed.

In the aforementioned task assignment problem, a set of tasks is given at the beginning. The task assignment as well as task processing are performed within an episode. In this sense, the existing works are not real-time MCS.

3 Problem formulation

3.1 Generic Real-Time Mobile Crowdsourcing

As shown in Figure 1, a real-time mobile crowdsourcing (RT-MCS) system is composed of one server and a set of n workers, denoted by S and $W = \{w_1, w_2, \cdots, w_n\}$, where w_i denotes the worker with ID i. Let H_{all} be a set of *human tasks* (or simply tasks) that can be generated. Each human task, denoted by $h_j \in H_{all}$, is generated according to some probability distribution and then stored in S's task queue, denoted by Q_S with the maximal length being B_S. Server S will assign one or more tasks to a worker, say w_i, when they establish a communication session. Each worker w_i has a task queue as a buffer to store a set of tasks assigned by S, which is denoted by Q_i with the maximal length being B_i. For simplicity, we assume $B_i = B_{max}$ for all $w_i \in W$. Once tasks are assigned, w_i can start processing the task at the head of Q_i. Task h_j has its workload and deadline, denoted by τ_j and D_j, respectively. The workload is the number of time steps for w_i to process task h_j, and w_i must process h_j and return to server S by D_j. For simplicity, we assume that the workload of a task is the same for all workers. If h_j is not processed within the deadline D_j, that task will be dropped from S's or w_i's task queue, Q_S or Q_i. Server S will collect completed tasks from w_i's processed task set, denoted by J_i, when they again establish a communication session.

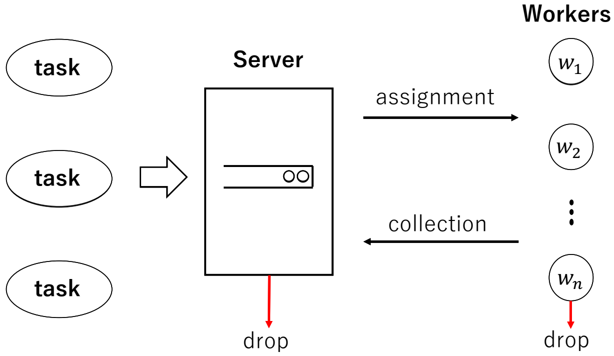


Fig. 1. An overview of real-time task assignment.

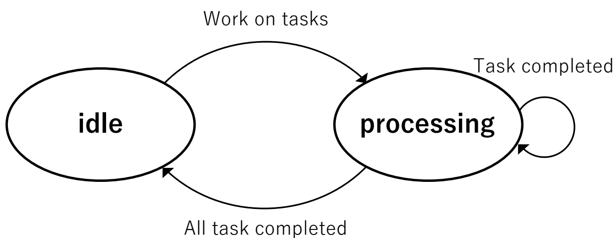


Fig. 2. The state transition of each worker.

Figure 2 shows the state transition diagram of worker w_i, where there are *idle* and *processing* states. Worker w_i is in the idle state at the beginning and will switch to the processing state once some tasks are assigned by server S. When worker w_i finishes processing all the assigned tasks, it goes back to the idle state.

The time sequence is assumed to be discrete. Let t denote a time step ranging from 0 to T. An MCS starts from time step 0 and ends at T, and such a period is called an *episode*. The MCS considered in this paper deals with real-time task assignments in the sense that generated tasks are assigned to workers in real-time and the episode is continuous, i.e., an MCS deals with an infinite number of tasks ($|H_{all}| = \infty$) until $T = \infty$.

RT-MCS differs from the existing MCS task assignment problems, which are considered as batch-based task assignment. To be specific, in [13], [15]–[18], the server (or the requester) assigns a finite set of tasks to a crowd of workers. In these works, there exists a certain time bound at which MCS terminates. Let D_{max} be the deadline of the task that has the largest timestamp among all the tasks in a given finite set of tasks. In this case, an episode completes before D_{max} or terminates at D_{max}, i.e., $T \leq D_{max}$. On the contrary, RT-MCS may generate an infinite number of tasks and there is no specific time step at which an episode ends. Therefore, we may define the real-time task assignment in RT-MCS by Definition 1.

Definition 1 (Real-Time Tasks Assignment) *The task assignment problem in MCS is said to be the real-time task assignment if* $|H_{all}| = \infty$, $T = \infty$, *and* $\Pr[h_j \texttt{ is generated at } t] > 0$ *for all* $t \in [0, \infty]$.

Probability distributions, e.g., Poisson process and exponential distributions, are involved in RT-MCS. To be specific, task generation as well as establishing communication sessions follow some probability distributions, which are application dependent. In this paper, we will instantiate an opportunistic network-based (ON-based) MCS in the next subsection.

Similar to the existing works [13], [15]–[22], we assume that workers always accept assigned tasks. This is because workers participate in an MCS system for payoff, and thus, workers are assumed to willingly accept tasks. If this assumption does not hold, the delay will increase. However, the proposed RT-MCS problem can easily handle the scenario in which workers may reject assigned tasks. Let r_i be the tasks rejection rate of worker w_i. When worker w_i rejects assigned tasks, we may ignore the contact event. In other words, the contact frequency of w_i can be defined by $\mu_i \cdot r_i$, i.e., the inter-meeting time is $\frac{1}{\mu_i \cdot r_i}$. Then, the original RT-MCS is applied for an ON with the modified contact frequency.

3.2 Instance of Opportunistic Network-Based MCS

An opportunistic network (ON) is a special type of ad hoc network, where wireless links are intermittently disconnected and no end-to-end communication path is available. The nodes in an ON can communicate with each other at a *contact*. Here, a contact is defined as an event in which two nodes, e.g., server S and worker w_i, are in the communication range. Hence, the opportunity of establishing communication sessions depends on the contact frequencies among server S and a set of workers W.

Let $\Lambda = \{\mu_1, \mu_2, \cdots, \mu_n\}$ be a set of contact frequencies, where μ_i denotes the contact frequency between S and $w_i \in W$. In general, μ_i is assumed to follow the exponential distribution [32]. In other words, the inter-meeting time between S and w_i is defined by $\frac{1}{\mu_i}$. The probability density function of server S meeting w_i at time step t is obtained by $\mu_i e^{-\mu_i t}$. Let t_0 be the current time step. The probability of S meeting w_i within a time constraint, say T', is defined by Equation 1.

$$\int_{t_0}^{T'} \mu_i e^{-\mu_i t} dt = e^{-\mu_i t_0} - e^{-\mu_i T'} \tag{1}$$

According to [33], task generation can be modeled as the Poisson process with parameter λ. That is, a task is generated for every $\frac{1}{\lambda}$ time step on average.

With the aforementioned natures of ONs and the characteristics of probability distributions, ON-based MCS works as follows. For each time step, human task h_j is generated according to the Poisson process with parameter λ. In addition, for each time step, the contact event between server S and each worker $w_i \in W$ occurs according to the exponential distribution with contact frequency μ_i. At a contact, server S can assign tasks to w_i and collect processed tasks from w_i as described in the previous section.

The notations used in this paper are listed in Table 1.

3.3 The Real-Time Task Assignment Problem

In this paper, two metrics, namely delay and completion rate, are applied to quantify the performance of task assignment algorithms as follows.

3.3.1 Delay

Let $d(h_j)$ be the delay of human task h_j, which is defined by the period of time required for server S to assign h_j to a worker (say w_i), for w_i to process h_j, and for S to collect processed task h_j from w_i. The delay $d(h_j)$ can be formulated by the sum of three kinds of delays as follows.

TABLE 1: Definition of notations.

Symbols	Definition
S	The server
w_i	Worker w_i
n	The number of workers
W	A set of workers, $W = \{w_1, w_2, \cdots, w_n\}$
μ_i	The contact frequency between S and w_i
$\frac{1}{\mu_i}$	The inter-meeting time between S and w_i
Λ	A set of contact frequencies, $\Lambda = \{\mu_1, \mu_2, \cdots, \mu_n\}$
λ	The task generation rate
$\frac{1}{\lambda}$	The inter-generating times of tasks
h_j	Human task h_j
τ_j	The workload of task h_j
D_j	The deadline of task h_j
H_{all}	The set of all the human tasks
H	The set of all the generated tasks
Q_S	The server S's task queue
Q_i	The worker w_i's task queue
B_S, B_i	The buffer sizes of Q_S and Q_i
$\tilde{Q}_i$	The estimated queue state of worker w_i
J_i	A set of processed tasks of w_i
t	A time step
T	The maximal time step
$d(h_j)$	The delay of task h_j
$c(h_j)$	The indicator function of task h_j
$C(H)$	The completion rate of task set H
$RT(.,.)$	Expected remaining time

1) **The task assignment delay**, denoted by d_a, is defined as the time elapsed in order for task h_j to be assigned to any worker, say w_i, since it has been issued. The task assignment delay is the sum of the queuing delay in S's task queue and the elapsed time since S meets w_i, both of which depend on the contact frequencies Λ and the queue's state of w_i. Note that the number of tasks polled from S's queue depends on the capacity of w_i (i.e., w_i's queue's state).
2) **The task processing delay**, denoted by d_p, is defined as the required time for worker w_i to process tasks h_j since w_i receives h_j. The processing delay depends on the amount of time h_j is in w_i's task queue and its workload τ_j.
3) **The task collection delay**, denoted by d_c, is defined as the required time for server S to collect processed task h_j from worker w_i, which depends on the contact frequency between S and w_i, i.e., μ_i.

