Have You Asked Your Neighbors? A Hidden Market Approach for Device-to-Device Offloading

(Invited Paper)

Dimitris Chatzopoulos*, Mahdieh Ahmadi*†, Sokol Kosta*‡, Pan Hui*
{dcab@cse.ust.hk, mahmadi@ce.sharif.edu, kosta@di.uniroma1.it, panhui@cse.ust.hk}
*The Hong Kong University of Science and Technology
†Sharif University of Technology
‡Sapienza University of Rome

***Abstract*—During the last years, researchers have proposed solutions to help smartphones offload heavy tasks to remote entities in order to improve execution time and reduce energy consumption. Lately, inspired by the promising results of message forwarding in opportunistic networks, many researchers have proposed strategies for task offloading towards nearby mobile devices. None of these strategies, though, proposes any mechanism that considers selfish users and, most importantly, that motivates and defrays the participating devices who spend their resources. In this paper, we address these problems and propose the design of a framework that integrates an incentive scheme and a reputation mechanism. Our proposal follows the principles of the Hidden Market Design approach, which allows users to specify the amount of resources they are willing to "sacrifice" when participating in the offloading system. The underlying algorithm, that users are not aware of, is based on a truthful auction strategy and a peer-to-peer reputation exchange scheme. Extensive simulations on real traces depict how our designed mechanism achieves higher offloading rate and produces less traffic compared to three benchmark algorithms. Finally, we show how collaborating devices get rewarded for their contribution, while selfish ones get sidelined by others.**

I. Introduction

Nowadays, we use smartphones to do most of the tasks that we used to do on traditional computers, and many more. Their capabilities do not stop increasing, with more memory, more powerful CPUs, and more sensors every year. At the same time though, developers continue building even more hungry applications in terms of computational requirements and energy consumption. To cope with these problems, *Computation Offloading* is seen as a solution for overcoming the restrictions imposed by applications on outdated devices [1], [2], [3], [4].

More recently, inspired by the success of message forwarding in Delay Tolerant Networks (DTN) [5], researchers have proposed computation offloading solutions that rely solely on nearby mobile devices, a technique known as device–to–device (D2D) offloading [6], [7], [8], [9], [10]. These works show that not only is D2D offloading possible, but is also beneficial in terms of energy consumption and execution time for tasks that can tolerate small delays [11]. However, the D2D architecture poses many challenges, which are related to the intermittent contacts between devices due to user mobility and to the inherent human selfish behavior. Researchers have found that even though users are dynamic, they tend to be repetitive; visiting the same places and meeting the same people most of the time, so the chances for two people to meet more than once, even though intermittently and with unpredictable duration, are high. Indeed, many previous works on device–to–device offloading have shown that human mobility does not pose serious problems when offloaded tasks tolerate small delays [7], [8], [9], [10].

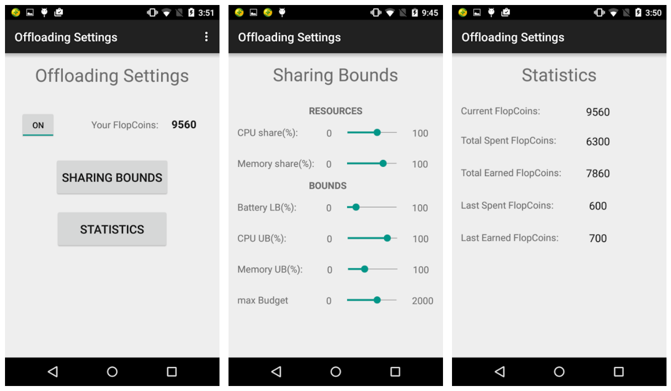
To the best of our knowledge though, none of these previous works on D2D offloading considers the case when users are not willing to collaborate. We believe that users will be cautious in installing and using a D2D offloading framework considering that executing tasks for other users implies high energy consumption. Not only that, considering the selfish human nature [12], every user would be interested in extending his battery life by *i)* trying to offload as many tasks as possible to others and *ii)* not accepting offloading requests from other users. For this reason, we believe that is extremely necessary to introduce *incentive schemes* and *reputation mechanisms* in the device–to–device offloading architecture [13]. Incentive schemes are important for rewarding collaborating users, while reputation mechanisms are needed to keep track of the reliability of the users.

In this paper we propose the design of a D2D offloading framework that integrates a distributed incentive scheme and a distributed reputation mechanism. The core modules of our proposal are: a D2D Offloading module, a Neighbor Selection module, an Incentive Scheme module, a Reputation Mechanism, and a Graphical User Interface. The *D2D Offloading* module is responsible for performing the actual task offloading between devices. In this work we do not focus on this component, since many previous works have already designed and implemented solid offloading systems, and we assume that it can be replaced by any of the previously proposed systems. Our contributions in this paper are all in the remaining modules. First, the *Neighbor Selection* module handles the communication with the nearby devices. This module implements two algorithms, which take their decisions based on the information provided by the *Incentive Scheme* and the *Reputation Mechanism.*

The incentive scheme uses a virtual currency, named *FlopCoin*, to compensate devices whenever they execute an offloadable task. The underlying algorithm is based on a truthful bidding strategy: Whenever a device receives a request for executing an offloadable task, the algorithm calculates the amount of FlopCoins the requesting device should pay for the service. This amount is determined by the resource needs of the offloadable task. After receiving all the bids from her neighbors, the requester device selects the most suitable for offloading, if any, and passes the information to the neighbor selection module. The reputation mechanism is used by each device to evaluate and remember the "goodness" of the others, and is an indication about the reliability of a peer to provide the result of an offloaded task.

