Mobile to Mobile Computational Offloading in Multi-hop Cooperative Networks

Colin Funai, Cristiano Tapparello, Wendi Heinzelman
Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY, USA
Email: {firstname.lastname}@rochester.edu

***Abstract*—As the number of mobile devices that natively support ad hoc communication protocols increase, large ad hoc networks can be created not only to facilitate communication among the mobile devices, but also to assist devices that are executing computationally intensive applications. Prior work has developed computation offloading systems for mobile devices, but this work has focused exclusively on offloading to single hop neighbors, due in part to the practical challenges of setting up multi-hop networks using existing ad hoc communication protocols. However, limiting the offloading of computation to one-hop neighbors inherently restricts the number of devices that can participate in the distributed computation. In this paper, we propose and evaluate the performance of computational offloading within a multi-hop cooperative network, where mobile devices are able to share the computational load with all other nodes in the network. Additionally, we present an iterative task assignment algorithm that can optimize the assignment of computational tasks to devices in such a multi-hop cooperative network, taking into account the communication overhead of the multi-hop network. Experimental results, obtained from an implementation on Android devices, are integrated with an analytical model that enables the evaluation of system performance under a variety of conditions. These experimental and analytic results demonstrate the benefit of enabling computation offloading to all devices in a multi-hop cooperative network.**

I. Introduction

Recent advancements in VLSI and MEMs have allowed for increasingly complicated processors to become cheaper and more accessible to a variety of applications. For instance, these processors have become so accessible that manufacturers are even offering ruggedized devices that use these processors [1], while still holding certification for certain environmental conditions, such as low pressure or extreme temperature [2].

With devices that can withstand such harsh conditions, it is not hard to imagine that when encountering hazardous situations, a first responder would benefit greatly from having the ability to take in data from the field, process it, and make an informed decision based on the result. For example, first responders might be equipped with ruggedized smart devices and deployed to the site of an earthquake to aide in relief efforts, or a military patrol might be able to perform a Signal Intelligence (SIGINT) application for finding improvised explosive devices (IEDs) [3]. Typically, these kinds of applications require immense processing capabilities and consume large amounts of the device's battery, if the device is even able to perform such computations.

One potential solution is for the mobile device to offload these computationally intensive tasks to a remote server [4], [5], either by setting up relay points throughout the area of interest or using satellite communication [6]. System like MAUI [4] and CloneCloud [5] are based on a remote execution scheme, where devices offload computationally intensive portions of an application to a remote server that returns the computed result. Although the aforementioned remote execution systems exhibit potential for massive performance gains, oftentimes military or disaster relief scenarios do not allow for the luxury of being able to permit high latencies associated with offloading computation to remote servers, either due to bandwidth or environmental constraints. Additionally, in these scenarios, the devices may not be connected to a remote server at any given time.

The concept of cyber foraging, where devices use nearby shared computing resources, was presented to provide an alternative to the high latencies associated with offloading computation to remote servers [7]. While cyber foraging is presented in the context of public, untrustworthy and unmanaged computing resources, this practice could be translated to military or disaster relief scenarios provided that the resources are deployed over an area of interest a-priori.

On the other hand, Hyrax [8] proved the benefit of utilizing all of the nodes in a network. By porting Hadoop Apache to mobile devices, the authors were able to create a distributed computing file system and enable distributed computing on mobile devices. Hyrax achieves this by utilizing a central server and infrastructure based communications.

Starting with the work presented in [8], several works presented solutions that are able to offload computation to other nearby mobile devices in a complete ad hoc manner [9]–[11]. Honeybee, for example, provides a programming framework to aide mobile to mobile computational offloading, using Bluetooth to facilitate the communication between devices [9]. More recently, the authors in [10], [11] proposed and implemented a mobile computational offloading system using Android's implementation of WiFi Direct.

