Connectivity-Aware Task Outsourcing and Scheduling in D2D Networks

(Invited Paper)

Zhen Hong, Zehua Wang, Wei Cai, and Victor C.M. Leung
Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, BC, Canada V6T 1Z4
Email: {hongz, zwang, weicai, vleung}@ece.ubc.ca

***Abstract*—With the flourishing of smart mobile devices (e.g., smartphones, tablets), development of mobile cloud computing has received more and more attentions from both industry and academia. Compared with the traditional way of executing large-scale computational tasks on powerful desktop computers and the cloud, mobile cloud computing is featured by the ubiquitous availability, flexibility, and low-cost. However, this feature also brings challenges when we build the satisfactory mobile computing system. First, the computational power on a mobile device is not comparable with that on a personal computer such that many computation-intensive tasks cannot be independently handled by mobile devices. Second, offloading computational tasks to the cloud introduces additional monetary costs (e.g. wireless communication cost, computational service cost), which may be pricy for users. In this paper, we propose a novel connectivity-aware task scheduling paradigm to enable mobile device users to accomplish computation-intensive tasks cooperatively in the device-to-device (D2D) network by incorporating the "fog" — aggregate of computational powers in the ad-hoc. A supernode at the base station is responsible for scheduling cooperation tasks based on user mobility. To further enhance the quality of experience (QoE) for the users, we propose a lightweight heuristic algorithm to perform task scheduling to ensure low cooperative task execution time. Simulation results show that our cooperative paradigm efficiently reduces the average task execution time for mobile device users in the D2D network.**

I. Introduction

Advances in computing technology are transforming the way people execute computational tasks for daily applications like stock trading [1], gaming [2], etc. Usage of traditional desktop computers for large computation works has been expanded to various ways of computing such as cloud computing. For example, cloud gaming platforms PlayStation Now [3] and GameFly [4] execute most gaming computational tasks on the cloud, which frees gamers from having to update their computing devices frequently. Stock market investors are now able to manipulate stock trading on their mobile devices by offloading most computational tasks to the cloud [1]. In recent years, with the explosion of smart mobile devices and their capacities in terms of computing power, storage, data transmission efficiency, etc., the concept of fog computing [5] is facilitated to overcome high data cost and mobility constraints.

As illustrated in Fig. 1, fog is an extension of the cloud: fog is lower and closer to users than the cloud; it consists of various user smart devices, computers and other computing units in the ad-hoc. Although it is widely adapted contemporarily to offload computational tasks to the cloud as the fog does not have as high density (calculation and storage capacities), fog computing prevents high carrier data transmission cost and cloud computing service cost, and its presence in users' vicinity allows short communication latency. Intermittent access to cellular data and non-seamless wireless coverage in the mobile environments are also discouraging factors for users to completely rely on the cloud. Faster and more responsive task cooperation and offloading in the ad-hoc becomes even more necessary in extreme situations like earthquake response.

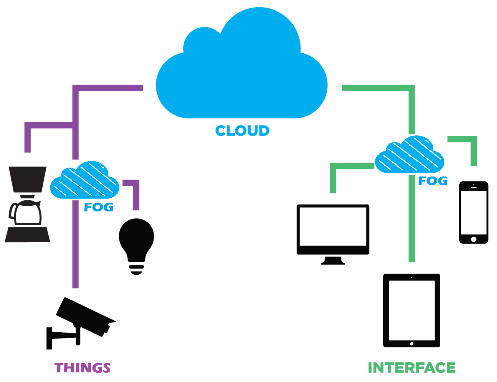


Fig. 1. Cloud and Fog Computing

The work in [6] shows that despite increasing usage of mobile devices in our daily lives, most of the computational power of these smart devices are still in the idle state and wasted, e.g. only email notification listeners and other low consumption applications run at the background for most of the time. If we can take advantage of computational power of these idle devices together with their storage and data layover abilities, cost-effective task cooperation in the device-to-device (D2D) networks is highly feasible. Such task cooperation and offloading context was first presented in Serendipity [7], a system that allows a mobile initiator to utilize computational resources available in other mobile systems in its surrounding to accelerate computing and save energy, whose performance is further analyzed in [8] to see significant potential gain in both execution time and device energy. The authors of [9] proposed a mobile application that enables cooperation of computationally intensive applications by making use of computational powers of mobile devices in nearby cloudlet.

While many previous works tried to exploit how idle computational power can be effectively utilized in the D2D networks, mobility aspects of users, especially the task cooperation scheduling in the mobile environments, still remain open issues. In this work, we will be the first to propose a task outsourcing and scheduling scheme that is probabilistically based on the mobility of smart mobile device users in an ad-hoc. The remaining part of this work is organized as follows. Section II gives an overview on related works. Section III presents details in the design of our proposed scheme. Section IV illustrates the problem formulation of our proposed scheme and Section V shows corresponding experimental evaluation results. Section VI discusses the limitations, challenges, and opportunities of the proposed scheme and our future work. Section VII concludes our work in this paper.

II. Related Work

To relieve the burden of wireless cellular networks and the cloud, mobile data and computational traffic can be delivered through other means to the users (*e.g.*, WiFi, D2D communications). This is known as mobile data and computation offloading. Several works have identified the benefits of WiFi data offloading [10], [11], [12]. The work in [10] showed that deferring the uploading tasks until WiFi access points are available can save the energy of smartphones. By jointly considering the power consumption and link capacity of wireless network interfaces, Ding *et al.* in [11] studied the criterion of downloading data from WiFi as well as the WiFi access point selection problem. Ge *et al.* in [13] investigated the data offloading with the joint consideration of cumulative distribution function based scheduling [14]. With a budget of energy consumption and monetary cost, the download duration is minimized in [12] by allocating the data traffic demand to wireless cellular and WiFi networks. However, mobile data traffic cannot always be offloaded to WiFi networks since the number of open-accessible WiFi access points is limited [11]. To fully exploit the benefits of data and computation offloading, mobile traffic can also be delivered via D2D networks. Specifically, mobile devices in close proximity can be connected via WiFi Direct [15] or Bluetooth in a D2D manner for data and task cooperation between users. This is referred to as D2D data and computation offloading. [7] and [8] explores task cooperation of mobile devices in the D2D network and showed that significant execution time and device energy can be saved. Authors of [16] presented a framework for opportunistic storage and processing in the mobile cloud. The work in [17] considers D2D technologies as candidates to deal with most local communications and time-sensitive computations in near future. A D2D network should make use of Bluetooth, WiFi-Direct, and other protocols to more efficiently provision services to applications such as video gaming and image processing.