Lastly, we introduce a component that allows users to select the amount of resources they are willing to share. We design this component and the overall offloading framework following the principles of the Hidden Market Design [14], which states that: *i)* the complexities of the system must be hidden to the final user and *ii)* the user interface experience must be seamless. Based on these principles, we have implemented a User Interface (UI) for Android, which is presented in Figure 1. The first image shows the main Android activity that the user sees when she opens the framework's settings. From this activity the user can activate or deactivate the service, can check the FlopCoin budget, or can open the next activities to set her preferences. Using the middle activity she can select the resources she is willing to share and enforce the lower bounds for each resource. For example, if the battery lower bound is set to 20%, the device stops accepting tasks for execution from the others when this battery level is reached. This work tries to establish the basis towards a device–to–device offloading framework in a realistic scenario, where the majority of the users would attempt to obtain free service from the others while limiting their contribution. In the formulation and the analysis of our proposal we assume that users can only control these bounds and activate/deactivate the service.

We evaluate our proposed architecture through extensive simulations on three real mobility traces and compare it with three other strategies: a ***Just-in-Time***, a ***Look-Ahead***, and the ***Tit-For-Tat*** strategy (TFT) [15]. We show that compared to the other algorithms, our strategy achieves: (*i*) higher rate of successfully offloaded tasks over the total tasks; (*ii*) lower traffic generated in the system; (*iii*) sidelining of the selfish users, due to the fact that they spend their FlopCoins and their reputation diminishes as they don't accept any task from the others; and (*iv*) rewarding of altruist users by increasing their FlopCoins and their reputation.



Your FlopCoins: 9560

CPU share(%): 0 100
Memory share(%): 0 100
Battery LB(%): 0 100
CPU UB(%): 0 100
Memory UB(%): 0 100
max Budget 0 2000

Current FlopCoins: 9560
Total Spent FlopCoins: 6300
Total Earned FlopCoins: 7860
Last Spent FlopCoins: 600
Last Earned FlopCoins: 700

Fig. 1: User interface of resource sharing configuration following the principles of the Hidden Market Design.

II. Related Work

Cloud computing enabled computation offloading from conventional devices to powerful cloud servers. Many offloading frameworks run Virtual Machine (VM) surrogates of mobile devices in the cloud and use them for computation offloading [1], [2], [3]. In more recent works, proximal resource-rich devices—like computers, access points, tablets, smartphones, etc.—are preferred for offloading instead of the distant cloud [4], [6], [7], [16], [17], [18]. The lower communication delay, lower packet loss, and higher bandwidth between the mobile and the proximal device make this solution more suitable than the public cloud in some circumstances.

However, as in previous solutions, they consider the case where all devices behave correctly and are willing to execute the same application. A common characteristic of the aforementioned proposals is that none of them considers human selfish behavior in the selection of the nearby device. All these works focus on the technical details of the computation offloading and assume that any nearby device, once selected for offloading, will execute the task and deliver the result.

In the direction of motivating people to cooperate in mobile ad hoc networks, Anderegg et al. propose a payment based mechanism [19], which is controlled by a central authority that assigns tokens to nodes that forward messages, based on their importance in the path. Michiardi et al. provide a reputation system where a central authority keeps a record for everyone's cooperative behavior [20]. The drawback of both [19] and [20] is their assumption that a central authority has direct access to the mobile nodes and is aware of the whole system.

In this paper we focus on the problem of selecting the most appropriate mobile devices for task offloading in a proximal device–to–device offloading scenario and incentivize mobile users to share their resources when they do not use them. Advancing on previous approaches, our proposed mechanism **(i)** does not take for granted that users will be willing to cooperate whenever some task is given to them for execution and **(ii)** is decentralized.

III. Problem Formulation

We consider a set of mobile users[1] $\mathcal{N}$ in the context of Delay Tolerant Networks who collaborate by using each others' resources for application offloading. We assume that the users are self interest in the way they tune their resource sharing bounds. A mobile application $\mathcal{A}$ is composed by tasks, has an execution duration parameter $\mathcal{A}_\tau$, and has a maximum allowed execution delay τ; with $\tau > \mathcal{A}_\tau$. $\mathcal{A}_\tau$ depends on the device's available resources and differs from device to device. Let's assume that at time t user i wants to execute an application $\mathcal{A}$. She can either execute $\mathcal{A}$ locally and have the result after $\mathcal{A}_\tau$ seconds or ask for help from her neighbors. We define $\mathcal{N}_i(t)$ as the set of neighbours in contact with user i at time t.

In case of offloading, user i will try to send parts of $\mathcal{A}$ to her neighbors in $\mathcal{N}_i(t)$ trying to satisfy the constraint about the maximum allowed delay τ. To this end, each user uses a dynamic reputation mechanism for calculating the probability θ_n that neighbor n will send the result of the task before $\tau - \mathcal{A}_\tau$ (this is the last useful moment for user i to execute $\mathcal{A}$ locally before the deadline τ). The probability function θ follows a beta distribution $Beta(\alpha, \beta)$, where α and β are updated after each offloading interaction. In case of a successful offloading towards neighbor n, user i increases α_n so that n's reputation increases, otherwise he increases β_n so that n's reputation decreases. The initial θ distribution is uniform, so no user in particular is preferred. To allow users' explicitly specify their trust towards known participants, we introduce another variable $F_i(n)$ that represents the trust of user i towards n. In Section VII we explain how θ_n and $F_i(n)$ are calculated.