While these works have shown the feasibility of offloading to mobile devices in an ad hoc setting, their implementations are limited to a single hop ad hoc network. The next step in the evolution of mobile computation offloading is to consider all the computational resources available in a multi-hop ad hoc

This work was supported in part by Harris Corporation, RF Communications Division and in part by CEIS, an Empire State Development designated Center for Advanced Technology.

network. In this regard, Serendipity enables remote computing among mobile devices over intermittently connected ad hoc networks [12], using a technique similar to those used in disruption tolerant networks [13]. By utilizing a modified version of Dijkstra's routing algorithm, the authors presented a water filling based task assignment scheme, that uses an estimate of the time required to transfer data between two nodes to calculate the number of tasks to assign to a particular node. Serendipity's performance has also been verified through an implementation on Android devices.

Given the above, our work aims at overcoming the limitations of the previous work by exploiting all the computational resources available in a multi-hop ad hoc network. In particular, our contributions in this paper are:

- Using an implementation on Android devices, we demonstrate the benefit of expanding the number of computational resources that can be utilized for task completion, and show that in a real application, a simple Greedy task distribution is able to reduce the total time of the distributed computation.
- We develop a provably optimal algorithm for distributing tasks to nodes within a multi-hop network that can be used to estimate the performance of offloading the computation over a multi-hop ad hoc network under different settings.

Hence, the overall conclusion of this paper is that not only is it feasible to offload computation to nodes that are multiple hops away, but there is also a measurable performance gain that can be achieved by doing so.

The rest of the paper is organized as follows. In Section II, we describe our mobile to mobile computational offloading system, and motivate the need for an intelligent task assignment. In Section III, we present an analytical model for calculating the amount of time and energy that is required to assign tasks to different nodes in a multi-hop network. In Section IV, we define an objective function representing the cost to compute some number of tasks, as well as a provably optimal iterative algorithm for optimizing this objective function. Section V presents measured and analytic results for computation offloading in a multi-hop WiFi Direct network. Section VI concludes the paper.

II. Motivation

Our mobile to mobile computational offloading system is composed of a set of N mobile devices, organized into a network through ad hoc communications. The devices are cooperative, meaning that we do not consider the presence of any selfish user. Moreover, we assume the existence of a global routing protocol, and each node has knowledge of all other available nodes within the network.

At a certain point in time, one of the devices needs to run a complex application, which can be divided into clearly defined homogenous tasks, as is the case for a military SIGINT application [3], or non-invasively determining the structural integrity of a building during disaster relief efforts [14]. These tasks are independent from each other, and their computation

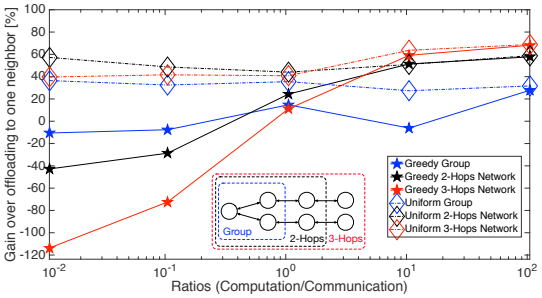


Fig. 1. Experimental measurement of the percent gain over offloading to only a single neighbor of a simple greedy and uniform task distribution scheme.

can be parallelized by offloading the tasks to different devices. The results of all the tasks' execution is then merged in order to complete the initial complex application. As a result, assigning a task to a device requires the transmission and reception of the task and the results of the task execution over the ad hoc route. This requires communication energy consumption on all the nodes along the routing path, and incurs a time delay due to the propagation of the data throughout the network.

In order to determine the benefit of utilizing multi-hop mobile to mobile computation offloading, we implemented a computational offloading system on Android devices, utilizing the WiFi Direct protocol [15]. We note that the functionalities for creating multi-group networks using WiFi Direct are not natively implemented in Android [16]. Nevertheless, the work presented in [16], [17] provide different solutions (e.g., time sharing between groups, broadcasting and multicasting data between groups and modifications of the Android OS) to seamlessly enable multi-group communication using stock and non-stock Android devices.

Given the above, we developed a testbed by interconnecting several 2013 Nexus 7s (N7), using the modified version of Android presented in [16]. The 2013 N7 has 16 GB of storage, 2 GB of memory, and a 1.5GHz quad-core Snapdragon CPU [18]. For all the results presented in this paper, we limited each device to compute only one task at a time, and the device could not request the next task until it returned the result. Each of the presented data points are an average of thirty trials. All of the measurements were taken in an indoor workspace, and the nodes were stationary during these experiments. We acknowledge that in real scenarios, nodes are generally mobile, and may enter or leave a network. However, to accurately determine the impact of the task assignment and to limit the variability across different experiments, we kept the nodes stationary. Moreover, all the results of this paper have been obtained with the screen turned off.