It has been shown in [6] that the computational power of our smart mobile devices are idle and wasted for most of the time before these devices become outdated and replaced with newer models. The work in [18] presents that contemporary smart devices (mostly quad-core devices) use less than two cores on average in their non-idle states with the consideration of simultaneously running applications in the background, not to mention the computing power that these devices can provide in their idle states. It is generally true that building more data centres can provision more computational powers for end users. However, a data centre needs to be built and maintained at a very high cost, which encourages us to exploit the task cooperation possibilities in the D2D networks bearing users' mobility.

Chi *et al.* in [19] designed an ad-hoc cloudlet based cooperative cloud gaming system. This tiered system defines the computing power of various mobile smart devices in the ad-hoc as ad-hoc mobile cloud; fixed computing units with strong processing capacity that are deployed in the ad-hoc are defined as an ad-hoc cloudlet. As a result, a three-tier cloud—cloudlet—ad-hoc-mobile-cloud system is formed to coordinate each user's need for cooperation in gaming contents and computational tasks in the system based on each user's specific preference on device energy, bandwidth, and response latency. This work takes the computational power of the fog into consideration, but does not take into account the impact of the mobility of these smart devices on task cooperation from a statistical point of view. Our work statistically considers the mobility of all helpers around each requesters, and slices a task into smaller pieces and allocates them to different sections of the system. Meanwhile, this work assumes that one device may seek for computing resource from only one other device In the D2D network for one computing task. Such assumption may limit the scheduling flexibility that may lead to lower quality of experience (QoE) for requesters in the system, which is assumed to be tackled in our work with dynamic program slicing techniques.

Authors of [20] investigated into fog computing and its characteristics, roles and potential applications in the area of Internet of Things (IoT). The work considers the fog as provider of excellent localization that enables low latency and context awareness. It demonstrated the roles of fog computing in connected vehicles, smart grid, wireless sensors, and actuators networks, and suggests that many future applications require the cooperation of both fog localization and cloud globalization. In our work, we extend the idea of fog-cloud cooperation in IoT to a more general application in the ad-hoc.

Considering mobile nature of smart device users in the ad-hoc, Wang *et al.* in [21] proposed a metric, expected available duration (EAD), based on the mobility and similarities of users' interests in the D2D network. EAD indicates the statistically determined expected duration of each user's interested files in the D2D network. With this metric, this work presents an optimization and performance promotion of a file sharing system in the D2D network to reduce the expensive data charge from cellular carriers and download more data from neighbours.

Despite the previous research efforts on mobile cloud computing, no approaches have been proposed to handle D2D task outsourcing and scheduling with connectivity-awareness. In our work, we novelly present the statistical model of task outsourcing and scheduling in the D2D network and propose a light-weight heuristic algorithm for computing the task scheduling that effectively enhances user QoE.

III. System Modelling

In this section, we will first introduce the basic system settings, and then present the connectivity model, dynamic program slicing and task cooperation scheduling backgrounds for our system.

A. Basic System Settings

We divide the set of all system members with a smart device in the D2D network into two member sets, namely, requester set P and helper set H. There are p requesters in P and h helpers in H, where $p \geq 1$ and $h \geq 1$. Each requester is denoted as $p_i \in P$ and each helper is denoted as $h_j \in H$, where $i \in \{1, 2, ..., p\}, j \in \{1, 2, ..., h\}$. The smart device of a requester p_i or helper h_j is subject to a D2D communication range r_i^p or r_j^h, respectively, above which D2D direct link connection is not possible. c_i^p or c_j^h is used to denote the available computational power of a requester or helper, indicating how fast or how much computation the smart device is able to handle per second for our cooperative scheme. Typically, this type of computational power is represented by how many clock cycles the device can run per second, 2.6 GHz for example. At any moment $\tau \geq 0$, each requester p_i initializes a task of complexity $T_{i,\tau}$ in clock cycles (indicating how many clock cycles needs to be run to get the result of the task) with its maximum wait time $t_{i,\tau}$ in seconds (indicating the maximum time p_i will wait for the result until he/she has to do the assigned uncompleted task slices by himself/herself).

B. Connectivity Model

Assuming the connection between a requester p_i and a helper h_j is symmetric, we denote the random variable $B_{i,j}(\tau) = 1$ (or $B_{i,j}(\tau) = 0$) to represent that p_i and h_j are connected (or disconnected) at $\tau \geq 0$. Moreover, let random variable $S_{i,j}^1$ denote the sojourn time that p_i and h_j are in the connected state, and $S_{i,j}^0$ denote that in the disconnected state. We consider both $S_{i,j}^1$ and $S_{i,j}^0$ follow the exponential distribution with parameters $\lambda_{i,j}$ and $\mu_{i,j}$, respectively. Therefore, we have the cumulative distribution functions (CDF) of $S_{i,j}^1$ and $S_{i,j}^0$ given by

$$Pr(S_{i,j}^1 \leq \tau) = 1 - e^{\mu_{i,j}\tau}, \tag{1}$$

and

$$Pr(S_{i,j}^0 \leq \tau) = 1 - e^{\lambda_{i,j}\tau}. \tag{2}$$

We represent the continuous time Markov chain (CTMC) model with two states illustrated in Fig. 2, and let $P_{i,j}(\tau)$ denote the 2×2 matrix with entries $p_{i,j}^{xy}(\tau) = Pr(B_{i,j}(\tau) = y | B_{i,j}(0) = x)$, where $x, y \in \{0, 1\}$. Referring to [22], we have the solution of $P_{i,j}(\tau)$ given by