A. Extra Overhead of Application Execution

Both Android and iOS use object oriented programming languages for development—Java and Objective C, respectively. In the context of mobile code offloading, an application $\mathcal{A}$ can be seen as the union of two sets of classes: $\mathcal{A} = \mathcal{A}^o \cup \bar{\mathcal{A}}^o = \{c_1, \ldots, c_L\}$, where $\mathcal{A}^o$ is the set of offloadable classes, $\bar{\mathcal{A}}^o$ is the set of non-offloadable classes, and c_i denotes a single class. Each class c represents a task that has a duration τ_c and consumes r_c resources when executed. When a device receives a task from another node, additional resources will be consumed. These costs can be expressed as a function of the needed resources as well as the network overhead. In our model we assume a function, shown in (1), that takes care of calculating these costs based on a metric μ. The metric can depend on the energy needs, the delay, the quality of experience of the end user or anything the application developer had imposed. The following notation is used: $\mathbf{c} \subset \mathcal{A}^o$ is the a set of classes and $b_{xy}(t)$ and $l_{xy}(t)$ are the bandwidth and the latency from node x to node y at time t, while n is the device that will execute the offloaded classes.

[1]In the rest of the paper we use the terms user, device, and node interchangeably

$$f_n^\mu(\mathbf{c}, b_{in}(t), b_{ni}(t), l_{in}(t), l_{ni}(t)) \longrightarrow \mathcal{R}^+ \qquad (1)$$

Given $\mathcal{A}^o$, we use the following notation to indicate if one class is offloaded to one node: $x_c^n = \{0,1\}$. $x_c^n = 1$ if the class c is offloaded to neighbor n and 0 otherwise. Users will decide to offload one task only if it is beneficial for them. Moreover, the same task can be offloaded to more than one neighbors. Using the previous notation, we can express these constraints using the following: $\sum_{n \in \mathcal{N}_i(t)} x_c^n \geq 0$. We express the replies of the offloaded tasks from the helper nodes: $y_c^n = \{0,1\}$. $y_c^n = 1$ if the task c output received from user n and 0 otherwise. After offloading part of $\mathcal{A}$, the aided device waits for

$$\tau - \mathcal{A}_\tau + \sum_{c \in \mathcal{A}^o} (\tau_c) \mathbf{1}_{\sum_{n \in \mathcal{N}_i(t)} y_c^n > 1} \qquad (2)$$

seconds, where τ_c is the execution time of task c, to receive the results of the offloaded tasks. The third factor of the above equation is used to increase the tolerance of user i and make him extend the waiting time in case some of the offloaded tasks are already finished. The probability of successfully receiving the result of one offloaded task c before the maximum allowed deadline can be expressed using the following formula:

$$\zeta_c \quad = \quad Pr\left[\sum_{n: x_c^n = 1} y_c^n \geq 1\right] = 1 - \prod_{n: x_c^n = 1} (1 - \theta_n) \quad (3)$$

where θ_n is the probability that node n, who accepted to execute the task, will return the result of the task in time.

IV. Incentive Scheme

We are dealing with a scenario where users asynchronously exchange resources. To measure how much each user is participating, we introduce the concept of a virtual currency called *FlopCoin*. Whenever a user is registered in the system is getting an initial amount of free FlopCoins. These FlopCoins are signed by the system provider in the form of a transaction. Every user i keeps a transactions list $\mathcal{T}_i$, where she stores every transaction she participated or she eyed. Whenever one user needs to offload a task, she runs an auction to ask for offers by any possible helper. We use a modified truthful mechanism for the biding process that is associated with an inverse second price auction [21]. Following this strategy, each neighbor bids her true cost of the task execution and if she is selected, she will be paid at least as much as the next biggest bid.

Bid example: Three neighbors n_1, n_2, and n_3 bid for executing one specific task c, with bid values $\gamma_c^{n_1}$, $\gamma_c^{n_2}$, and $\gamma_c^{n_3}$ such that $\gamma_c^{n_1} < \gamma_c^{n_2} < \gamma_c^{n_3}$. If the requesting user decides to give the task to n_2, n_2 will receive $\gamma_c^{n_3}$ FlopCoins since that is the next bid bigger than $\gamma_c^{n_2}$. If h helpers are selected, with $h > 1$, then all of them will be paid according to the $h+1$-th bid bigger than the maximum of the selected bidders. In the following, we denote with ξ_c the final price the aided device has to pay for the task c.

The bids depend on the cost described above by function $f_n^{\mu}(\cdot)$ and the reciprocal trust between the helper and the requester. The actual bid is an array of bids, one for every possible combination $\mathbf{a}_k$:

$$\begin{aligned} b_n(\mathbf{a}_k, i, t) &= \Big(\gamma_1^n, \gamma_2^n, \dots\Big) = \\ \Big(1 - F_n^{\mu}(i)\Big) &\cdot f_n\big(\mathbf{a}_k, b_{in}(t), b_{ni}(t), l_{in}(t), l_{ni}(t)\big) \end{aligned}$$

Each helper n, after executing the received task and returning the result to the requester, receives the following number of FlopCoins:

$$m_n = \sum_{c=1}^{|A^o|} \sum_{n:x_c^n=1, y_c^n=1} \xi_c > \sum_{c=1}^{|A^o|} \sum_{n:x_c^n=1, y_c^n=1} \gamma_c^n. \quad (4)$$

So for the requester the total offloading cost in terms of FlopCoins is: $\sum_{n \in \mathcal{N}_i} m_n$.

At the end of the auction, the user who needs help broadcasts a signed message (msg_1) with the list of the selected neighbors that are going to help her. This message is the acknowledgment of the helpers that they are going to receive the FlopCoins if they will deliver their task. At the end of the offloaded executions and before the requester starts executing locally the remaining parts of the application, she broadcast a second signed message (msg_2) where she includes all the helpers that replied with the output of the offloaded to them task and the amount they will get. Any user who receives her payment broadcasts an acknowledgement (msg_3) together with reputation related information (more details about these information in Section VII). The set of these three messages define a complete *FlopCoin transaction* between mobile users. Any user in the neighborhood can collect any subset of these messages.