Therefore, the delay, $d(h_j)$, is defined by Equation 2.

$$d(h_j) = d_a(h_j) + d_p(h_j) + d_c(h_j) \tag{2}$$

Since each task has its deadline D_j, only the tasks with $d(h_j) + t_j \leq D_j$ are essential. Here, t_j is the time step at which task h_j was generated. Thus, we define the *essential delay*, denoted by $\hat{d}(h_j)$, as shown in Equation 3.

$$\hat{d}(h_j) = \begin{cases} d(h_j) & \text{if } d(h_j) + t_j \leq D_j \\ D_j - t_j & \text{otherwise} \end{cases} \tag{3}$$

3.3.2 Completion Rate

If task h_j is processed and collected by its deadline, i.e., $d(h_j) + t_j \leq D_j$, then h_j is said to be *completed*. Let $c(h_j)$ be the indicator of task h_j, which equals to 1 if h_j is completed by D_j and 0 otherwise. Let $H \subseteq H_{all}$ be the set of tasks issued at server S and $\hat{H}$ be the set of completed tasks. We define the *task completion rate*, denoted by $C(H)$, as the ratio between the number of completed tasks and the number of issued tasks as formulated by Equation 4.

$$C(H) = \frac{|\hat{H}|}{|H|} = \frac{\sum_{\forall h_j \in H} c(h_j)}{|H|} \tag{4}$$

3.3.3 The Problem Definition

The most important metric in task assignment is the completion rate. Thus, we define the real-time task assignment problem for RT-MCS by Definition 2.

Definition 2 (Real-time Task Assignment Problem) *The goal of the real-time task assignment problem is to maximize the completion rate C while keeping the essential delay $\hat{d}(h_j)$ of each task h_j as small as possible.*

3.4 Research Challenges

The real-time task assignment problem is a new class of task assignment problems for MCS, and we are facing new research challenges listed as follows.

- **Challenge 1:** Most of the existing task assignment algorithms are batch-based, where the server (also called the requester) assigns a given set of tasks to a crowd of workers and an episode has a time bound. In other words, the task assignment strategy can be derived with a complete model of an MCS, i.e., the server knows the number of tasks, the workload of each task, the contact frequency of each worker, and so on. On the contrary, tasks may or may not be generated at each time step in RT-MCS, and the episode is continuous. In this case, the server must seek to find a better task assignment strategy with incomplete knowledge of the task set. These differences force us to take a different design approach from the existing ones. Therefore, the first challenge is how to integrate not only contact frequencies but also task generation rate into algorithm designs.
- **Challenge 2:** In RT-MCS, at every contact with a worker, the server decides whether or not to assign tasks to the worker depending on many factors, e.g., the workload as well as deadline of tasks, the contact frequency, and the worker's queue status. However, the worker's capacity is time-varying, which affects the processing delay and contact delay. Therefore, the second challenge is how to model the time-varying state of each worker to predict the expected processing delay and contact delay toward deriving the closed-form solutions of task completion probability and delay.
- **Challenge 3:** When too many tasks are generated with respect to the capacity of an MCS, the newly generated tasks cannot be completed within their deadlines with a high probability. The state of such a situation is called the *busy* state; otherwise, an MCS is said to be in the *not-busy* state. The critical condition is extremely important for fundamental understanding of the performance bound of an MCS. Therefore, the third challenge is to discover the

critical condition of MCS's busy state based on the system parameters.

While some task assignment algorithms consider not only task completion rates and delay but also expertise and reputation of workers, we exclude such considerations from this paper. Our RT-MCS is built upon an opportunistic network, where the contact events among the requester and workers are opportunistic, i.e., the opportunities that the requester can assign tasks to workers are limited. Therefore, we emphasize the network aspect of the task assignment particularly arisen in opportunistic networks without considering the expertise/reputation of workers.

4 Real-Time Task Assignment Algorithm

4.1 Overview

In this section, the real-time task assignment (RTA) algorithm is proposed, in which tasks are randomly generated according to the Poisson distribution and server S dynamically assigns these tasks to a crowd of workers at contacts.

First, the expected optimal assignment is introduced which defines the best worker to process task h_j. Here, the best worker refers to the worker with the highest probability of processing task h_j and returning the processed task to server S by deadline D_j. To this end, we will introduce the concept of the expected remaining time, and then maximizing the expected remaining time is equivalent to maximizing the completion probability.

In reality, the optimal assignment is unfeasible due to many factors. Thus, we will design a greedy-based algorithm that determines the best worker based on server S's state and worker w_i's state for all $w_i \in W$ by approximating the expected remaining time.

4.2 Expected Optimal Task Assignment

In this section, the expected optimal task assignment, which maximizes the completion rate, is formulated. Note that the expected optimal task assignment differs from that of the batch-based task assignment problem in the sense that the optimal assignment defines the best worker, say w^*, who can process a given task, h_j, with the highest probability among all the workers.

Let $RT(w_i, h_j)$ be the remaining time after server S collects processed task h_j from worker w_i, which is defined by Equation 5.

$$RT(w_i, h_j) = D_j - t - d_a(h_j) - d_p(h_j) - d_c(h_j) \quad (5)$$

Let $W_c(t)$ be the set of workers that server S has contact with at time step t. For simplicity, we assume that task h_j is generated at time step $t = 0$. Assume that task h_j is located at the head of server S's queue Q_S, and worker w_i has no task in her task queue, i.e., $Q_i = \emptyset$. Such conditions can be met if $|W| = \infty$ (i.e., there will be at least one worker with no task in her task queue) and $|W_c(t)| = \infty$. That is, all the tasks in the queue are immediately assigned to the workers in $W_c(t)$. As a result, there will be no queue delay at the server's queue Q_S, and $d_a(h_j)$ is dominated by the inter-meeting time. Thus, $RT(w_i, h_j)$ is bounded from above by Equation 6.

$$RT(w_i, h_t) \quad \leq \quad \begin{cases} D_j - \tau_j - \dfrac{1}{\mu_i} & \text{if } w_i \in W_c(t) \\ D_j - \tau_j - \dfrac{2}{\mu_i} & \text{otherwise} \end{cases} \quad (6)$$

We will show that minimizing the delay of task h_j (i.e., maximizing the remaining time) is equivalent to maximizing the completion probability of h_j. Let $c(w_i, h_j)$ be the indicator function that returns 1 if task h_j is completed by worker w_i and collected by server S by deadline D_j, and 0 otherwise. We may derive Theorem 1.

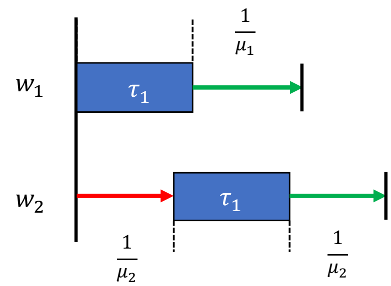


Fig. 3. An example of optimal task assignment.

Theorem 1 $\Pr[c(w_i, h_j) = 1] > \Pr[c(w_k, h_j) = 1]$ *for all* $w_k \in W\backslash\{w_i\}$, *if and only if* $RT(w_i, h_j) > RT(w_k, h_j)$ *holds.*

Proof: The probability of $c(w_i, h_j) = 1$ at a particular time instance t is formulated by Equation 7.

$$\Pr[C(w_i, h_j) = 1] = 1 - \exp\left[-\mu_i \left(RT(w_i, h_j) + \frac{1}{\mu_i}\right)\right] \quad (7)$$

$$= 1 - \frac{1}{\exp\left[\mu_i RT(w_i, h_j) + 1\right]} \quad (8)$$

Hence, $Pr[c(w_i, h_j) = 1]$ is monotonically increased when $RT(w_i, h_j)$ increases. Therefore, the above claim must be true. ∎

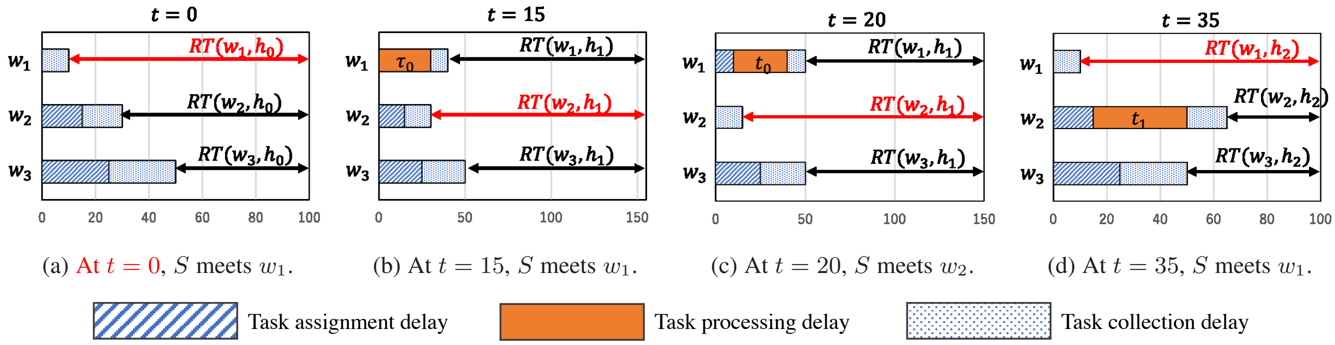
The optimal assignment of h_j is defined by the worker that maximizes $RT(w_i, h_j)$ for all $w_i \in W$ by Definition 3.