To evaluate the performance of the envisioned multi-hop mobile to mobile computation offloading system, in Figure 1 we compare the experimental performance gain to complete a set of 50 tasks relative to a task distribution that only utilizes one additional device with extending the computation to a larger network (see Figure 1). For assigning the tasks, we implemented both a simple Greedy task distribution, where

each device in the network requests a new task as soon as it has completed the previous task, and a Uniform task distribution, where the tasks are uniformly distributed throughout the network. While both task assignments provide substantial performance gain compared to offloading to a single neighbor when the computation time is much larger then the time required to assign the tasks and receive the results, extending the computation to the multi-hop network can be detrimental when the communication time is greater than the computation.

As a result, it is not straightforward to determine how to best assign the tasks due to the communication costs of distributing the tasks. Moreover, while both the Uniform and the simple Greedy task distribution already provide gains in some cases, it is not possible to directly determine the performance of a Greedy distribution before assigning the tasks and, at the same time, it is not clear if further performance improvements are even possible. Thus, in the next sections we first model the different elements of the system under consideration and then present an iterative algorithm that is guaranteed to return the optimal task assignment.

III. System Model

In this section we present the details of the different components of the system described in Section II, which are then used to derive the optimal task distribution policy presented in Section IV.

A. Task Time Model

We start by considering the total time D_i required to compute a given task at a particular device i. We define this total time as the time between the task assignment and the end of the transmission of the result of the task execution. Thus, in order to determine the time required to assign, compute and then receive the results of the task execution we need to consider two main components, namely the communication time required to distribute the task and receive the results, and the time actually spent for the task execution. In order to derive the communication time, we consider that each task is characterized by three parameters, namely the task data size t, the relative result data size r and the task complexity c. Thus, according to this definition, assigning a task (t, c, r) to a device requires the transmission/reception of t bits, while the result of the task execution to be reported back to the task generator entails the transmission/reception of r bits. The time required to execute a single task depends on the CPU of the mobile device and on the complexity c of the task to be executed, which, without loss of generality, can be considered as a function $e_i(c)$. Therefore, the total time required to compute a single task at device i depends on the device's computational capabilities, and on the characteristics of the radio technology used for the data exchange, as well as on the number of hops H_i between the device that generated the tasks and device i.

Given the above, the total delay $D_i(t, c, t)$ experienced by a task (t, c, r) assigned to device i is given by

$$D_i(t,c,r) = \sum_{l=1}^{H_i} \left(\frac{t}{T_l^{\text{tx}}} + \frac{r}{T_{\overline{l}}^{\text{tx}}} \right) + \sum_{l=1}^{H_i-1} 2T_l^{\text{sw}} + e_i(c), \quad (1)$$

where T_l^{tx} represents the transmission throughput of the l communication hop, $\overline{l}$ represents the communication hop l in the reverse path, and T_l^{sw} accounts for additional switching time when routing the data between two subsequent communication links as can be the case, for example, with the WiFi Direct protocol (see, e.g., [16]). We note that the transmission throughput also accounts for parallel use of the same communication link (by, e.g., the simultaneous assignment of different tasks) and, since the throughput on wireless networks fluctuates due to the signal strength and channel impairments, assigning values to the transmission throughput only serves as an example of the achievable performance of a network.

B. Task Energy Model

In most cases, the system described in this paper will consist of battery operated mobile devices. As a result, it is important not only to characterize the delay experienced by the computation of each task, but also to determine the impact in terms of energy consumption of the task distribution. Therefore, in what follows we define the total energy consumption required to assign a task to a particular device.

Similar to the task time model defined in the previous section, distributing a task entails a communication energy consumption and the task execution energy consumption. In order to define the communication energy model, we first define P_l^{tx} and P_l^{rx} to be the transmission and reception power consumption of a particular ad hoc communication link l, respectively. We note that a particular link l actually represents a pair of nodes (h, k), where h is the device transmitting the data and k is the device receiving the data. Thus, with a little abuse of notation we can write $P_l^{\text{tx}} = P_h^{\text{tx}}$ and $P_l^{\text{rx}} = P_k^{\text{rx}}$, where P_h^{tx} represents the power consumption of mobile device h during transmission, and P_k^{rx} represents the power consumption of mobile device k during reception. In the same way, we define P_i^{ex} to represent the power consumption of mobile device i during task computation.