V. D2D Offloading Decision

To take an offloading decision, the requester considers the importance of the application and the current state of the device. In some cases, a user may need application's output as soon as possible, while in other she may be interested in prolonging her battery lifetime. We consider the following user goals: **(i)** maximize QoE by minimizing the execution cost based on the developed utility functions, **(ii)** maximize the probability of receiving the result of the offloaded tasks by selecting multiple helpers task, and **(iii)** minimize the number of FlopCoins required to execute the application. Given a metric μ, we define $\lambda_c(\mu)$ as the cost of executing the task locally, and $\eta_c^n(\mu)$ as the cost of offloading the task to neighbor n. The offloading decision becomes an optimization problem, which can be formulated as at Equations (5)-(9), where B is an upper bound of the budget per task, and T denotes a lower bound in the probability of receiving the result of task c from the selected helpers. Because of the stochasticity of θ_n, the output of the optimization problem may not completely reflect the reality. We can reduce the entropy of the mechanism if we select the neighbors n with variance smaller than a threshold, i.e. $\sigma_n^2 \leq V$.

$$\min_{\mathbf{x}} \sum_{c=1}^{|\mathcal{A}^o|} \Bigg(\sum_n x_c^n \Big(\theta_n \eta_c^n(\mu) + (1-\theta_n)\lambda_c(\mu)\Big) \quad (5)$$
$$+ \lambda_c(\mu) \prod_n (1 - x_c^n) \Bigg)$$

subject to:

$$\sum_{c=1}^{|\mathcal{A}^o|} \sum_n x_c^n \xi_c \leq L \cdot B \quad \forall c, n \quad (6)$$
$$\zeta_c \geq T \quad \forall c \quad (7)$$
$$\sum_{c=1}^{|\mathcal{A}^o|} x_c^n r_c \leq \mathcal{R}_n \quad \forall n \quad (8)$$
$$x_c^n \in \{0,1\}, \eta_c^n(\mu) \geq 0, \xi_c \geq 0, \lambda_c(\mu) \geq 0, \theta_n \in [0,1] \quad (9)$$

The space of possible controls requires exponential time and space and is not suitable for a fast decision. To overcome these issues, we use pre-processed data to provide a linear complexity algorithm in Section VI and we evaluate its performance, for multiple metrics, in Section VIII-B, while in Section VII we explain how the θ values are calculated.

VI. Neighbor Selection Algorithm

The solution of problem (5)-(9) is not linear or convex and can not be solved via an optimal algorithm with low complexity. The stochasticity of the mobile environment and the mobile users' unawareness that their mobile devices are used for executing others' tasks makes any optimally designed algorithm to outperform because it is impossible to predict the users' movement and any other cause of disconnections. We propose an online algorithm with linear complexity on the offloadable tasks and the possible helpers. Its most heavy part is a pre-processing computation that sorts the neighbors list and the tasks list. The task list pre-processing is executed only once, during the installation phase of the application. The tasks are sorted based on their priorities and on their dependencies. Our algorithm finds the set of neighbors to offload the tasks based on their reputation score in the requester's database, constrained by the upper bound of budget per task (Eq. 6), and a minimum guarantee that the offloading will be successful (Eq. 7). The algorithm calculates the FlopCoins and it keeps offloading tasks as long as there is enough budget and neighbors.

In the worst case, the algorithm examines every task for every neighbor. In an average case, if there is a neighbor with high reputation score nearby, the decision will be taken in constant time. If the aided device doesn't receive the offloaded task from a helper before the imposed deadline, as described in Equation 2, she executes the

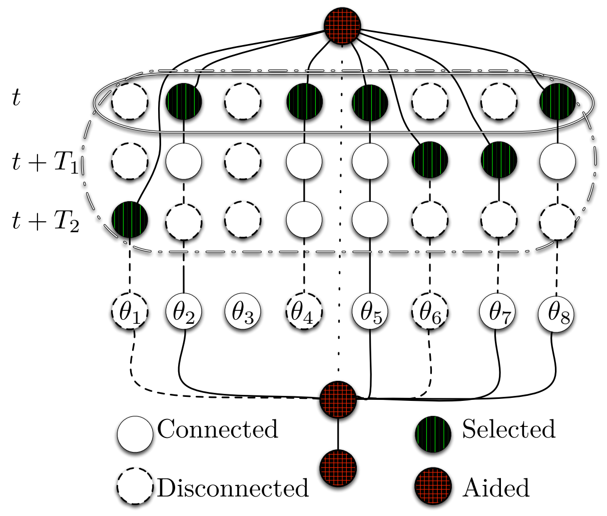


Fig. 2: Visual example with 9 users (Aided and 8 more). At time t, $\mathcal{N}_{aided}(t) = \{2,4,5,8\}$ and $\theta_i \geq \theta_j$ for $i < j$.

task locally. If the aided device meets another known peer who was not in the list of possible helpers at the time it executed our algorithm, it offloads the task to her as well, if she has available budget and has not offloaded the task to enough helpers in order to satisfy Equation 7.

Visual Example: Figure 2 is a graphical representation of how the algorithm works. On application execution, the aided device, at time t, can offload only to four nodes $\mathcal{N}_{aided}(t) = \{2,4,5,8\}$, since the others are disconnected (first row). Later, after T_1, in the second row, she connects with two more devices $\mathcal{N}_{aided}(t+T_1) = \{2,4,5,6,7,8\}$. After checking if the remaining budget is enough, she offloads some tasks to them as well. If the probability of receiving the result is not high enough, when she meets user 1 (third row) $\mathcal{N}_{aided}(t+T_2) = \{1,2,4,5,6,7,8\}$, she offloads also to him. However, because of the stochastic environment and users' mobility only helpers 2, 5, 7, and 8 managed to send their tasks before the deadline and the aided device executed the remaining ones locally.