$$P_{i,j}(\tau) = \begin{pmatrix} \frac{\lambda_{i,j}}{\psi} + \frac{\mu_{i,j}}{\psi}\kappa & \frac{\mu_{i,j}}{\psi} - \frac{\mu_{i,j}}{\psi}\kappa \\ \frac{\lambda_{i,j}}{\psi} - \frac{\lambda_{i,j}}{\psi}\kappa & \frac{\mu_{i,j}}{\psi} + \frac{\lambda_{i,j}}{\psi}\kappa \end{pmatrix}, \tag{3}$$

where $\kappa = e^{-(\mu_{i,j}+\lambda_{i,j})\tau}$, and $\psi = \mu_{i,j} + \lambda_{i,j}$.

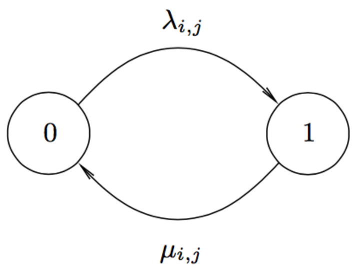


Fig. 2. Continuous-time Markov chain for the connected-disconnected transition between p_i and h_j. Let 1/0 represent the state that p_i and h_j are connect/disconnected. The transition rates from 0 to 1 and from 1 to 0 are given by $\lambda_{i,j}$ and $\mu_{i,j}$, respectively

The parameters $\mu_{i,j}$ and $\lambda_{i,j}$ used in (3) for p_i and h_j can be obtained by maximum likelihood estimation (MLE) on each of them. Specifically, without loss of generality, consider p_i and h_j were disconnected initially and the connectivity between p_i and h_j has changed m times before the current time τ. Therefore, p_i and h_j have recorded a vector of time $\vec{\tau}_{i,j} = (\tau_1, ..., \tau_m) \in R_+^m$, where each element $\tau_{i,j}^k < \tau(k = 1, ..., m)$ represents the time when the connectivity between p_i and h_j changed. Assume p_i and h_j are currently connected (then m must be an odd number and the case that p_i and h_j are currently disconnected can be analyzed via following approach similarly), the $\mu_{i,j}$ and $\lambda_{i,j}$ estimated by MLE up to current time τ are given by

$$\hat{\mu}_{i,j}^t = \frac{m-1}{2\sum_{k=1}^{\frac{m-1}{2}}(t_{i,j}^{2k} - t_{i,j}^{2k-1})}, \tag{4}$$

and

$$\hat{\lambda}_{i,j}^t = \frac{m-1}{2\sum_{k=1}^{\frac{m-1}{2}}(t_{i,j}^{2k+1} - t_{i,j}^{2k})}. \tag{5}$$

Because the connections between p_i and h_j are assumed symmetric, same results are obtained on p_i and h_j.

For the people study or work together, $\vec{\tau}_{i,j}$ is kept being recorded by both p_i and h_j as the system time increases. According to (4) and (5), $\hat{\mu}_{i,j}^t$ and $\hat{\lambda}_{i,j}^t$ will converge. We denote $\hat{\mu}_{i,j} = \lim_{t\to\infty} \hat{\mu}_{i,j}^t$ and $\hat{\lambda}_{i,j} = \lim_{t\to\infty} \hat{\lambda}_{i,j}^t$, which are the MLE of $\hat{\mu}_{i,j}$ and $\hat{\lambda}_{i,j}$, respectively.

Given the connection station $B_{i,j}(\tau)$ between p_i and h_j at time τ, the probability that they are connected at future time

$\tau' \geq \tau$ is given by

$$Pr(B_{i,j}(\tau') = 1|B_{i,j}(\tau)) \\ = \begin{cases} \frac{\lambda_{i,j} - \lambda_{i,j} e^{-(\lambda_{i,j}+\mu_{i,j})(t'-t)}}{\lambda_{i,j}+\mu_{i,j}}, \; B_{i,j}(\tau) = 0, \\ \frac{\lambda_{i,j} + \mu_{i,j} e^{-(\lambda_{i,j}+\mu_{i,j})(t'-t)}}{\lambda_{i,j}+\mu_{i,j}}, \; B_{i,j}(\tau) = 1. \end{cases} \quad (6)$$

C. Dynamic Program Slicing

In general, computational tasks cannot be arbitrarily sliced into different parts. However, many dynamic program slicing techniques is facilitating our need for distribution of tasks [23]. For example, MapReduce [24] allows Google to slice and run an average of one hundred thousand MapReduce jobs every day from 2004-2008. Without the availability of dynamic program slicing, a large load of task may not be sent back to the requester in a timely manner, and increases the risk of helper being out of the device communication range of the requester on completion of the task execution. Therefore, we adopt the assumption that tasks can be sliced in an arbitrary manner in our system.

D. Task Cooperation Scheduling

In our system, we assume that a computing unit, which we refer to as a *supernode*, with enough capacity to perform task cooperation scheduling for all devices is available at the cellular base station covering the D2D network. At the beginning of each task period, the supernode collects task and connectivity information sent wirelessly from the devices and compute the task scheduling for requesters and helpers in the D2D network based on probability of connection among them according to (6). Note that the task and connectivity information sent to the supernode is very limited in size and transmission time, which are assumed to be negligible for simplicity. Each task period is divided into discrete time slots for scheduling to ensure accuracy, latency, and to lower the chance of losing computation results due to changes in connectivity. This computation will be based on an effective light-weight algorithm elaborated in Section. IV-D.

IV. Problem Formulation

As mentioned in previous section, we assume that a supernode with enough capacity is present at each base station covering the D2D network of our interest. At the beginning of each task period, the supernode will, with the knowledge of all devices and tasks in the D2D network, calculate the probability distribution of the connectivities between devices, and assign computational tasks to each helper or requester device accordingly. Connectivity-awareness, computational task assignment, response delay optimization, are mathematically formulated in this section.