As a result, we define the total energy expenditure when assigning a task (t, c, r) to device i as

$$\begin{aligned} E_i(t,c,r) = &\sum_{l=1}^{H_i} \left[\frac{t}{T_l^{\text{tx}}}(P_l^{\text{tx}} + P_l^{\text{rx}}) + \frac{r}{T_{\overline{l}}^{\text{tx}}}(P_{\overline{l}}^{\text{tx}} + P_{\overline{l}}^{\text{rx}}) \right] \\ &+ \sum_{l=1}^{H_i-1} 2E_l^{\text{sw}} + P_i^{\text{ex}} e_i(c), \end{aligned} \quad (2)$$

where, similar to Eq. (1), $\overline{l}$ represents the communication hop l in the reverse path (e.g., if l represents the pair of nodes (h, k), $\overline{l}$ refers to the pair (k, h)), and E_l^{sw} represents the amount of energy required for switching when routing the data between two subsequent communication links.

IV. Optimal Task Distribution

The goal of our analysis is to find the optimal task distribution policy that minimizes an overall cost metric by utilizing all the available nodes in a multi-hop network. In the following sections, we first formulate the optimal tasks distribution

problem, and then we present an iterative algorithm that is able to find the optimal task assignment.

A. Optimization Problem

Let N be the number of mobile devices participating in the distributed computation and M be the number of homogeneous tasks[1] to be distributed. We define $\boldsymbol{\phi}$ to be a vector such that $\boldsymbol{\phi} = [\phi_0, \phi_1, \ldots, \phi_N]$, where ϕ_i represents the number of tasks assigned to a particular device i, such that $\phi_i \in \mathbb{Z}^+$, $0 \leq \phi_i \leq M$ and $\sum_{i=1}^{N} \phi_i = M$, and Φ to be the set of all the possible task assignments that satisfy these conditions. Moreover, let $\boldsymbol{C} = [C_1, C_2, \ldots, C_N]$ be a cost vector, where C_i with $i = 1, \ldots, N$ represents the cost of assigning a task to device i.

Given the above, our objective is to solve the following optimization problem:

$$\min_{\boldsymbol{\phi} \in \Phi} \mathrm{F}(\boldsymbol{\phi}), \tag{3}$$

where $\mathrm{F}(\boldsymbol{\phi}) = \max_{i=1,\ldots,N} \{\phi_i C_i\}$ represents the cost of a particular task assignment $\boldsymbol{\phi} \in \Phi$. We note that $\mathrm{F}(\boldsymbol{\phi})$ accounts for the assignment of tasks to bottleneck nodes and is suitable for, e.g, finding the minimum time required to perform the distributed computation of the M tasks.

B. Proposed Policy

The optimization problem defined in Eq. (3) can be seen as a variation of the Linear Bottleneck Assignment Problem (LBAP) [19]. Different from a traditional LBAP, we consider the possibility of assigning to a particular agent (i.e., device) more than one task. We note that while different algorithms for solving the LBAP problem have been proposed in the literature, e.g., [20], allowing an agent to execute more than one task adds additional complexity to the problem. At the same time, the assumption of homogeneous tasks allows us to simplify the problem, which can be optimally solved with an iterative algorithm. In what follows, we first propose an algorithm to solve problem (3), and then prove the optimality of the solution determined by our algorithm.

In particular, we aim to find an optimal policy $\boldsymbol{\phi}$ for problem (3). To this end, let $\boldsymbol{X}^{\alpha}$ be a vector of length N, so that X_i^{α} represents the number of tasks that have been assigned to device i up to the assignment of task α, with $\alpha \leq M$. In addition, let $\boldsymbol{Y}$ be a decision vector of length N, so that all but one of the elements of $\boldsymbol{Y}$ are zero. Given the above, the algorithm for finding the optimal task assignment is defined in Algorithm 1.