Definition 3 *The optimal assignment of* h_j *is defined by worker* w^*, *such that* $w^* = \underset{\forall w_i \in W}{\operatorname{argmax}} RT(w_i, h_j)$.

Example of the optimal assignment Figure 3 shows an example of the optimal task assignment. Consider that there are two workers, w_1 and w_2, whose expected inter-meeting times are $\frac{1}{\mu_1} = 40$ and $\frac{1}{\mu_2} = 30$, respectively. Assume that task h_1 with $\tau_1 = 30$ and $D_1 = 200$ is issued at $t = 0$. Server S has a contact with w_1 at $t = 10$, and task h_1 is at the head of Q_S. If server S will determine which of w_1 and w_2 is the best worker to assign h_1 to, according to Equation 5, we can deduce $RT(w_1, h_1) = 200 - 10 - 30 - 40 = 120$ and $RT(w_2, h_1) = 200 - 10 - 30 - 30 - 30 = 100$. Hence, $\underset{w_k \in \{w_1, w_2\}}{\operatorname{argmax}} RT(w_k, h_1) = w_1$, and worker w_1 is the best worker for assigning task h_1 to.

4.3 Expected Delay

The key to identifying the best worker for processing each task is to approximate the expected remaining time in which the delay of tasks must be estimated. The remaining time when worker w_i processes task h_j is computed by $RT(w_i, h_j) = D_j - d_a(h_j) - d_p(h_j) - d_c(h_j)$. However, all three kinds of delays involve random variables. Among them, the task collection delay $d_c(h_j)$ can be easily estimated by $\frac{1}{\mu_i}$.



(a) At $t = 0$, S meets w_1. (b) At $t = 15$, S meets w_1. (c) At $t = 20$, S meets w_2. (d) At $t = 35$, S meets w_1.

Fig. 4. Example of the real-time task assignment algorithm.

For the assignment delay and processing delay, we will estimate $d_a(h_j)$ and $d_p(h_j)$ as elaborated on the subsequent subsections.

4.3.1 *Expected Assignment Delay*

For server S to assign task h_j to a worker, the task must be at the head of her task queue, Q_S. Thus, the remaining time toward the deadline when h_j moves to the head of Q_S will be $D_j - t$, where the t represents the current time step. Recall that $W_c(t)$ denotes a set of workers who are in contact with server S at time step t. If $w_i \in W_c(t)$, there will be no assignment delay for server S to assign h_j to w_i. Otherwise, it will take $\frac{1}{\mu_i}$ on average. Therefore, the assignment delay, $d_a(h_j)$, can be approximated by Equation 9.

$$d_a(h_j) \approx \begin{cases} 0 & \text{if } w_i \in W_c(t) \\ \dfrac{1}{\mu_i} & \text{otherwise} \end{cases} \tag{9}$$

4.3.2 *Expected Processing Delay*

For worker w_i and task h_j, the expected processing time of h_j is defined by the sum of the workload of the tasks that worker w_i currently has in her task queue, denoted by Q_i, and the workload of h_j. However, server S does not have direct access to Q_i. Hence, we define the estimated queue state of worker w_i at time step t by $\tilde{Q}_i$. Thus, processing delay $d_p(h_j)$ can be estimated by Equation 10.

$$d_p(h_j) \approx \sum_{\forall h_k \in \tilde{Q}_i} \tau_k + \tau_j \tag{10}$$

4.3.3 *Expected Remaining Time*

The expected remaining time of h_j when processed by worker w_i is approximated by Equation 11.

$$\begin{aligned} RT(w_i, h_j) &= D_j - t - d_a(h_j) - d_p(h_j) - d_c(h_j) \\ &\approx \begin{cases} D_j - t - \left(\displaystyle\sum_{\forall h_k \in \tilde{Q}_i} \tau_k + \tau_j \right) - \dfrac{1}{\mu_i} & \text{if } w_i \in W_c(t) \\ D_j - t - \left(\displaystyle\sum_{\forall h_k \in \tilde{Q}_i} \tau_k + \tau_j \right) - \dfrac{2}{\mu_i} & \text{otherwise} \end{cases} \end{aligned} \tag{11}$$

4.4 RTA Algorithm

The input to the RTA algorithm includes server S, a set of workers W, a set of human tasks H_{all}, a set of contact frequencies Λ, and a task generation rate λ. In addition to these parameters, the server S's local variables include Q_S and $\tilde{Q}_i$ for all $w_i \in W$, whose sizes are bounded by B_S and B_{max}, respectively. On the contrary, worker w_i has two local variables, Q_i and J_i, both of which are bounded by $B_i = B_{max}$ for all $w_i \in W$. Here, J_i is the set of tasks completed by w_i.

The pseudocode of RTA is shown in Algorithm 1. The server S's task queue Q_S, the estimated worker's state $\tilde{Q}_i$, the worker w_i's task queue Q_i, and the worker w_i's processed task set J_i for all $w_i \in W$ are initialized as empty sets. At each time step, a new task is generated by the Poisson distribution with parameter λ, which is written as $h \leftarrow_{Poisson} H_{all}$. Then, the new task h is enqueued to Q_S.

Upon having contact with worker w_i at time step t, server S runs the task assignment decision as shown in lines 7 to 16. Server S peeks the task at the head of Q_S, say h_j. Then, server S approximates the best worker for processing tasks h_j by $w^* = \underset{\forall w_k \in \hat{W}}{\operatorname{argmax}} RT(w_k, h_j)$, where $\hat{W}$ is a set of available workers with their task queue not being full, i.e., $\hat{W} \subseteq W$ such that $0 \leq |\tilde{Q}_i| < B_i$ for all $w_i \in \hat{W}$. If $w^* = w_i$, then the contacted worker w_i is the best. In this case, server S assigns h_j to w_i and adds h_j to its local variable $\tilde{Q}_i$. At the worker side, the assigned task is enqueued to w_i's task queue, Q_i. If worker w_i is in the idle state, she will switch to the processing state. Otherwise, server S skips this assignment since w_i is not the best one. Should the deadline of a task expire, i.e., $t > D_j$ for h_j, then task h_j is dropped from Q_S (and from $\tilde{Q}_i$ if necessary).

The worker's processes are shown in lines 17 to 22. At each step, worker w_i processes task h_j at the head of Q_i, which will take τ_j time steps. When h_j finishes task processing, h_j is removed from Q_i and added to J_i. Note that J_i never overflows, since Q_j and J_i are of the same length. If Q_i becomes empty, then w_i goes to the idle state. Otherwise, it will stay in the processing state.

Upon having contact with server S, worker w_i returns the processed tasks to server S. The completed tasks are removed from J_i. At the server side, the completed tasks are removed from $\tilde{Q}_i$. Should the deadline of task h_j expire, i.e., $D_j \geq t$, then task h_j will be dropped from Q_i or J_i.

Example of the real-time task assignment algorithm Figure 4 shows an example of how server S assigns tasks by RTA. In this example, there are three workers, w_1, w_2, and w_3, whose inter-meeting times $\frac{1}{\mu_1}, \frac{1}{\mu_2}$, and $\frac{1}{\mu_3}$ are set to be $10, 15$, and 25, respectively. Assume that server S has three tasks in its queue,

Algorithm 1 RTA(S, W, Λ, λ, H_{all})

/* Initialization */
Server S initializes $Q_s \leftarrow \emptyset$ and $\tilde{Q}_i \leftarrow \emptyset$ for $1 \leq i \leq n$.
Worker w_i initializes $Q_i, J_i \leftarrow \emptyset$ for $1 \leq i \leq n$.
/* Task generation at server S */
$h \leftarrow_{Poisson} H_{all}$.
h is enqueued to Q_S.
/* Task assignment : server S meets with worker w_i*/
Peek h_j at the head of Q_S.
$w^* \leftarrow \underset{\forall w_i \in W}{\operatorname{argmax}} R(w_i, h_j)$.
if $w^* = w_i$ and $|Q_i| < B_{max}$ **then**
  S assigns h_j to w_i and adds h_j to $\tilde{Q}_i$
  w_i enqueue h_j to Q_i.
  if w_i is in the idle state **then**
    w_i goes to the processing state.
else
  S does not assign h_j to w_i.
/* Task processing at worker w_i */
w_i processes h_j at the head of Q_i.
if w_i finishes processing h_j **then**
  w_i removes h_j from Q_i and adds it to J_i.
if $Q_i = \emptyset$ **then**
  w_i goes to the idle state.
/* Task collection : server S meets with worker w_i */
if J_i is not empty **then**
  S collects all the tasks in J_i from w_i.
  w_i initializes $J_i \leftarrow \emptyset$.
/* Handling the tasks with missed deadline */
if there exists h_j such that $t \geq D_j$ in Q_S **then**
  h_j is dropped from Q_S.
else if there exists h_j such that $t \geq D_j$ in Q_i (or J_i) **then**
  h_j is dropped from Q_i (or J_i) and $\tilde{Q}_i$.