A. Connectivity Awareness

To analyze the connectivity between requesters and helpers for more accurate and cost-effective computation assignment, we first define a $p \times (h+1)$ probability matrix R_t at each time slot $t \in \{0, 1, ..., \frac{T}{\Delta t}\}$. For a given t, the element $R_{i,j,t}$ is the probability of connection between p_i and h_j if $j \in \{1, 2, ..., h\}$, and $R_{i,(h+1),t}$ indicating the probability of connection between p_i and herself/himself. Obviously, $R_{i,(h+1),t} = 1 \; \forall \; i, t$. At the start of each task period, i.e. $t = 0$, we randomly generate $R_{i,j,0} \; \forall \; i \in \{1, 2, ..., p\}, j \in \{1, 2, ..., h\}$ based on the pre-defined initial connection probability. Thereafter, according to equation (6), we generate $R_{i,j,t} \; \forall t \in \{1, 2, ..., \frac{T}{\Delta t}\}$ for each p_i-h_j pair.

At any given time slot $t \in \{1, 2, ..., \frac{T}{\Delta t}\}$, we define a p-element vector $\vec{I_t}$ with its element $\vec{I}_{i,t}$ representing the amount of self-computing computational task assigned to p_i, and a $p \times h$ matrix J_t with element $J_{i,j,t}$ representing the amount of assisting computational task assigned to h_j for p_i. By concatenating $J_{i,j,t}$ and $\vec{I}_{i,t}$, we get a $p \times (h+1)$ computation assignment matrix $M_t = [J_{i,j,t} \; \vec{I}_{i,t}]$ at time slot t. Joining all M_t where $t \in \{1, 2, ..., \frac{T}{\Delta t}\}$, we get a $p \times (h+1) \times \frac{T}{\Delta t}$ 3-dimensional system computation assignment matrix, M, containing the computation assignment in a task period $\tau \in [0, T]$. Consequently, $M_{i,j,t}$ where $j \in \{1, 2, ...h\}$ is the amount of computational task assigned to h_j for p_i and $M_{i,h+1,t}$ is the amount of self-computing computational task assigned to p_i at time slot $\tau = t$.

Note that the element-wise product between R and M,

$$M^{exp} = M \odot R, \quad (7)$$

is the matrix indicating the expected amount of computational task done and result sent back to requesters. For example, $M^{exp}_{i,j,t} = M_{i,j,t} \cdot R_{i,j,t}$ is the expected amount of assisting task done by h_j and sent back to p_j at time slot $\tau = t$.

B. Computation Assignment & Maximum Wait Time

At the start of each task period, all requesters will specify to the supernode at the base station the amount of computation, in clock cycles, required for the coming task period. We represent these tasks with a p-element vector $\vec{\gamma}$ with γ_i corresponding to the total amount of task required by p_i. Meanwhile, for an ensured QoE, p_i is also subject to a maximum wait time in each task period for the result of the computational task. We use another p-element vector $\vec{\phi}$ with ϕ_i corresponding to the maximum wait time for p_i in seconds. Apparently, $\phi_i \leq T, \; \forall i$. Therefore, the computation assignment needs to ensure that a task is expected to be completed before the maximum wait time for all requesters, that is:

$$\gamma_i \leq \sum_{t=1}^{\phi_i/t} \sum_{j=1}^{(h+1)} M^{exp}_{i,j,t}, \; \forall i \in \{1, 2, ..., p\} \quad (8)$$

C. Computation Capacity of Mobile Devices

Each mobile device is subject to a computation capacity denoted as c^p_i for p_i's mobile device and c^h_j for h_j's mobile device where $i \in \{1, 2, ..., p\}, j \in \{1, 2, ..., h\}$. When talking about computation capacity of a device, one typically will refer to the CPU of it. CPU processing capacity is typically referred to in terms of Megahertz (MHz) or Gigahertz (GHz). Professionals talk about clock speed, which is the standard ability of the CPU to cycle through its operations over time. Therefore, a 1 GHz CPU is able to tick its clock around 1 billion times per second, which in turn can perform more complicated computational tasks. Without loss of generality, we regulate these computational power values in clock cycles to a scale of 0 to 100, i.e. $0 \leq c_i^p \leq 100$ and $0 \leq c_j^h \leq 100 \; \forall \; i, j$, for simplicity. Each entry of the computation assignment matrix, $M_{i,j,t}$, refers to the number of clock cycles required to perform the corresponding task section. For example, $M_{3,4,0} = 1000$ means that at $t = 0$, h_4 is assigned to help p_3 for a 1000 clock cycles worth of computational task. If $c_4^h = 50$Hz, then it takes h_4 $\frac{M_{3,4,0}}{c_j^h} = \frac{1000}{50\text{Hz}} = 20s$ to perform the task. Therefore, each device is subject to an amount of computational task in clock cycles at each time slot as a higher limit, that is:

$$\sum_{i=1}^{p} M_{i,j,t} \leq c_j^h, \; \forall j \in \{1, 2, ..., h\}, t \in \{1, 2, ..., \frac{T}{\Delta t}\} \quad (9)$$

for all helper devices, and

$$M_{i,h+1,t} \leq c_i^p, \; \forall i \in \{1, 2, ..., p\}, t \in \{1, 2, ..., \frac{T}{\Delta t}\} \quad (10)$$

for all requester devices as the $(h+1)$th column of the computation assignment matrix are representing the amount of self-computing tasks.