C. Performance Analysis

To prove that Algorithm 1 is able to determine an optimal solution to the optimization problem defined in Eq. (3), we first show that starting from an optimal task assignment for K tasks, it is possible to construct an optimal task assignment for $K + 1$ tasks.

[1]The extension of the results of this paper to the case of non-homogeneous tasks is considered as future work.

Algorithm 1 Cost–Optimal Task Distribution

1: X^0=$\{0, 0, 0, ..., 0\}$
2: **for Each Task** $\alpha \in [1, ..., M]$ **do**
3: $\quad$ **Y**=$\{0, 0, 0, ..., 0\}$
4: $\quad$ $i = \mathrm{argmin}((\mathbf{X}^{\alpha-\mathbf{1}} + \overrightarrow{1}) \circ \mathbf{C})$
5: $\quad$ $Y_i = 1$
6: $\quad$ $\mathbf{X}^{\alpha} = \mathbf{X}^{\alpha-\mathbf{1}} + \mathbf{Y}$
7: **end for**

Theorem IV.1. *Given an optimal K tasks assignment Γ^K, an optimal $K+1$ tasks assignment Γ^{K+1} can be obtained as*

$$\Gamma^{K+1} = \underset{\psi \in \Psi^{K+1}}{argmin}[\ max_i\{\psi_i C_i | i = 1, ..., N\}],$$

where $\Psi^{K+1} = \{\psi^i\}$, $\psi^i = \Gamma^K + \epsilon^i$ and ϵ^i is an all-zeros vector with a one in position i, for each $i = 1, \ldots, N$.

Proof. We will prove this in two steps. In the first step, we consider the case where Γ^K is the only optimal task assignment for K tasks, while in the second step, we consider the case of multiple K task assignments that have the same optimal cost.

For the case where there is a unique optimal solution to distribute K tasks, let's assume by contradiction that Γ^{K+1} determined according to the above definition is not optimal. This, in turn, means that there exists a $K+1$ tasks assignment Θ^{K+1}, such that $\Theta^{K+1} \notin \Psi^{K+1}$ and

$$\mathrm{F}(\Theta^{K+1}) < \mathrm{F}(\Gamma^{K+1}). \tag{4}$$

Moreover, we can consider Θ^{K+1} to be derived from a K tasks assignment Θ^K, such that $F(\Gamma^K) < \mathrm{F}(\Theta^K)$ and

$$\mathrm{F}(\Theta^K) \leq \mathrm{F}(\Theta^{K+1}). \tag{5}$$

As a result, by combining (4) and (5), we obtain

$$\mathrm{F}(\Theta^K) < \mathrm{F}(\Gamma^{K+1}). \tag{6}$$

On the other end, assuming that $F(\Theta^K) = \Theta_m^K C_m$ (i.e., device m is the bottleneck device that determines the overall cost of the non-optimal task assignment Θ^K), we have that $\Gamma_m^K < \Theta_m^K$. Thus, $\Gamma_m^K + 1 \leq \Theta_m^K$. Now, if $\Gamma_m^K + 1 < \Theta_m^K$, then $\mathrm{F}(\Gamma^K + e^m) = \mathrm{F}(\Gamma^K) < \mathrm{F}(\Theta^{K+1})$, which contradicts the hypothesis in (4). If $\Gamma_m^K + 1 = \Theta_m^K$, instead, $\mathrm{F}(\Gamma^K + e^m) = \mathrm{F}(\Theta^K) \leq \mathrm{F}(\Theta^{K+1})$, which still contradicts the hypothesis in (4). As a result, if Γ^K is the unique optimal K tasks assignment, then the $K+1$ task assignment Γ^{K+1} obtained by Theorem IV.1 is an optimal $K+1$ tasks assignment.

For the case where $\mathrm{F}(\Gamma^K) = \mathrm{F}(\Theta^K)$, there exist at least one Γ_m^K, $m = 1, \ldots, N$ such that $\Gamma_m^K < \Theta_m^K$ since $\Gamma^K \neq \Theta^K$. Thus, $\Gamma_m^K + 1 \leq \Theta_m^K$ which, following an argument similar to the one describe above, results in $\mathrm{F}(\Gamma^{K+1}) = \mathrm{F}(\Gamma^K + e^m) \leq \mathrm{F}(\Theta^{K+1})$, which contradicts the hypothesis in (4).