say $Q_S = [h_0, h_1, h_2]$ at time step $t = 0$. The workload and deadline of these tasks are set to be $\tau_0 = 30$, $D_0 = 100$, $\tau_1 = 20$, $D_1 = 170$, $\tau_2 = 25$, and $D_2 = 135$. At the beginning, each worker has no task in her queue, i.e., $Q_i = \emptyset$ for $1 \leq i \leq 3$. Assume that server S meets with w_1 at $t = 0$. Server S computes the expected remaining time for each worker and obtains $RT(w_1, h_0) = 90$, $RT(w_2, h_0) = 70$, and $RT(w_3, h_0) = 50$, as shown in Figure 4 (a). Since $RT(w_1, h_0)$ is larger than the others, server S will assign h_0 to w_1. Then, worker w_1 starts processing task h_0. Consider that server S meets worker w_1 again at $t = 15$. The state of task assignment is as shown in Figure 4 (b). At this moment, the expected remaining time will be $RT(w_1, h_1) = 115$, $RT(w_2, h_1) = 125$, and $RT(w_3, h_1) = 105$, respectively. At $t = 15$, worker w_1 has not finished task h_0, and as a result, worker w_1 is not the best worker for processing task h_1. In fact, assigning tasks to w_2 results in the largest remaining time. Hence, server S skips this contact event with worker w_1. At $t = 20$, S meets w_2, S computes the remaining time of each worker and obtains $RT(w_1, h_1) = 100$, $RT(w_2, h_1) = 135$, and $RT(w_3, h_1) = 100$, as shown in Figure 4 (c). Server S assigns task h_1 to worker w_2, because $RT(w_2, h_1)$ has the largest value among the other remaining times. After this, assume that server S has a contact with worker w_1 at $t = 35$. At this moment, worker w_1 is supposed to finish task h_0 at time step 30, since the task processing of h_0 starts at time step 0 and workload $\tau_0 = 30$. Thus, worker w_1 returns processed task h_0 to server, and worker w_1 has no tasks in her queue Q_1. On the contrary, worker w_2 is still processing h_1. Thus, the state of task assignment is illustrated in Figure 4 (d). The server computes $RT(w_1, h_2) = 90$, $RT(w_2, h_2) = 50$, and $RT(w_3, h_2) = 50$, and concludes that worker w_1 is the best worker. Then, task h_2 is assigned to worker w_1.

5 Analysis of Real-Time Task Assignment

In this section, we first derive the critical condition to categorize the state of RT-MCS into either the *busy* state or *not-busy* state. If the RT-MCS is in the busy state, newly created tasks will not be completed by their deadline with a high probability. Then, we will model the expected completion probability of tasks and the expected delay of completed tasks.

5.1 Critical Condition

Let $\bar{\mu}$, $\bar{\tau}$, and $\bar{D}$ be the average contact frequency of workers, the average workload of tasks, and the average deadline of tasks, respectively. The expected number of unprocessed tasks that a worker has, denoted by $N(\bar{\tau})$, is formulated as follows.

$$N(\bar{\tau}) = \frac{(\bar{\tau} + \frac{1}{\bar{\mu}}) \cdot \lambda}{n} \tag{12}$$

Recall that the delay of task h_j is the sum of the assignment delay $d_a(h_j)$, processing delay $d_p(h_j)$, and collection delay $d_c(h_j)$. From Equation 12, the expected processing delay of an arbitrary task, denoted by $\tilde{d}_p$, is derived as follows.

$$\tilde{d}_p = (N(\bar{\tau}) + 1) \cdot \bar{\tau} \tag{13}$$

We introduce the effective contact frequency of an unspecified worker, denoted by $\hat{\mu}$, which incorporates the contact frequency and the expected workload of each worker, as follows.

$$\hat{\mu} = \frac{1}{\frac{1}{\bar{\mu}} + (\tilde{d}_p - \bar{\tau})} \tag{14}$$

Let ρ be the utilization of a worker's task queue. Since the frequency of contact with any of n workers can be obtained by $n \cdot \bar{\mu}'$, the utilization of a task queue is formulated by Equation 15.

$$\rho = \frac{\lambda}{n \cdot \hat{\mu}} \tag{15}$$

Let $\tilde{d}$ be the delay of a newly created task, which can be estimated by the sum of assignment, processing, and collection delays, as shown in Equation 16.

$$\tilde{d} = \frac{1}{\hat{\mu}} + (N(\bar{\tau}) + \frac{1}{n}) \cdot \bar{\tau} + \frac{1}{\bar{\mu}} \tag{16}$$

The critical condition can be defined by Definition 4.

Definition 4 (Critical Condition) *RT-MCS is said to be in the busy state if Equation 17 holds, and in the not-busy state otherwise.*

$$\rho > 1 - \frac{1}{n \cdot \hat{\mu} \cdot (\bar{D} - \tilde{d})} \tag{17}$$

5.2 Completion Probability Analysis

The completion probability is defined as the probability that unspecified task h is completed within its deadline. Recall that $c(h)$ is the indicator function that returns 1 if h is completed by its deadline and 0 otherwise. We will approximate $\Pr[c(h) = 1]$.

5.2.1 Completion Probability Analysis in The Not-busy State

Using the expected effective contact frequency of a worker, the expected queuing delay in the not-busy state is computed based on the queuing theory as follows.

$$\tilde{d}_a = \frac{1}{n \cdot \hat{\mu} - \lambda} \tag{18}$$

Then, the expected completion probability of a task is formulated by Equation 19.

$$\begin{aligned} \Pr[c(h) = 1] &= \int_0^{\bar{D}-\tilde{d}_a-\tilde{d}_p} \bar{\mu} e^{-\bar{\mu}t} dt \\ &= 1 - \exp[-\bar{\mu}(\bar{D} - \tilde{d}_a - \tilde{d}_p)] \end{aligned} \tag{19}$$

Here, $\bar{D} - \tilde{d}_a - \tilde{d}_p$ is the remaining time for server S to collect processed task h from a worker.

5.2.2 Completion Probability Analysis in The Busy State

In the busy state, some tasks could be dropped. Let $N'(\bar{\tau})$ be the size of a worker's task queue for the busy state. Since many tasks will be dropped from a worker's task queue, $N'(\bar{\tau})$ is relatively small compared to $N(\bar{\tau})$. In addition, we define the expected processing delay and the expected effective contact frequency for the busy state, denoted by $\tilde{d}'_p$ and $\hat{\mu}'$, respectively. These parameters can be estimated by Equations 20, 21, and 22.

$$N'(\bar{\tau}) = \begin{cases} N(\bar{\tau}) \cdot \dfrac{n \cdot \frac{1}{\bar{\tau}}}{\lambda} & \text{if } \lambda > \dfrac{n}{\bar{\tau}} \\ N(\bar{\tau}) & \text{otherwise} \end{cases} \tag{20}$$

$$\tilde{d}'_p = (N'(\bar{\tau}) + 1) \cdot \bar{\tau} \tag{21}$$

$$\hat{\mu}' = \frac{1}{\frac{1}{\bar{\mu}} + (\tilde{d}'_p - \bar{\tau})} \tag{22}$$

Using $\hat{\mu}'$, the task dropping rate at server S's task queue, denoted by r_S, can be formulated as follows.

$$r_S = \begin{cases} \dfrac{n \cdot \hat{\mu}'}{\lambda} & \text{if } \lambda \geq n \cdot \hat{\mu}' \\ 1 & \text{otherwise} \end{cases} \tag{23}$$

Then, the expected completion probability of tasks in the busy state can be derived by Equation 24.

$$\begin{aligned} \Pr[c(h) = 1] &= r_S \int_0^{\bar{D}-(\tilde{d}'_p-\bar{\tau})-\frac{1}{\bar{\mu}}} \hat{\mu}' e^{-\hat{\mu}' t} dt \\ &= r_S(1 - \exp[-\hat{\mu}'(\bar{D} - (\tilde{d}'_p - \bar{\tau}) - \frac{1}{\bar{\mu}})]) \end{aligned} \tag{24}$$

Here, $\bar{D} - (\tilde{d}'_p - \bar{\tau}) - \frac{1}{\bar{\mu}}$ indicates the remaining time for server S to assign task h to any worker.

5.3 Delay Analysis

The delay of a completed task is defined as the number of time steps for a task to be completed since it was generated. For the busy state, the delay of tasks may be greater than the duration of remaining until their deadline. As we define the essential delay in Equation 3, the delay of only the completed tasks are important. Thus, we will approximate the expected delay, $\tilde{d}$, of tasks for the not-busy state.

Recall that $\tilde{d}$ is the sum of $\tilde{d}_a$, $\tilde{d}_p$, and $\tilde{d}_c$. The assignment delay can be obtained from Equation 18 and the processing delay is formulated as $(1 + \lfloor N(\bar{\tau}) \rfloor) \cdot \bar{\tau}$. In addition, the collection delay is simply $\frac{1}{\bar{\mu}}$. Therefore, the expected delay of tasks can be formulated by Equation 25.

$$\begin{aligned} \tilde{d} &= \tilde{d}_q + \tilde{d}_p + \tilde{d}_c \\ &= \frac{1}{n \cdot \hat{\mu} - \lambda} + (1 + \lfloor N(\bar{\tau}) \rfloor) \cdot \bar{\tau} + \frac{1}{\bar{\mu}} \end{aligned} \tag{25}$$

The expected delay for the busy state is omitted, since the delay of a newly created task, say $\hat{d}(h_j)$ of h_j, will be D_j with a high probability.