D. Response Delay Optimization

In our work, we emphasize the importance of low task execution time towards a requester's QoE. In each task period, there is $n = \frac{T}{\Delta t}$ time slots and we define the expected completion time, t_i^{exp} for each requester p_i as follows:

$$t_i^{exp} = \arg\min_t \sum_{\tau=1}^{t} \sum_{j=1}^{(h+1)} M_{i,j,t}^{exp} \geq \gamma_i, \quad (11)$$

where $i \in \{1, 2, ..., p\}, j \in \{1, 2, ..., h\}, t \in \{1, 2, ..., \frac{T}{\Delta t}\}$. Finally, we derive the objective function as follows:

$$\begin{aligned} \text{Minimize:} \quad & \sum_{i=1}^{p} t_i^{exp} \\ \text{Subject to:} \quad & (7)(8)(9)(10)(11). \end{aligned} \quad (12)$$

Calculation of the optimal computation assignment matrix M is similar to the famous knapsack problems that is NP-hard [25]. Here, the computational power of helper devices are like knapsacks with different sizes, and the computational tasks from requesters are like items with different sizes. Solving for the optimal solution for the task scheduling resembles solving for an optimal solution for knapsack problem: it is NP-hard. For effectiveness, especially considering the trend of increasingly massive number of smart mobile devices in D2D networks, we proposed a light-weight heuristic algorithm to efficiently find the sub-optimal solution for the computation assignment matrix M illustrated in Algorithm 1. Note that we make substantial use of the *linprog* function in *MATLAB* for calculating the computation assignment matrix M by transforming M element-wise into a one-dimensional unknown vector $\vec{x}$ with $p \cdot (h+1) \cdot \frac{T}{\Delta t}$ entries from elements in M. According to *linprog*, A, b in Algorithm 1 represent the inequality constrains for $\vec{x}$, and A_{eq}, b_{eq} represent the equality constrains for $\vec{x}$. There is no upper bound for $\vec{x}$, and the lower limit for elements of $\vec{x}$ is 0.

Algorithm 1: The algorithm to schedule task cooperation for $i \in \mathcal{U}$ uses to obtain $\mathcal{K}_{j,i}^t$ from user $j \in \mathcal{U} \backslash \{i\}$.

```
Initialize $\vec{\phi}$.
$[A, b, A_{eq}, b_{eq}]$ := get_linprog_parameter($\vec{\phi}$)
$[x, feasible]$ := linprog($\vec{0}, A, b, A_{eq}, b_{eq}, \vec{0}$)
if feasible then
    $\mathcal{C} := \{1, 2, ..., p\}$
    Loop
        while $\mathcal{C} \neq \emptyset$ do
            for $i \in \mathcal{C}$ do
                $temp := \phi_i$.
                $\phi_i := \lfloor \frac{\phi_i}{2} \rfloor$.
                $[A, b, A_{eq}, b_{eq}]$ :=
                  get_linprog_parameter($\vec{\phi}$).
                $[x, feasible]$ := linprog($\vec{0}, A, b, A_{eq}, b_{eq}, \vec{0}$).
                if !feasible then
                    $\phi_i := temp$.
                    $\mathcal{C} := \mathcal{C} \backslash \{i\}$.
```

TABLE I
DEFAULT SIMULATION PARAMETERS

Number of requesters p	6
Number of helpers h	10
Tasks period T	60s
Size of time slots Δt	5s
Maximum computation capacity $c^{\max}$	100
Minimum computation capacity $c^{\min}$	30
Computation capacity c_i^p or c_j^h	$U[c^{\min}, c^{\max}]$
Mean task size per second σ	50 ($U[25, 75]$)
Mean maximum wait time	40s ($U[20\text{s}, 60\text{s}]$)
$\lambda_{i,j}$	$U[10^{-5}, 10^{-3}]$
$\mu_{i,j}$	$U[10^{-3}, 10^{-2}]$
Initial connection probability at $\tau = 0$	50%

V. EXPERIMENT

In a D2D network, a variety of factors may affect the performance of our cooperative system and proposed algorithm. In this section, we will examine the following effects:

- *Effect of number of requesters:* we fix the number of helpers and vary the number of requesters to see the performance change of our system.
- *Effect of number of helpers:* we fix the number of requesters and vary the number of helpers to see the performance change of our system.
- *Effect of mean maximum wait time:* we vary the mean maximum wait time during the random generation to see how the performance will be affected.

- *Effect of mean task size:* we vary the mean task size of each requester in a task period during the random generation to see how the performance will be affected.

In our simulation, we compare our proposed cooperative task scheduling scheme to the basic no-cooperation case and a greedy case:

- *Basic non-cooperation:* no D2D cooperation is performed during the simulation, the task execution is done solely on each requester device itself.
- *Greedy D2D task cooperation:* without connectivity-awareness, each helper device will equally contribute its available computing power to all connecting requesters at the beginning of a task period. For example, helper h_5 will assign $\frac{c_5^h}{3}$ computing power to each of p_1, p_3, p_4 for the current task period if and only if p_1, p_3, p_4 are the only requesters in connection with h_5 at time $\tau = 0$.
- *Connectivity-aware task scheduling:* at the start of each task period, the supernode at the base station will perform task scheduling calculation according to Algorithm 1.

To validate the performance of our proposed system, we set up the following experiment. Default simulation parameters are illustrated in TABLE I, where uniform distribution between two values a, b is denoted as $U[a, b]$.

A. Effect of number of requesters in the system

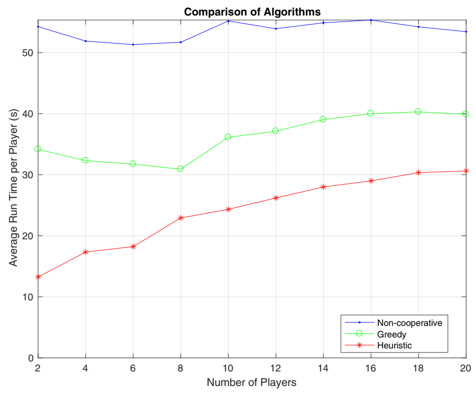
When there is a fixed number of helpers in the D2D network, the number of requesters becomes an important factor affecting our system. Imagine there are much more requesters in the network than helpers, then each helper may be assigned to assist many requesters at the same task period. On contrast, if there are only a few requesters in the system, many helpers may assist only one requester at the same time, which greatly reduces the average task execution time for the requesters.