VII. Reputation Mechanism

We propose a subjective reputation scheme. Mobile users exchange their opinions about others and, based on their trust, they update their opinions. The opinion exchange process is triggred on demand and not periodically in order to be energy efficient and to do not cause extra managerial traffic. Any user i keeps a trust database F_i and a reputation database H_i, both with values from 0 to 1, about the other peers. Trust values are set by the final user to specify the real trust users have towards each other, while reputation is updated automatically according to the following equation:

$$H_i^n(k) = \begin{cases} i\text{'s opinion about } n & \text{if } k = 0 \\ k\text{'s opinion about } n & \text{if } F_i(k) \geq TR_i \\ 0 & \text{if } F_i(k) < TR_i. \end{cases} \tag{10}$$

where k represents a friend of i with trust score (from database F_i) higher than a threshold TR_i. $H_i^n(0)$ represents the first-hand opinion of i about n, based on their mutual cooperation. $H_i^n(k)$ is the reputation score of neighbor n about k. We use a Bayesian framework [22], to determine the probability distribution θ_n that neighbor n will be able to return an offloading result in time. More specifically, we use a beta distribution $Beta(\alpha,\beta)$ to determine this probability. If there is no available information about neighbor n, $\alpha_i^n(0) = \beta_i^n(0) = 1$ which means that the distribution is uniform. User i saves the opinion of user k about user n:

$$\theta_n(k) \sim Beta(\alpha_n^k, \beta_n^k),\ H_i^n(k) = \{\alpha_i^n(k), \beta_i^n(k)\} \tag{11}$$

We calculate the probability of one neighbor to be trustworthy via the weighed average: $H_i^n = \{\alpha_i^n, \beta_i^n\}$ where:

$$\alpha_i^n = \frac{\alpha_i^n(0) + \sum_{k\in\tilde{\mathcal{N}}_i} E[\theta_i(k)]\alpha_i^n(k)}{1 + \sum_{k\in\tilde{\mathcal{N}}_i} E[\theta_i(k)]} \tag{12}$$
$$\beta_i^n = \frac{\beta_i^n(0) + \sum_{k\in\tilde{\mathcal{N}}_i} E[\theta_i(k)]\beta_i^n(k)}{1 + \sum_{k\in\tilde{\mathcal{N}}_i} E[\theta_i(k)]}$$

where $\tilde{\mathcal{N}}_i$ contains all the contacts $k : F_k \geq TR_i$. We propose a collective intelligence scheme where each user i in the neighborhood broadcasts her updated opinion $H_i^{\bar{n}}$ for user $\bar{n}$ with whom she interacted. If user i receives this broadcast message, she updates her opinion according to equation (12). α_i^j and β_i^j contain the weighted time average of all the interactions between i and j. Any increase of α_i^j moves the mass of user's j distribution to the right, that means that increases her reputation, while any increase of β_i^j moves the distribution to the left and decreases her reputation. Both increments improve the knowledge user i has about user j and as a result decrease the variance of the distribution.

After the execution of the offloaded task, the requester i calculates how beneficial was the offloading and then updates the reputation of every participating device. If a neighbor n caused improvement x, then

$$\alpha_i^n(0) = \alpha_i^n(0) + x \tag{13}$$

while if it caused damage y, i will update n's reputation

$$\beta_i^n(0) = \beta_i^n(0) + y \tag{14}$$

The values of x and y can be calculated based on equation 1. The benefit cause by a successful task offloading is the difference between the local execution cost and the cost of offloading the task and receiving the result. The harm caused by an unsuccessful offloading is the sum of the local execution and the cost of offloading the task. In the case of one helper is not getting paid, she decreases the asker's reputation by the FlopCoins she was going to receive.

$$\beta_n^i(0) = \beta_n^i(0) + m_j \tag{15}$$

TABLE I: Characteristics of offloadable applications in related work.

Name	Energy (J)	Duration (s)	Data Sent(KB)	Data Received (KB)	Reference
Face Detection	**3**	**8**	*100*	*1*	[1]
400 frames video game	**52**	**1,6**	*1*	*122,880*	[2]
Virus Scanning	**49**	**59,3**	**1024**	*1*	[3]
behavior profiling	**2,4**	**3,3**	*1*	*1*	[3]
speech-to-text	*44,7*	**16**	**2048**	*12*	[7]

A. The disconnection case

Every time two nodes meet, they exchange at least three messages, namely msg_1, msg_2, and msg_3, as defined in Section IV. In case all these messages were correctly delivered, the system keeps working as normal. However, a few undesirable cases are possible:

Loss of msg_1: A user that bids to help in an execution is no longer connected to the asker and is not receiving any offloaded task. In this case, the asker needs to find more mobile users in order to satisfy the threshold of Equation 7. For this reason, the asker decreases the reputation of the helper, but not vice-versa.

Loss of msg_2: A user received a task for execution, she executed the task, and got disconnected before she sent the result to the asker. This user will not be able to send the result to the asker immediately, so she will not be paid and it can even see his reputation be decreased. The helper user will also decrease the reputation of the asker, given that she did some job and it didn' get paid. However, if these two nodes meet in the future and before the task's deadline, they will be able to fix their reputations and the asker will be able to get the result back.

Loss of msg_3: When msg_3 is lost, only the surrounding neighbors are loosing the opportunity of collecting one more transaction.

VIII. Implementation & Evaluation

We implement an event-driven simulator to depict the performance of our proposal. We used three datasets, Infocom 05 and Infocom 06 from the Haggle project [23] and Humanet [24], which contain user mobility traces in different environments. The duration of the simulation is one day. We considered all the mobile users, which are 41 for Infocom 05, 78 for Infocom 06, and 56 for Humanet. Infocom traces present similar properties with each other while Humanet is evidently different. In Humanet, mobile users are characterized by a higher number of contacts, shorter contact duration, and smaller intercontact time.