6 Performance Evaluation

To evaluate the proposed analysis model, simulation is conducted. The simulation settings are presented as follows.

6.1 Simulation Configuration

In our simulations, a mobile crowdsourcing system consists of one server S and a set of n workers, $W = \{w_1, w_2, \cdots, w_n\}$, where n ranges from 5 to 60. The inter-meeting time between server S and each worker w_i ranges from 5 to 50 time steps, and its maximal variance is set to be $0.5\bar{\mu}$, where $\bar{\mu}$ is the average contact frequency. Each task h_j is randomly generated according to the Poisson distribution with parameter λ. The inter-generating times of tasks $1/\lambda$ ranges from 5 to 25 time steps. The task workload τ_j ranges from 10 to 100 time steps and the deadline of task D_j ranges from 150 to 1050 time steps. The maximal variances of task workload and deadline are set to be $0.5\bar{\tau}$ and $0.5\bar{D}$, where $\bar{\tau}$ and $\bar{D}$ are the average task workload and average duration until deadline, respectively.

Depending on the parameter setting, a mobile crowdsourcing system is either in the busy-state or not-busy-state. We will indicate such a critical condition for individual simulation results.

The probability distributions for contact frequencies and task generations are based on the observations in the works [32], [33]. In order to make our simulations as reasonable as possible, a various set of parameters are given to approximate the task completion rate and delays. In addition, the server can store a sufficiently large number of tasks in its hard disk, and thus, the queue length limitation is excluded from the consideration of our simulations.

Each simulation lasts 7000 time steps, which is sufficiently large as a continuous episode. For each set of parameters, 1000 simulation experiments are conducted and the average performance is computed.

As performance metrics, the completion rate and delay are considered. The former is defined as the ratio between the number of completed tasks collected by the server and the number of tasks generated at the server. The latter is defined as the number of time steps for a task to be collected by the server elapsed starting from the time step when the task was issued.

6.2 Model Validations

Figures 5, 6, 7, 8, and 9 present the completion rate of tasks for both the busy and not-busy states. In these figures, each point illustrates the average completion rate of tasks, and the range represents the variations of the completion rate obtained by simulations.

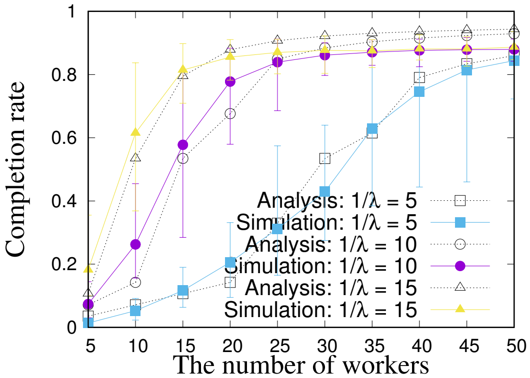


Fig. 5. The completion rate w.r.t. the number of workers.

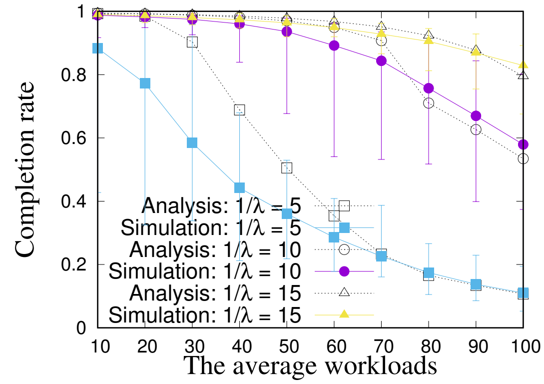


Fig. 6. The completion rate w.r.t. the average workloads.

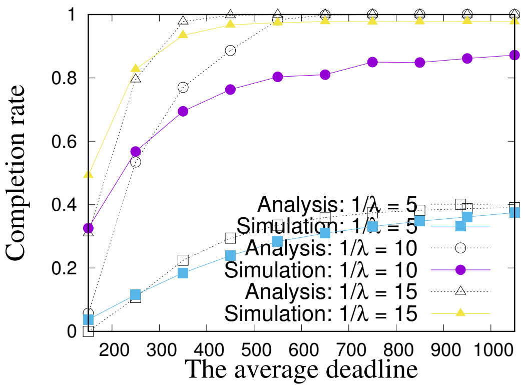


Fig. 7. The completion rate w.r.t. the average deadline.

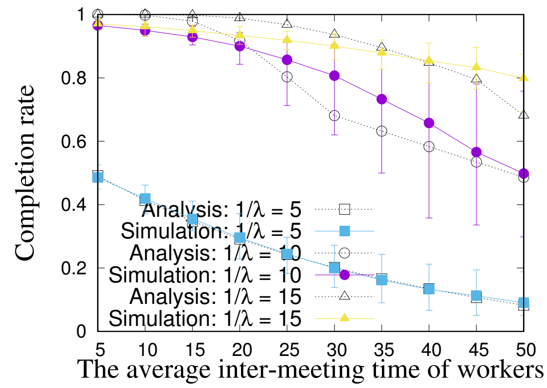


Fig. 8. The completion rate w.r.t. the average inter-meeting times of workers.

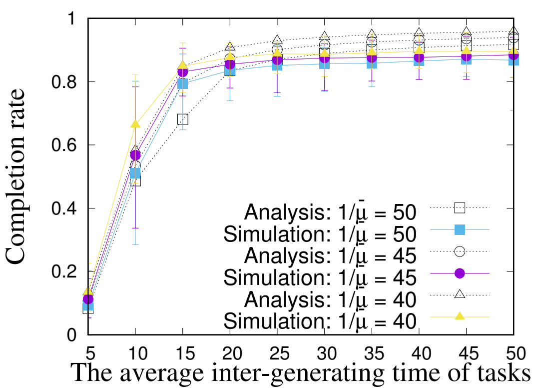


Fig. 9. The completion rate w.r.t. the inter-generating times of tasks.

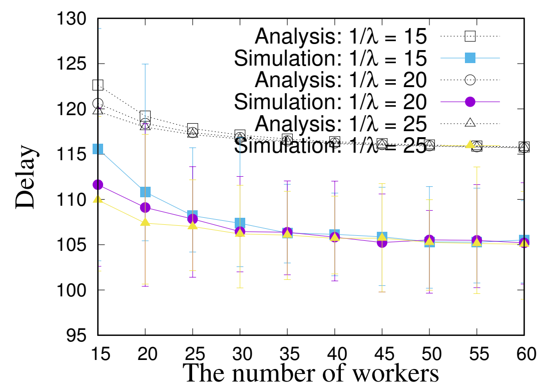


Fig. 10. The delay of completed tasks w.r.t. the number of workers.

Figure 5 shows the completion rate of tasks for different values of inter-generating times of tasks with respect to the number of workers. As can be seen in the figure, the completion rate increases in proportion to the number of workers because increasing the number of workers creates more opportunities for the server to assign tasks, and a set of workers can process tasks in parallel. Note that when $\rho \geq 1$ (i.e., $n \leq 35$ when $\frac{1}{\lambda} = 5$, $n \leq 20$ when $\frac{1}{\lambda} = 10$, and $n \leq 10$ when $\frac{1}{\lambda} = 15$), the completion rate is low because the server's queue will grow long, and many tasks will be dropped. For both the not-busy and busy states, the analytical results present the close approximation of the simulation results.

Figure 6 depicts the completion rate of tasks for different inter-generating time of tasks with respect to the average workload. It is intuitive that the completion rate decreases as the workload of a task increases. This is because the larger the workload, the more time a worker takes to process that task. This implies that the total number of tasks that can be completed by the fixed number of workers decreases. Furthermore, there are significantly lower opportunities for the server to assign tasks to workers when the system is in the busy state (i.e., $\bar{\tau} \geq 30$ when $\frac{1}{\lambda} = 5$ and $\bar{\tau} \geq 80$ when $\frac{1}{\lambda} = 10$). The analytical results present a similar trend as the simulation results for both the not-busy state and the busy state.

Figure 7 gives the completion rate of tasks for different inter-generating times of tasks with respect to the average deadline. As the average deadline of tasks increases, the completion rate increases. This is simply because workers will have sufficient time to process tasks when the deadline is long. When $\frac{1}{\lambda} = 5$, the system is extremely busy and many tasks have to wait at the server's queue before they are assigned to workers. Thus, the completion rate when $\frac{1}{\lambda} = 5$ is much lower than the others

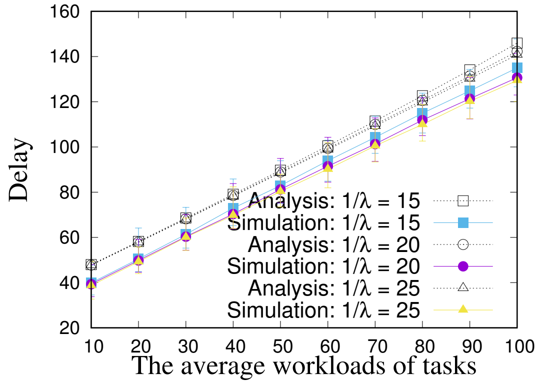


Fig. 11. The delay of completed tasks w.r.t. the average workloads.