As illustrated in Fig. 3(a), the average task execution time per player is only 25% of that in the non-cooperative case, and 40% of that in the greedy case. The increase in number of requesters degrades our system in that the average task execution time for requesters is increased. However, as shown in Fig. 3(b), the effect of increase in number of requesters also itself degrades and levels off when the number of requesters is around 14 (note that the number of helpers here is 10). In comparison, number of players seems to have limited effects on non-cooperative and greedy cases. In the greedy case, the average task execution time changed from 34s when $p = 2$ to 40s when $p = 20$. Normalized speed of the greedy D2D cooperation is always around 50% better than non-cooperative case, compared to 75% to 400% faster for our heuristic algorithm.

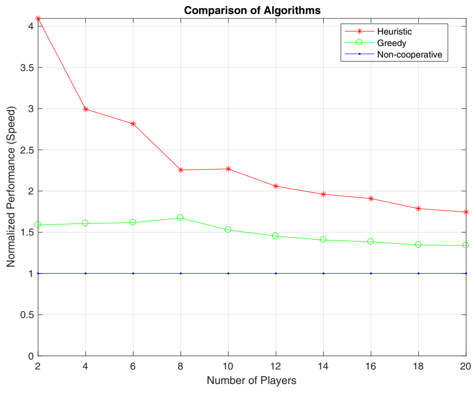
B. Effect of number of helpers in the system

Similarly, when there is a fixed number of requesters in the D2D network, the number of helpers in turn becomes an important factor affecting our system. The effect of increase in number of helpers in the system is shown in Fig. 4.

As shown, the increase in helper upgrades our system in that the average task execution time over all requesters is decreased for both heuristic and greedy algorithms. Fig. 4(a) shows that initially, when $h = 2$, our system with scheduling based on the heuristic algorithm takes only 58% and 68% of the average task execution time for the non-cooperative and greedy cases respectively. While the increase in number of helpers decreases the average execution time for the greedy algorithm, the decreasing effect start to fade out when the number of helpers increases beyond $h = 10$. However, the average execution time for our connectivity-aware task scheduling base on the heuristic algorithm continue to remaining decreasing all the way, reaching to only around 20% of the non-cooperative case. Fig. 4(b) illustrates the normalized performance of the greedy and heuristic algorithm with respect to the non-cooperative case. Though the performance of the greedy task cooperation remains level beyond $h = 10$, cooperative task execution speed of our connectivity-aware scheduling continue to enhance as the number of helpers grows up with a close-to-linear relationship. Note that the average execution speed for the heuristically calculated task scheduling for the 6 requesters increased by a factor of 200% when we increase the number of helpers from $h = 10$ (1.67 times more than number of requesters) to $h = 20$ (3.33 times more than number of requesters).