To be as realistic as possible, we simulate the tasks by choosing uniformly from five representative applications used in previous works, presented in Table I. In bold we denote the values specified by the authors in their original papers. Whenever these values were missing, we estimated them based on the description of each application. The estimated values are denoted in italic. Furthermore, to evaluate the energy consumption of task transmission and result reception, we measured the energy per bit consumed when transmitting/receiving with Wi-Fi. We sent/received 10 MB of data using Wi-Fi with 10Mb/s data rate and 10ms latency. We generate 100 tasks per user with arrival rate following a Poisson distribution with $\lambda = 0.0055$ tasks per second. The initial amount of FlopCoins is 8000 and the initial battery capacity 100% for all users. The users are divided in three categories: altruistic—who put the battery lower bound to 25%, conservatives—who put the battery lower bound to 75%, and selfish—who put the battery lower bound to 95%. All the results are averaged over 50 simulation runs and standard deviation error bars are provided.

TABLE II: Performance Metrics

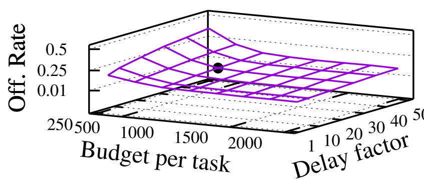
Name	Description
Total budget	Initial number of FlopCoins given to each user.
Budget per Task	An upper bound on the amount of FlopCoins that a node is willing to pay when offloading. This value is usually set by the user through the UI in Figure 1.
Delay factor	Denoted with τ_d, is a multiplication factor used to set the maximum allowed delay of the tasks: $\tau = \tau_c * \tau_d$.
Busy Node	If a node receives a task it becomes busy. A busy node will not accept more tasks while there is one in execution.
Successfully Offloaded Task	A task that was offloaded and the result was successfully received before the deadline and before the local execution started.
Offloading Rate	Fraction between the successfully offloaded tasks and all generated tasks.
Generated Traffic	Number of all offloaded tasks over the number of successfully offloaded. It represents the number of replicas created in the system per each successful offloaded task.

A. Benchmark Algorithms

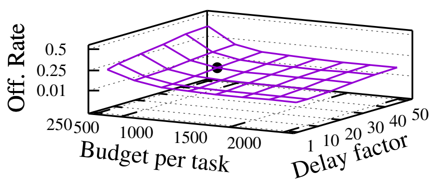
We implement the following three benchmark algorithms and compare them with our strategy using the metrics described in Table II:

Just-in-Time algorithm: Users try to offload only at the moment when a task is generated. If the user has a task for execution, she scans the neighborhood and tries to offload the task to all the nodes in contact. If there are no nodes in contact, the task is executed locally.

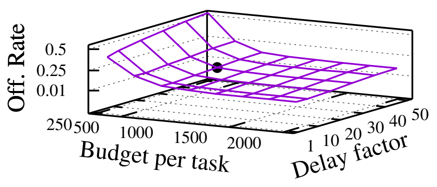
Look-Ahead algorithm: Users try to offload whenever they meet another node, if the task is not already being executed locally.



(a) Humanet

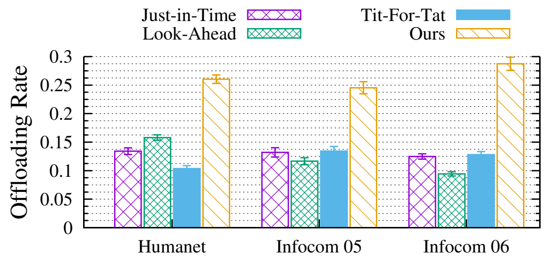


(b) Infocom 05

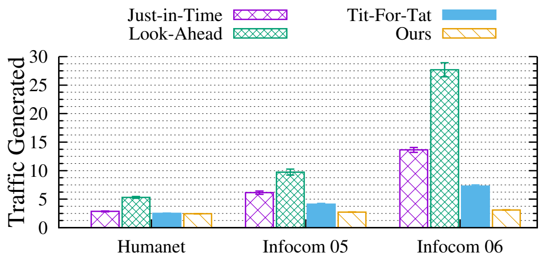


(c) Infocom 06

Fig. 3: Offloading rate of the proposed algorithm for different values of Budget per task and Delay factor. The black dot represents the selected delay and budget values used in the rest of the experiments.



(a) Offloading Rate



(b) Generated Traffic

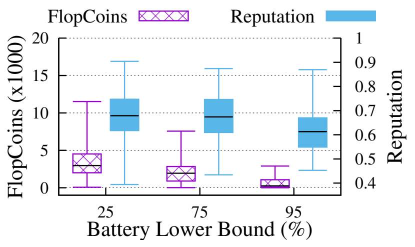
Fig. 4: Comparison of the proposed algorithm with Just-in-Time, Look-Ahead and TFT algorithms in terms of Offloading Rate (higher is better) and Generated Traffic (lower is better).

Tit-for-Tat algorithm: Users act as in the Look-Ahead case but they respond to offloading requests depending on their past experience. Initially, every node is willing to help any other node, but after their first interaction, they mimic the others' nodes action. For example, if node i offloaded a task to node j and she received the result of the task in time, she will accept a task from j in the future. On the contrary, if i didn't receive the result from j, she will reject any request from j until she offloads another task to j and correctly receives the result. In all simulations the users keep executing until all the tasks are finished, the simulation is finished, or their battery depletes.