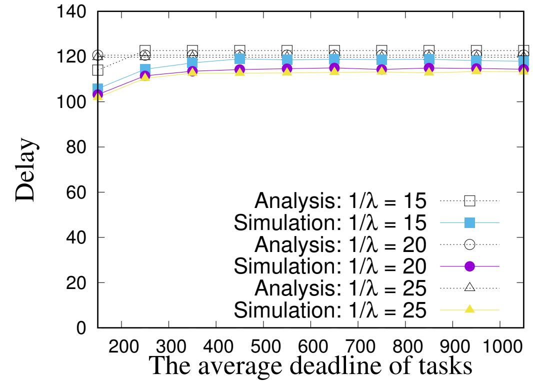


Fig. 12. The delay of completed tasks w.r.t. the average deadline.

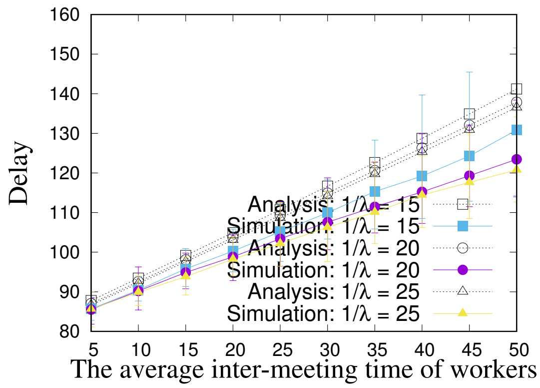


Fig. 13. The delay of completed tasks w.r.t. the average inter-meeting time of workers.

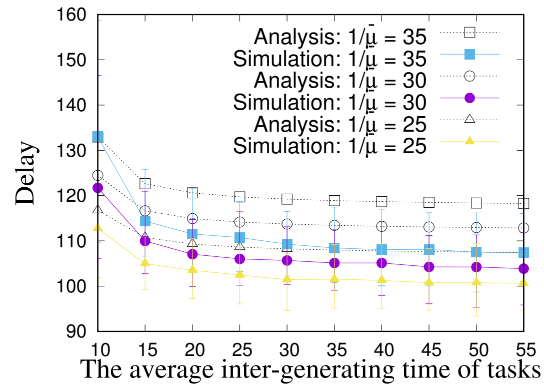


Fig. 14. The delay of completed tasks w.r.t. the inter-generating times of tasks.

regardless of the deadlines. For both the not-busy and busy states, the analytical results present relatively close approximations of the simulation results.

Figure 8 shows the completion rate of tasks for different values of inter-generating times of tasks with respect to the average inter-meeting times between the server and a worker. As can be seen in the figure, the completion rate decreases as the average inter-meeting time of workers increases, since increasing the inter-meeting time of workers reduces the opportunities for the server to assign tasks. As a result, many tasks must wait for a long time in the server's queue before they are assigned to workers. When $\frac{1}{\lambda} = 5$ (i.e., the system is in the busy state), the completion rate of tasks is much lower than that of other cases. The analytical results present a similar trend as the simulation results for both the not-busy and busy states.

Figure 9 illustrates the completion rate of tasks for different average inter-meeting times between the server and a worker with respect to the inter-generating times of tasks. As the inter-generating times of tasks increases, the completion rate increases. This is because increasing the inter-generating times of tasks reduces the number of tasks issued at the server. When $\frac{1}{\lambda_r} \leq 10$, the system is in the busy state and the completion rate of tasks is much lower than that of the other cases. For both the not-busy and busy states, the analytical results present the close approximations of the simulation results.

Figures 10, 11, 12, 13, and 14 present the delay of completed tasks for the not-busy state. In these figures, each point depicts the average delay of completed tasks, and the range represents the variations of delay obtained by simulations.

Figure 10 shows the delay of completed tasks for different values of inter-generating times of tasks with respect to the number of workers. As shown in the figure, the delay of the completed tasks gradually decreases as the number of workers increases. This is because there are more opportunities for the server to assign tasks to workers when there are more workers in the system. However, when there exists a sufficient number of workers in the system, say 35, the delay remains mostly the same. When the system is in the not-busy state, the difference between analytical and simulation results is not that significant.

Figure 11 illustrates the delay of completed tasks for different values of inter-generating times of tasks with respect to the average workload of tasks. It is intuitive that the larger workload causes workers to take a longer time to process tasks. In other words, a fixed number of workers completes fewer tasks when the workload is large. In fact, as can be seen in the figure, the delay of the completed tasks increases in proportion to the average workload of tasks. The analytical results present the close approximation of the simulation results.

Figure 12 gives the delay of completed tasks for different values of inter-generating times of tasks with respect to the average deadline of tasks. The delay of completed tasks is rarely affected by the deadline except when $\bar{D} \leq 250$. Note that all the tasks that cannot be processed within the deadline are dropped. This indicates that only the quickly processed tasks are considered when the deadline is short, i.e., the delay is not normalized. As a result, the delay tends to be short when $\bar{D} \leq 250$. On the contrary, the delay remains mostly the same for $\bar{D} \geq 300$. As can be seen in the figure, our analytical results closely approximate the simulation results even for the extreme cases, such as $\bar{D} = 150$ and $\bar{D} = 1050$.

Figure 13 depicts the delay of completed tasks for different values of inter-generating times of tasks with respect to the average inter-meeting time between the server and a worker. The delay of completed tasks increases in proportion to the average inter-meeting time. According to the definition, a task is considered to be completed when it is assigned to a worker, processed by that worker, and then collected by the server. Thus, shorter inter-meeting time between the server and each worker in the system will have a significant impact on the delay. The analytical results present the close approximation of the simulation results.

Figure 14 shows the delay of completed tasks of different average inter-meeting time between the server and a worker with respect to inter-generating times of tasks. When the value of inter-generating times of tasks increases, there will be a smaller number of tasks in the system and the assignment delay at the server, as well as the processing delay at the worker, will decrease. As a result, the delay of completed tasks gradually decreases as the inter-generating time of tasks increases. As presented in the figure, the proposed closed-form solution closely predicts the simulation results.

6.3 Comparisons with Existing Algorithms

In Figures from 15 to 24, the proposed RTA is compared with NTA [16] as an existing online task assignment algorithm. In order to compare RTA and batch-based algorithms, we introduce the task generation cycle. That is, a set of tasks are generated at the server every 100, 300, and 500 time steps according to a Poisson distribution. Each of these is denoted by Batch-100, Batch-300, and Batch-500, respectively. Note that the total number of tasks generated during the entire simulation execution is the same on average regardless of the task generation cycle, as long as the Poisson distribution parameter is the same. In addition, a cool-down period is introduced at the end of simulation, during which no task is generated, but the task processing still continues until all the issued tasks are completed or discarded.

Figures 15, 16, 17, 18, and 19 show the completion rates of different algorithms with respect to different parameter settings. The completion rates of RTA present the same trend as those of analyses. Our RTA always results in a higher completion rate compared with batch-based algorithms. As can be seen in the figures, Batch-100 presents a higher completion rate than Batch-300 and Batch-500. As a rule of thumb, the shorter task generation cycle leads to higher completion rates. This is because the queue delay at the server and workers will be long, when the task generation cycle is large. In contrast, tasks are generated in every time step in RT-MCS, and thus, RTA efficiently assigns tasks to workers.

Figures 20, 21, 22, 23, and 24 illustrate the delay of different algorithms with respect to different parameters. The delay of RTA in each figure presents the same trends as those of analyses. In addition, the proposed RTA results in shorter delay compared with the batch-based algorithms. For the batch-based algorithm, the shorter task generation cycle results in shorter delay except in the busy-state. For example, when the workload is 100, the inter-meeting time is 50, or the number of workers is 5, the delay of Batch-500 is shorter than or mostly the same as that of Batch-300. This is because the delay is computed only for the completed tasks. As a result, only the tasks with smaller workload are most likely to be processed, which leads to the shorter delay.

6.4 Discussion

From the performance evaluation, our completion analysis and delay models closely approximate the simulation results. For the completion rate, the simulation results indicate the expected trends as predicted by Equation 24. That is, the completion rate increases, when the number of workers, the average deadline, or the average inter-generation time of tasks increases; the completion rate decreases, when the average workload of tasks or the average inter-meeting time of workers increases. For the delay, some parameters make larger impact and the others do not, since there is no exponential factor as shown in Equation 25. To be specific, the delay increases in proportion to the average workload of tasks or the average inter-meeting time of workers. On the contrary, the number of workers, the average deadline, and the average inter-generation time of tasks do not make much impact on the delay in the non-busy state. This is because the delay is bounded between the task workload and the deadline.

From the MCS service provider's and MCS client's perspectives, our approximate models can be used as follows. The expected task completion probability can be used for admission controls, i.e., the service provider can decide if new tasks generated at the server are accepted or not. The service provider can tell the expected task delay for the MCS client, when she/he outsources new tasks. The critical conditions for the busy and non-busy states prompt the service provider to smarter actions, e.g., more incentives are given to workers when the system is in the busy state.