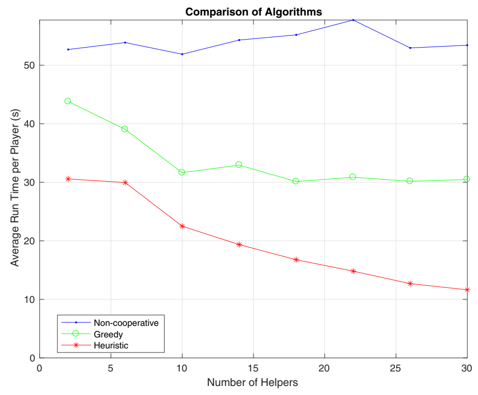


(a) Effect on Average Task Execution Time

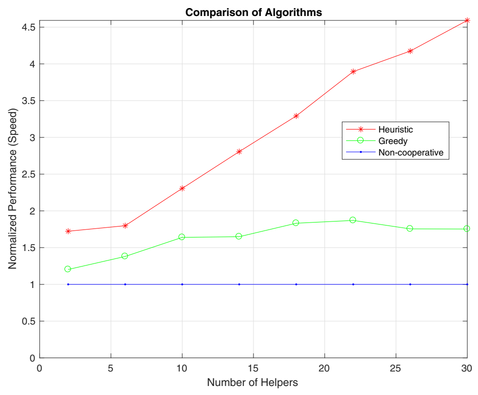


(b) Normalized Performance

Fig. 3. Effect of Number of Requesters

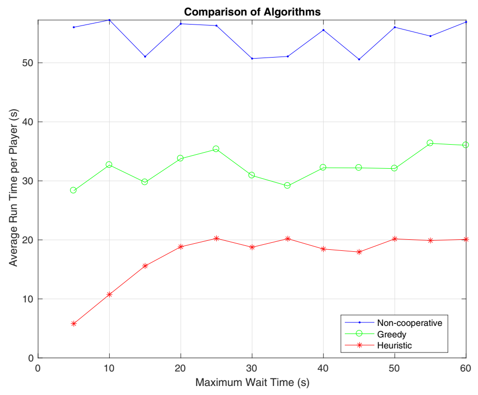


(a) Effect on Average Task Execution Time

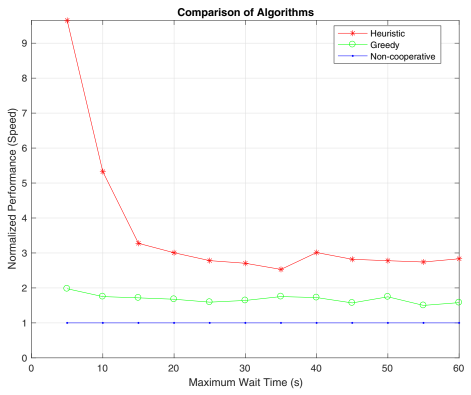


(b) Normalized Performance

Fig. 4. Effect of Number of Helpers



(a) Effect on Average Task Execution Time



(b) Normalized Performance

Fig. 5. Effect of Maximum Wait Time

C. *Effect of mean maximum wait time of requesters*

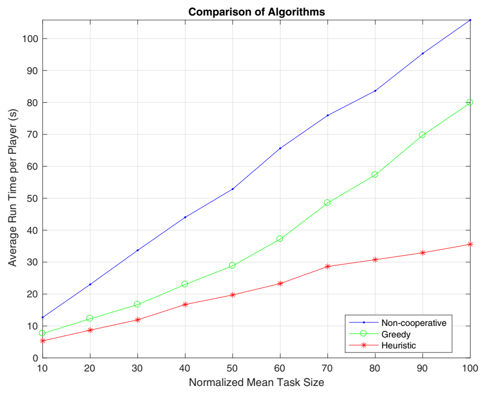
During a task cooperation in the D2D network, the requester typically wants the task to be done in a timely manner. For example, if a requester wishes to perform a neural network based stock index prediction task [26] that predicts the price of a stock in 30 seconds, this requester will want to get the computation result within 30 seconds (could be 20 seconds, 25 seconds, etc. depending on the specific application logic and handling behind).

As illustrated in Fig. 5, the performance of the greedy cooperation algorithm and non-cooperative task execution is not noticeably affected the maximum wait time of requesters. In contrast, average task execution time is around $\frac{1}{3}$ that of the non-cooperative case and around $\frac{2}{3}$ that of the greedy case when the mean maximum wait time is beyond 20 seconds. It is worth noticing that the average task execution time is lowered to approximately only 10% that of the non-cooperative case when mean maximum wait time is reduced to 5 seconds. The reason behind is that the number of feasible solutions decreases as the mean maximum wait time decreases. When the mean maximum wait time of requesters is very short, e.g. 5 seconds in Fig. 5, the number of feasible solutions becomes very small, and any feasible solution becomes very close to the optimal solution. Therefore, shorter mean maximum wait time generally allows our proposed heuristic algorithm to find out better scheduling for task cooperation in the D2D network.

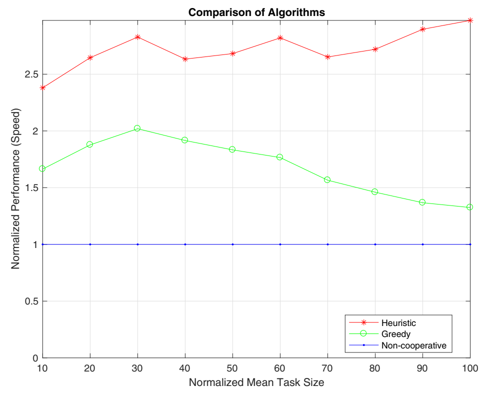
D. *Effect of mean task size of requesters*

Finally, we investigated into the effect of change in mean task size to our proposed system. The generation of task sizes in a period T is a uniform distribution $U[0.5\sigma T, 1.5\sigma T]$. Note that it is possible that the task size for a requester within a period is over the computing capacity of the requester device itself. Therefore, a D2D cooperation is required if the task needs to be done within the maximum wait time.

From Fig. 6(a), we can see that the average task execution time is approximately proportional to the mean task size for all three compared cases. In other words, a requester who wishes to execute a more complex computing task, e.g. neural network based stock index prediction, that requires higher amount of computation will need more time to cooperatively execute the tasks in the D2D network than other requesters who only need to perform simple computational tasks, e.g. simple image processing to shrink the file size of an image, that require much less amount of computation. Note that the average task execution time for non-cooperative case is beyond the duration



(a) Effect on Average Task Execution Time



(b) Normalized Performance

Fig. 6. Effect of Mean Task Size

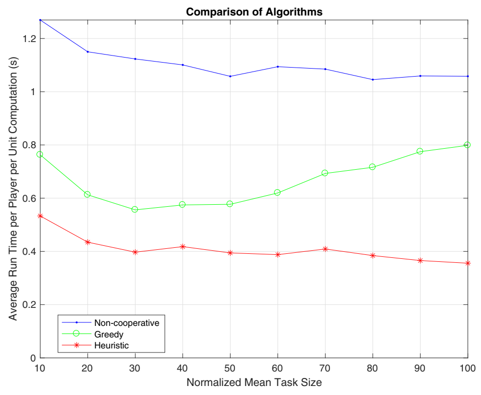


Fig. 7. Effect of Mean Task Size on Average Task Execution Time per Player per Unit Computation

of a task period (60s) when the mean task size is beyond 50, meaning that these tasks are typically not completed by their maximum wait time. Average task execution time for the greedy cooperation case is better but is still beyond the duration of a task period when mean task size grow beyond 80. Average task execution time for our heuristic case is increasing at a much slower path and remains well under 40s even when the mean task size in increased to 100.

Normalized performance in Fig. 6(b) illustrates that the normalized average execution speed of our heuristically scheduled task cooperation experienced a 25% performance increase when the mean task size is increased from 10 to 100. In comparison, the normalized execution speed of the greedy case experienced a constant decrease when the mean task size increase from 30 beyond, from a 200% faster speed than the non-cooperative case to only 30%. The normalized execution speed of the heuristic algorithm based scheduling, however, is 300% faster than the non-cooperative case when the mean task size is 100.

If we look at the average task execution time per unit of computation, as illustrated in Fig. 7, we can see that as the mean task size increases, the average execution time required per unit of computation actually decreases a little and remains stable overall in our proposed heuristic algorithm based task scheduling. This is an indication that, in terms of execution time per unit of computation, the performance of our cooperative task scheduling is quite stable.