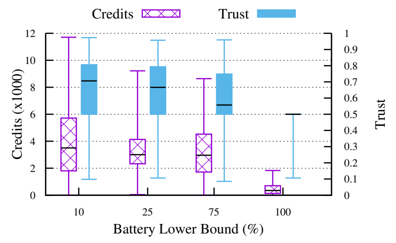
B. Simulation Results

We initially examine the performance of our algorithm for different delay factor values $\tau_d \in \{1, 10, 20, 30, 40, 50\}$ and varying the budget per task from 250 to 2000, increasing it by 250. We visualize the results of this analysis in Figure 3. A combination of small budget per task and high delay factor gives the best results in terms of offloading rate. However, setting small budget per task upper bound impedes offloading of large tasks, since neighbor nodes will refuse to consume their resources without getting the right compensation. According to other simulation parameters, i.e. total budget and considered applications, we fixed the budget per task upper bound to 750, which is the smallest value that allows even the biggest task to be offloaded. We fixed the delay factor τ_d to 30, in order to increase the maximum allowed delay of the shortest task to more than 30 seconds, so to increase its chances of successful offloading. These values are represented by the black dots in Figure 3. As we can see from these plots, by choosing these values we penalize our algorithm, since other combinations could allow for higher offloading rate values. Anyway, we make this choice with the consideration of our initial assumption that each offloadable task should have the chance to be offloaded.

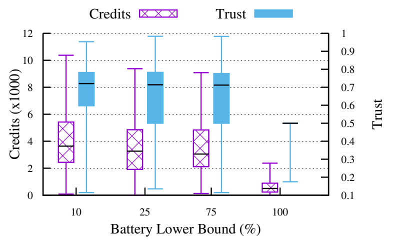
For all strategies, Figure 4a shows the *Offloading Rate*, defined as the fraction between the successfully offloaded tasks over all generated tasks, while Figure 4b shows the *Traffic Generated*, defined as the number of replicas created in the system for each task. To better understand the dynamic of the system, these figures should be read together considering that high offloading rate usually leads to high traffic. From Figure 4a, it is clear that our algorithm outperforms the benchmarks in terms of offloading rate in all traces, by offloading more than 25% of the tasks, while the others have an offloading rate of less than 15%. As Figure 4b shows, our strategy achieves the high offloading rate with a very low overhead in terms of traffic. In all the traces our algorithm generates less than 3 replicas in the system, while the other strategies have high variability depending on the trace, reaching tens of replicas in case of Infocom 06. The traffic variability of the other algorithms on the traces is due to the statistical properties of each trace. For example, compared to the two Infocom traces, in Humanet the contact durations between users are very short, so devices don't have time to exchange many tasks, even though they meet very frequently. As for the two Infocom traces, they present similar properties, except for the number of users and



(a) Humanet



(b) Infocom 05



(c) Infocom 06

Fig. 5: Redistribution of FlopCoins and Reputation Scores among the three different user categories. The reputation score of the selfish users decreases with time, making them undesired by other nodes for offloading. This way, they quickly consume their FlopCoins, and without being able to gain more their possibilities for task offloading are limited.

number of contacts between users, which are higher in Infocom 06. This explains the higher number of messages in Infocom 06 (Figure 4b).

Compared to the Just-in-Time approach, our strategy does not offload to nodes who are in contact on task arrival, but it is more careful, hoping to meet nodes with high reputation before the task deadline. Moreover, our algorithm decides to stop offloading the task to other nodes if the probability of getting back the result becomes higher than a threshold, as explained in Eq. 7. According to the simulations, our algorithm never selects more than 3 nodes in order to reach the threshold (see Figure 4b) on all traces, which reduces the number of replicas in the system significantly. At first, one would expect the Look-Ahead to have the highest offloading rate. The problem with this approach is that the nodes get overloaded by offloading requests, so they soon become busy and stop accepting tasks for execution. This is also confirmed by the high traffic generated as shown in Figure 4b.

Is interesting to notice how Tit-for-Tat approach not only performs worst than our strategy in terms of offloading rate, but it also does not perform better than the other two benchmark algorithms, which do not implement any form of intelligence. In this strategy, nodes refuse to accept tasks from other nodes until a successful offloading is accomplished. Following the actions of previous negative interactions, impedes many tasks from being offloaded resulting in low offloading rate, but at the same time reduces the number of message exchanges, resulting also in low overhead, when compared to the other non-intelligent schemes (see Figure 4b). Even so, the number of replicas generated by this strategy is higher than our algorithm.

After comparing our strategy with the three benchmarks, we performed a new experiment to show the effects of the incentive scheme and the reputation mechanism on rewarding the collaborating nodes and sidelining the selfish ones. Given the limitations of the datasets, we diminished the initial budget of each node to 2000 FlopCoins, so that the selfish nodes can finish their budget during one-day. All the nodes are initialized with the same reputation score, which is 0.5. Figure 5 shows how the FlopCoins are distributed among the users and how the reputation of the users changes after the simulation. These results are presented separately for users belonging to one of the three categories previously defined: altruistic, conservative, and selfish. In the y1-axis we show the box-plot representing the minimum, 25^{th}, 50^{th}, 75^{th} percentiles, and the maximum of the cumulative number of FlopCoins at the end of the simulation for each group. In the y2-axis we show the box-plot of the cumulative reputation of the users in each group. The trend from left to right is decreasing for both metrics, until it drops to a minimum for the free-riders (nodes with battery threshold 95%). It is important to notice that these values are strictly related to each other. When a node selects another from the list of the known nodes, it does so according to their reputation score: The higher the reputation, the higher the chances for a node to be selected. Based on this intuition, in general, a node will only increase its reputation or maintain the same one (if a node has worst reputation than the others, it will not be selected very often, so the chances that it fails to successfully execute a task are lower). This intuition is confirmed by the results of Figure 5, where we can see that almost all selfish nodes finish their FlopCoins, while the altruistic ones increase their budget. In the long term, if the selfish nodes want to be part of the system, they should start gaining some FlopCoins and improve their reputation. To do so, they should be more generous and change the bounds accordingly, e.g. decrease the battery lower bound.