7 Conclusion

In this paper, we first introduce the real-time task assignment problem for opportunistic network-based MCS, for which the goal is to maximize completion rate while keeping the delay as short as possible. Then, a generic real-time task assignment algorithm is proposed, where randomly generated tasks are assigned to a crowd of workers at a contact event based on the greedy strategy. The optimal assignment strategy is approximated by estimating the expected remaining time. Then, we build closed form approximation solutions for the completion rate as well as delay in order to illuminate the fundamental performance issues of the real-time task assignment in MCS. The computer simulations demonstrate that our analytical models provide close approximations of the simulation results.

References

[1] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti, "Crowdsourcing with Smartphones," *IEEE Internet Comput.*, vol. 16, no. 5, pp. 36–44, 2012.

[2] T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner, "mCrowd: A Platform for Mobile Crowdsourcing," in *SenSys*, 2009, pp. 347–348.

[3] X. Kong, X. Liu, B. Jedari, M. Li, L. Wan, and F. Xia, "Mobile Crowdsourcing in Smart Cities: Technologies, Applications, and Future Challenges," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8095–8113, 2019.

[4] Y. Wu, Y. Wang, and G. Cao, "Photo Crowdsourcing for Area Coverage in Resource Constrained Environments," in *INFOCOM*, 2017, pp. 1–9.

[5] E. Wang, Y. Yang, J. Wu, K. Lou, D. Luan, and H. Wang, "User Recruitment System for Efficient Photo Collection in Mobile Crowdsensing," *IEEE Trans. Human-Mach. Syst.*, vol. 50, no. 1, pp. 1–12, 2019.

[6] T. Yan, B. Hoh, D. Ganesan, K. Tracton, T. Iwuchukwu, and J.-S. Lee, "Crowdpark: A Crowdsourcing-Based Parking Reservation System for Mobile Phones," *University of Massachusetts at Amherst Tech. Report*, pp. 1–14, 2011.

[7] P. Zhou, Y. Zheng, and M. Li, "How long to wait? Predicting Bus Arrival Time with Mobile Phone Based Participatory Sensing," in *MobiSys*, 2012, pp. 379–392.

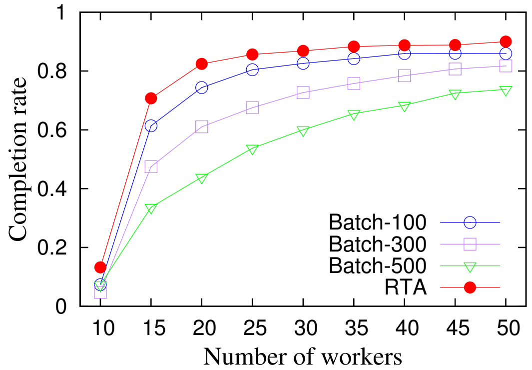


Fig. 15. The completion rate of different algorithms w.r.t. the number of workers.

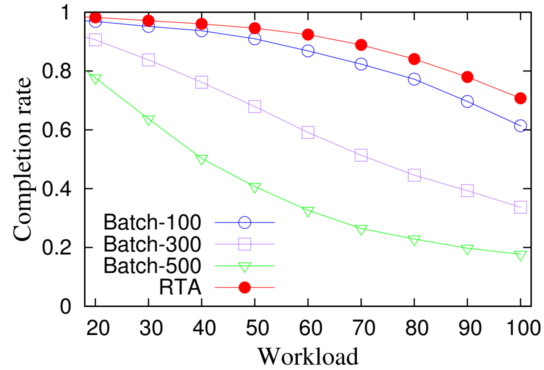


Fig. 16. The completion rate of different algorithms w.r.t. the average workloads.

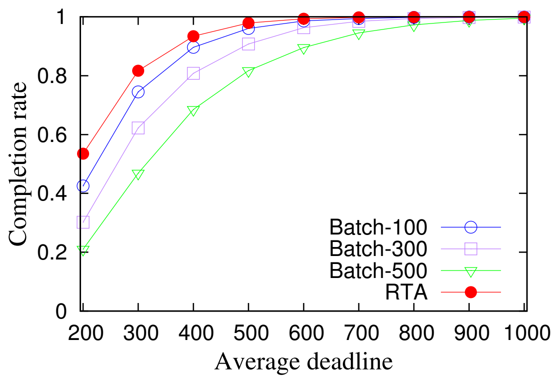


Fig. 17. The completion rate of different algorithms w.r.t. the average deadline.

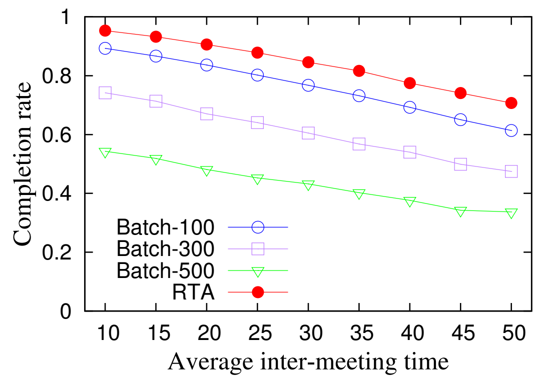


Fig. 18. The completion rate of different algorithms w.r.t. the average inter-meeting times of workers.

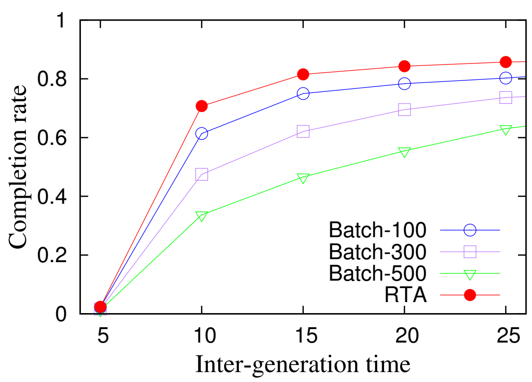


Fig. 19. The completion rate of different algorithms w.r.t. the inter-generating times of tasks.

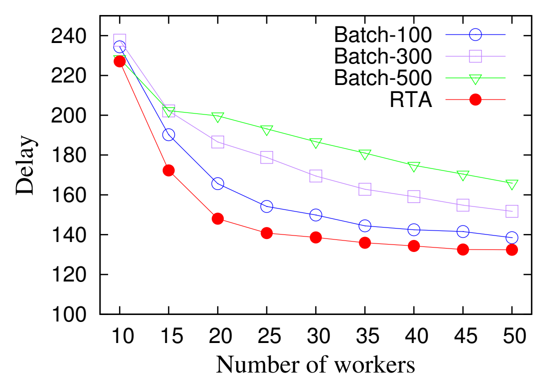


Fig. 20. The delay of completed tasks of different algorithms w.r.t. the number of workers.

[8] S. Teng, W. Ku, and K. Chuang, "Toward Mining Stop-by Behaviors in Indoor Space," *ACM Trans. Spatial Algorithms Syst.*, vol. 3, no. 2, pp. 7:1–7:38, 2017.

[9] B. Hui, C. Jiang, P. Ankireddy, W. Wang, and W. Ku, "Indoor Navigation for Users with Mobility Aids Using Smartphones and Neighborhood Networks," in *MSN*, 2021, pp. 681–682.

[10] A. I. Baba, M. Jaeger, H. Lu, T. B. Pedersen, W. Ku, and X. Xie, "Learning-Based Cleansing for Indoor RFID Data," in *SIGMOD*, 2016, pp. 925–936.

[11] J. Yu, W. Ku, M. Sun, and H. Lu, "An RFID and Particle Filter-Based Indoor Spatial Query Evaluation System," in *EDBT*, 2013, pp. 263–274.

[12] A. Farshad, M. K. Marina, and F. Garcia, "Urban wifi Characterization via Mobile Crowdsensing," in *NOMS*. IEEE, 2014, pp. 1–9.

[13] R. Mizuhara, K. Sakai, and S. Fukumoto, "A Collaborative-Task Assignment Algorithm for Mobile Crowdsourcing in Opportunistic Networks," in *ICC*, 2018, pp. 1–6.

[14] M. Garcia, J. Rodrigues, J. Silva, E. R. Marques, and L. M. Lopes, "Ramble: Opportunistic Crowdsourcing of User-Generated Data using Mobile Edge Clouds," in *FMEC*. IEEE, 2020, pp. 172–179.

[15] X. Zhang, Y. Wu, L. Huang, H. Ji, and G. Cao, "Expertise-Aware Truth Analysis and Task Allocation in Mobile Crowdsourcing," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 1001–1016, 2021.

[16] M. Xiao, J. Wu, L. Huang, Y. Wang, and C. Liu, "Multi-task Assignment for Crowdsensing in Mobile Social Networks," in *INFOCOM*, 2015, pp. 2227–2235.

[17] S. Karaguchi, K. Sakai, and S. Fukumoto, "Quality-Aware Task Assignment in Opportunistic Network-Based Crowdsourcing," in *IPCCC*, 2018, pp. 1–7.