VI. Discussion and Future Work

There are a few limitations in our work:

- *Dynamic task slicing in an arbitrary manner:* In our work, we adopted the assumption that the original computational task to be performed cooperatively in the D2D network can be sliced into arbitrary pieces. Even though it is unrealistic to partition a computational task into pieces with an arbitrary size, it is generally feasible to slice a reasonably-sized task into many pieces making use of a program slicing technique like that proposed in [23] or [24]. For simplicity, we just assume a computation piece of an exact portion, e.g. 33%, of the original task is available.
- *Performance of heuristic algorithm with respect to optimal solution:* As explained in Section V-C, the performance of our heuristic task scheduling we propose is suboptimal, and is closer to the optimal performance only when mean maximum wait time among all players are small enough. The reason for adoption of our heuristic algorithm for computation of task scheduling is that calculation of the optimal solution is NP-hard especially if we have more time slots within a task period. To relieve the burden of computation time for task scheduling, we have to sacrifice some performance over the average task execution time. In our future work, we will take success rate into account and transform our problem in this work to an optimization problem over the success rate, which is convex and optimal solution can be achieved within polynomial time.
- *Fairness among requesters and helpers:* It feels intuitively unfair for a helper to assist requesters in the D2D network without being assisted in one period. However, this helper in another period may in turn become a re-

quester and is substantially assisted for task cooperation. To guarantee fairness, we will add in a credit model in the system so that a helper can gain task cooperation credits or cash credits after assisting other requesters. The helper can use these credits to ask for task assistance if needed in the future.

VII. Conclusion

We presented a connectivity-aware D2D-based mobile task outsourcing and scheduling paradigm in this paper. A supernode at the base station, with knowledge of user mobilities and thus probability model of device connectivities, will perform task scheduling to reduce average task execution time for requesters in the network to enhance user quality of experience. Simulation results based on realistically examined mobility model show that our system substantially reduces average task execution time for requesters in the D2D network. When mean maximum wait time is as low as 5 seconds, the average task execution time per player is only around 10% that of the non-cooperative case. The performance of our system also is approximately directly proportional to the number of helper devices in the D2D network, which is increasingly desirable as the number of smart devices is rapidly increasing for potential applications in the area of Internet of Things.

Acknowledgment

This work is supported by funding from the Natural Sciences and Engineering Research Council of Canada under grant STPGP 447524.

References

[1] Y.-M. Tai and Y.-C. Ku, "Will stock investors use mobile stock trading? a benefit-risk assessment based on a modified utaut model," *Journal of Electronic Commerce Research*, vol. 14, no. 1, pp. 67–84, 2013. [Online]. Available: http://ezproxy.library.ubc.ca/login?url=http://search.proquest.com.ezproxy.library.ubc.ca/docview/1372332718?accountid=14656

[2] W. Cai, V. C. M. Leung, and M. Chen, "Next generation mobile cloud gaming," in *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, March 2013, pp. 551–560.

[3] PlayStation Now, https://www.playstation.com/en-ca, online.

[4] GameFly, https://www.gamefly.com/, online.

[5] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15. New York, NY, USA: ACM, 2015, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/2757384.2757397

[6] Dave Chaffey, "Mobile marketing statistics compilation," http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/?new=1, online.

[7] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '12. New York, NY, USA: ACM, 2012, pp. 145–154. [Online]. Available: http://doi.acm.org/10.1145/2248371.2248394

[8] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proceedings of the 19th Annual International Conference on Mobile Computing &#38; Networking*, ser. MobiCom '13. New York, NY, USA: ACM, 2013, pp. 203–205. [Online]. Available: http://doi.acm.org/10.1145/2500423.2504576

[9] C. Shi, M. H. Ammar, E. W. Zegura, and M. Naik, "Computing in cirrus clouds: The challenge of intermittent connectivity," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 23–28. [Online]. Available: http://doi.acm.org.ezproxy.library.ubc.ca/10.1145/2342509.2342515

[10] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 255–270. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814459

[11] A. Y. Ding, B. Han, Y. Xiao, P. Hui, A. Srinivasan, M. Kojo, and S. Tarkoma, "Enabling energy-aware collaborative mobile data offloading for smartphones," in *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, June 2013, pp. 487–495.

[12] F. Mehmeti and T. Spyropoulos, "Is it worth to be patient? analysis and optimization of delayed mobile data offloading," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 2364–2372.

[13] X. Ge, X. Li, H. Jin, J. Cheng, and V. C. M. Leung, "Joint user association and scheduling for load balancing in heterogeneous networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[14] X. Ge, H. Jin, and V. C. M. Leung, "Opportunistic downlink scheduling with resource-based fairness and feedback reduction in distributed antenna systems," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 7, pp. 5007–5021, July 2016.

[15] A. Asadi and V. Mancuso, "Wifi direct and lte d2d in action," in *2013 IFIP Wireless Days (WD)*, Nov 2013, pp. 1–8.

[16] C. A. Chen, M. Won, R. Stoleru, and G. G. Xie, "Energy-efficient fault-tolerant data storage and processing in mobile cloud," *IEEE Transactions on Cloud Computing*, vol. 3, no. 1, pp. 28–41, Jan 2015.

[17] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5g," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 74–80, February 2014.

[18] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C. J. Wu, "A study of mobile device utilization," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2015, pp. 225–234.

[19] F. Chi, X. Wang, W. Cai, and V. C. M. Leung, "Ad hoc cloudlet based cooperative cloud gaming," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, Dec 2014, pp. 190–197.

[20] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[21] Z. Wang, H. Shah-Mansouri, and V. Wong, "How to download more data from neighbors? a metric for d2d data offloading opportunity," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[22] J. R. Norris, *Markov chains*. Cambridge university press, 1998, no. 2.

[23] R. Gerber and S. Hong, "Slicing real-time programs for enhanced schedulability," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 3, pp. 525–555, May 1997. [Online]. Available: http://doi.acm.org/10.1145/256167.256394

[24] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org.ezproxy.library.ubc.ca/10.1145/1327452.1327492

[25] H. Kellerer, U. Pferschy, and D. Pisinger, *Introduction to NP-Completeness of Knapsack Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 483–493. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24777-7_16

[26] P.-C. Chang, C.-H. Liu, J.-L. Lin, C.-Y. Fan, and C. S. Ng, "A neural network with a case based dynamic window for stock trading prediction," *Expert Systems with Applications*, vol. 36, no. 3, Part 2, pp. 6889 – 6898, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417408006209