IX. Discussion

Consideration of malicious users: We are aware of the fact that the device–to–offloading architecture is vulnerable to different kind of attacks. For example, in our system a malicious user can *hack* the system by artificially increasing his FlopCoin budget so he can keep offloading even though he does not contribute by executing tasks for others. Another possible attack is for malicious users to collude with each other and artificially increase their reputation. In this work, we do not consider these or other types of attacks, which are part of our future works. We only focus on the architectural and algorithmic design of a D2D offloading framework that deals with selfish users.

Privacy issues of Computation Offloading: It is worth mentioning that the use of our proposal does not increase the chances of any privacy leakage. First, the privacy defenses are implemented by the offloading module, which is not part of the current work. In general, offloading modules deal with this issue depending on the operating system of the mobile devices. For example, an Android offloading module will run the offloaded task on an isolated Dalvik Virtual Machine [1], limiting the task's access on the physical device.

X. Conclusion and Future Work

We proposed a D2D computation offloading framework that integrates a distributed incentive scheme and a distributed reputation mechanism. The framework was designed following the principles of the Hidden Market Design approach.and it works transparently, as a background process, explores and connects with nearby devices, communicates with them, learns and estimates their trustworthiness. More importantly, the underlying neighbor–selection algorithm is linear with low complexity. We compared our scheme with three benchmark strategies and observed that our algorithm outperforms the others in terms of task offloading rate and traffic generated. Furthermore, we showed how a combination of the incentive mechanism and the reputation scheme rewards the collaborating users, while punishing and sidelining the selfish ones. As future work, we will integrate in our design security mechanisms to protect against malicious users who can tamper the software or create colluding groups, we will implement the framework on real devices, and we will perform experiments with real users.

XI. Acknowledgements

This research has been supported, in part, by General Research Fund 26211515 from the Research Grants Council of Hong Kong, Innovation and Technology Fund ITS/369/14FP from the Hong Kong Innovation and Technology Commission, and the European Commission under the Horizon 2020 Program through the RAPID project (H2020-ICT-644312).

References

[1] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proceedings of IEEE INFOCOM*, 2012.

[2] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10, 2010, pp. 49–62.

[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of EuroSys*, 2011.

[4] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "Comet: Code offload by migrating execution transparently," in *Proceedings of the 10th USENIX OSDI*, ser. OSDI'12, 2012, pp. 93–106.

[5] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket switched networks and human mobility in conference environments," in *Proceedings of SIGCOMM workshop on Delay-tolerant networking*, 2005.

[6] E. E. Marinelli, "Hyrax: cloud computing on mobile devices using mapreduce," Carnegie Mellon University, Tech. Rep., 2009.

[7] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of MobiHoc '12*, 2012, pp. 145–154.

[8] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond*, ser. MCS '10, 2010, pp. 6:1–6:5.

[9] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," in *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International.* IEEE, 2012, pp. 123–132.

[10] Y. Li and W. Wang, "Can mobile cloudlets support mobile applications?" in *INFOCOM, 2014 Proceedings IEEE.* IEEE, 2014, pp. 1060–1068.

[11] D. Chatzopoulos, K. Sucipto, S. Kosta, and P. Hui, "Video compression in the neighborhood: An opportunistic approach," in *Communications (ICC), 2016 IEEE International Conference on*, 2016.

[12] C. Bermejo, R. Zheng, and P. Hui, "An empirical study of human altruistic behaviors in opportunistic networks," in *Proceedings of the 7th International Workshop on Hot Topics in Planet-scale mObile Computing and Online Social neTworking*, ser. HOTPOST '15, 2015, pp. 43–48.

[13] D. Chatzopoulos, P. Hui, and D. Huang, "Fides: A hidden market approach for trusted mobile ambient computing," in *IEEE INFOCOM Student Workshop (Poster Session)*, 2015, pp. 86–87.

[14] S. Seuken, D. C. Parkes, E. Horvitz, K. Jain, M. Czerwinski, and D. Tan, "Market user interface design," in *Proceedings of the 13th ACM Conference on Electronic Commerce*, ser. EC '12, 2012, pp. 898–915.

[15] R. Axelrod, "The emergence of cooperation among egoists," *American Political Science Review*, vol. 75, pp. 306–318, 6 1981.

[16] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, 2009.

[17] M. Guirguis, R. Ogden, Z. Song, S. Thapa, and Q. Gu, "Can you help me run these code segments on your mobile device?" in *IEEE Global Telecommunications Conference (GLOBECOM)*, 2011.

[18] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing.* ACM, 2014, pp. 287–296.

[19] L. Anderegg and S. Eidenbenz, "Ad hoc-vcg: A truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents," in *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '03, 2003, pp. 245–259.

[20] P. Michiardi and R. Molva, "Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," in *Proceedings of the IFIP TC6/TC11 Sixth Joint Working CCMS: ACMS.* Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 2002, pp. 107–121.

[21] V. Krishna, *Auction Theory.* Academic Press, 2002.

[22] S. Buchegger and J.-Y. Le Boudec, "A Robust Reputation System for Peer-to-Peer and Mobile Ad-hoc Networks," in *P2PEcon 2004*, 2004.

[23] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, "CRAWDAD data set cambridge/haggle (v. 2006-01-31)," Downloaded from http://crawdad.org/cambridge/haggle/, Jan. 2006.

[24] J. M. Cabero, V. Molina, I. Urteaga, F. Liberal, and J. L. Martin, "CRAWDAD data set tecnalia/humanet (v. 2012-06-12)," http://crawdad.org/tecnalia/humanet/, Jun. 2012.