[18] M. Xiao, J. Wu, L. Huang, R. Cheng, and Y. Wang, "Online Task Assignment for Crowdsensing in Predictable Mobile Social Networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 8, pp. 2306–2320, 2016.

[19] F. Yucel, M. Yuksel, and E. Bulut, "Coverage-Aware Stable Task Assignment in Opportunistic Mobile Crowdsensing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 4, pp. 3831–3845, 2021.

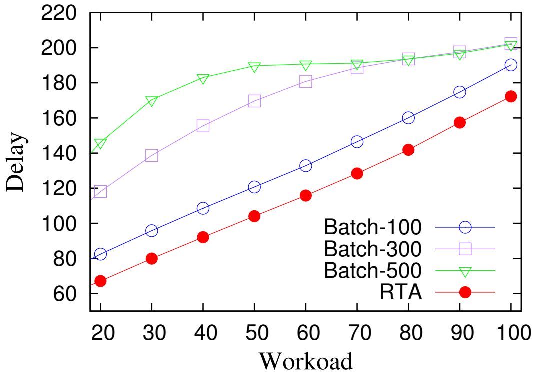


Fig. 21. The delay of completed tasks of different algorithms w.r.t. the average workloads.

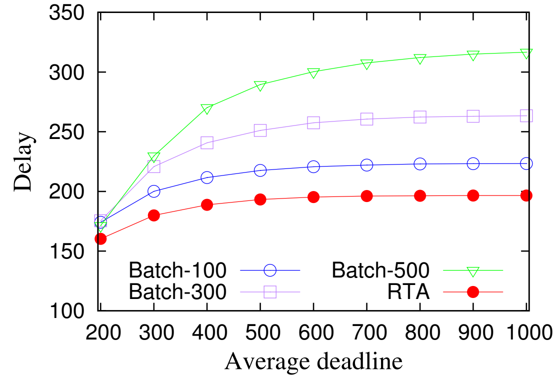


Fig. 22. The delay of completed tasks of different algorithms w.r.t. the average deadline.

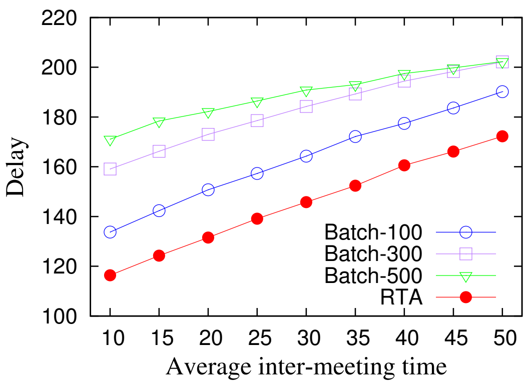


Fig. 23. The delay of completed tasks of different algorithms w.r.t. the average inter-meeting time of workers.

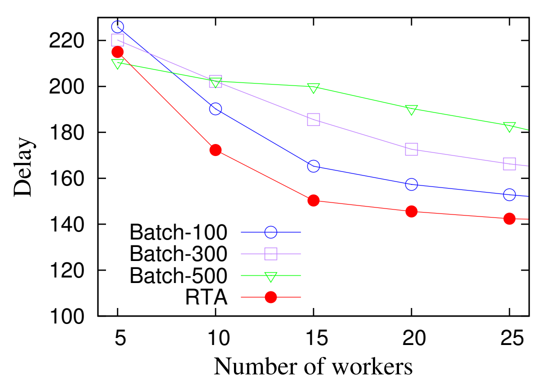


Fig. 24. The delay of completed tasks of different algorithms w.r.t. the inter-generating times of tasks.

[20] F. Yucel and E. Bulut, "Online Stable Task Assignment in Opportunistic Mobile Crowdsensing with Uncertain Trajectories," *IEEE Internet Things J.*, 2021.

[21] ——, "Location-Dependent Task Assignment for Opportunistic Mobile Crowdsensing," in *CCNC*. IEEE, 2020, pp. 1–6.

[22] K. Sakai, K. Takenaka, M. Sun, and W. Ku, "Priority-Aware Task Assignment in Opportunistic Network-Based Mobile Crowdsourcing," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 2, pp. 2124–2137, 2024.

[23] K. Han, H. Huang, and J. Luo, "Quality-Aware Pricing for Mobile Crowdsensing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1728–1741, 2018.

[24] I. Boutsis and V. Kalogeraki, "On Task Assignment for Real-Time Reliable Crowdsourcing," in *ICDCS*, 2014, pp. 1–10.

[25] T. Zhou, B. Xiao, Z. Cai, M. Xu, and X. Liu, "From Uncertain Photos to Certain Coverage: A Novel Photo Selection Approach to Mobile Crowdsensing," in *INFOCOM*. IEEE, 2018, pp. 1979–1987.

[26] A. Hamrouni, H. Ghazzai, M. Frikha, and Y. Massoud, "A Photo-Based Mobile Crowdsourcing Framework for Event Reporting," in *MWSCAS*. IEEE, 2019, pp. 198–202.

[27] S. Yu, X. Chen, S. Wang, L. Pu, and D. Wu, "An Edge Computing-Based Photo Crowdsourcing Framework for Real-Time 3D Reconstruction," *IEEE Trans. Mobile Comput.*, vol. 21, no. 02, pp. 421–432, 2022.

[28] J. P. G. Barrón, M. Á. Manso, R. Alcarria, and R. P. Gomez, "A Mobile Crowdsourcing Platform for Urban Infrastructure Maintenance," in *IMIS*. IEEE, 2014, pp. 358–363.

[29] X. Yan, W. W. Ng, B. Zhao, Y. Liu, Y. Gao, and X. Wang, "Fog-Enabled Privacy-Preserving Multi-Task Data Aggregation for Mobile Crowdsensing," *IEEE Trans. Dependable Secur. Comput.*, pp. 1–13, 2023.

[30] Q. Zhang, Y. Wang, G. Yin, X. Tong, A. M. V. V. Sai, and Z. Cai, "Two-Stage Bilateral Online Priority Assignment in Spatio-Temporal Crowdsourcing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 2267–2282, 2023.

[31] W. Li, H. Chen, W.-S. Ku, and X. Qin, "Turbo-GTS: A Fast Framework of Optimizing Task Throughput for Large-Scale Mobile Crowdsourcing," *ACM Trans. Spat. Algorithms Syst.*, vol. 6, no. 1, pp. 1–29, 2020.

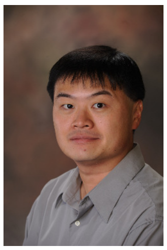
[32] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance Modeling of Epidemic Routing," *Comput. Netw.*, vol. 51, no. 10, pp. 2867–2891, 2007.

[33] M. S. Bernstein, D. R. Karger, R. C. Miller, and J. Brandt, "Analytic methods for optimizing realtime crowdsourcing," *CoRR*, vol. abs/1204.2995, 2012.

Haruumi Imamura is currently a graduate student at the Department of Electrical Engineering and Computer Science, Tokyo Metropolitan University. His research interests include distributed algorithm designs for mobile and wireless networks.

Kazuya Sakai (S'09-M'14) received his Ph.D. degree in Computer Science and Engineering from The Ohio State University in 2013. He is currently an associate professor at the Department of Electrical Engineering and Computer Science, Tokyo Metropolitan University. His research interests are in the area of information and network security, wireless and mobile computing, and distributed algorithms. He received the IEEE Computer Society Japan Chapter Young Author Award 2016. He is a member of the IEEE and ACM.

Min-Te Sun (S'99-M'02) received his B.S. degree in mathematics from the National Taiwan University in 1991, his M.S. degree in computer science from Indiana University in 1995, and his Ph.D. degree in computer and information science from The Ohio State University in 2002. Since 2008, he has been with the Department of Computer Science and Information Engineering at National Central University, Taiwan. His research interests include distributed algorithm design and wireless network protocol development.



Wei-Shinn Ku (S02-M07-SM12) received his Ph.D. degree in computer science from the University of Southern California (USC) in 2007. He also obtained both the M.S. degree in computer science and the M.S. degree in electrical engineering from USC in 2003 and 2006, respectively. He is a professor with the Department of Computer Science and Software Engineering at Auburn University. He was a Program Director with the National Science Foundation between 2019 and 2022. His research interests include databases, data science, mobile computing, and cybersecurity. He has published more than 170 research papers in refereed international journals and conference proceedings. He is a senior member of the IEEE and a member of the ACM.

Jie Wu is Laura H. Carnell Professor at Temple University and the Director of the Center for Networked Computing (CNC). He served as Chair of the Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University, where he received his Ph.D. in 1989. His current research interests include mobile computing and wireless networks, routing protocols, network trust and security, distributed algorithms, applied machine learning, and cloud computing. Dr. Wu regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing, IEEE/ACM Transactions on Networking, and Journal of Computer Science and Technology. Dr. Wu is/was general chair/co-chair for IEEE DCOSS09, IEEE ICDCS13, ICPP16, IEEE CNS16, WiOpt21, ICDCN22, IEEE IPDPS'23, ACM MobiHoc'23, and IEEE CCGrid 2024 as well as program chair/cochair for IEEE MASS04, IEEE INFOCOM11, CCF CNCC13, and ICCCN20. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a Fellow of the AAAS and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He is a Member of the Academia Europaea (MAE).