Therefore, Γ^{K+1} obtained by Theorem IV.1 is an optimal $K+1$ tasks assignment. □

We will now use the result of Theorem IV.1 to show that the algorithm presented in Section IV-B returns an optimal solution to the optimization problem (3).

Theorem IV.2. *Algorithm 1 yields an optimal solution to the optimization problem defined in Eq.* (3).

Proof. In what follows, we will prove by induction on the number of tasks M that the solution determined by Algorithm 1 is an optimal solution to the optimization problem defined in Eq. (3). For the case $M = 1$, the set of all possible tasks assignment is represented by $\Phi = \{[1, 0, ..., 0], [0, 1, ..., 0], ..., [0, 0, ..., 1]\}$; as a result, we can easily see that

$$C^1_{opt} = \min_{\phi \epsilon \Phi} \left[\max_{i=1,...,N} \{\phi_i C_i\} \right] = \min(\overrightarrow{1} \circ \mathbf{C}), \quad (7)$$

where $\overrightarrow{1}$ is a vector of all ones. Thus, C^1_{opt} can further be simplified to

$$C^1_{opt} = C_i, \quad (8)$$

where $i = \text{argmin}(\overrightarrow{1} \circ \mathbf{C})$.

Now to prove that our algorithm is capable of yielding an optimal solution for $M = 1$ task, let C_{algo} be the cost of the solution produced by the algorithm, defined as

$$C^1_{algo} = \max_{i=1,...,N} \left\{ X^1_i C_i \right\}. \quad (9)$$

Since $\mathbf{X^0}$ is initialized to a vector of zeros, as shown in step 2 of our algorithm, C^1_{algo} simplifies to

$$C^1_{algo} = \max \left[\{(X^0_i + 1) C_i\} \bigcup \{(X^0_j + 0) C_j | \forall j \neq i\} \right], \quad (10)$$

where X^0_i is the device which is chosen at steps 4 and 5 of our algorithm, and $\mathbf{X^0_j}$ is the subset of $\mathbf{X^0}$ that were not chosen at step 4 (i.e., by the argmin function). Eq. (10) can further be rewritten as

$$C^1_{algo} = \max \left[\{1 \cdot C_i\} \bigcup \{0 \cdot C_j | \forall j \neq i\} \right] = C_i, \quad (11)$$

which, compared with Eq. (8), shows that C^1_{algo}=C^1_{opt}.

Let's now assume that $C^K_{algo} = C^K_{opt}$ up to $M = K$, and that $\boldsymbol{X}^K = \Gamma^K$, where Γ^K is the optimal solution for $M = K$. For the case where tasks $M = K + 1$ we can write

$$C^{K+1}_{opt} = \min_{\phi \epsilon \Phi} \left[\max_{i=1,...,N} \{\phi_i C_i\} \right]. \quad (12)$$

However, as shown in Theorem IV.1, we can restrict the search of the optimal solution to the set Ψ^{K+1}, which has reduced cardinality to N. This means that we can rewrite Eq. (12) as

$$C^{K+1}_{opt} = \min_{\psi \epsilon \Psi^{K+1}} \left[\max_{i=1,...,N} \{\psi_i C_i\} \right], \quad (13)$$

which in turn can be expanded and ultimately yields three possible outcomes, i.e.,

$$C^{K+1}_{opt} = \min \begin{bmatrix} \Gamma^K_g C_g | \forall l S.T. (\Gamma^K_l + 1) C_l = \Gamma^K_g C_g \\ (\Gamma^K_g + 1) C_g \\ (\Gamma_h + 1) C_h | \forall h S.T. (\Gamma^K_h + 1) C_h > \Gamma^K_g C_g \end{bmatrix}, \quad (14)$$

where $\Gamma^K_g C_g = C^K_{opt}$, which can be rewritten as

$$C^{K+1}_{opt} = \max_{i=1,...,N} \left[\{(\Gamma^K_i + 1) C_i\} \bigcup \{(\Gamma^K_j + 0) C_j | \forall j \neq i\} \right], \quad (15)$$

where $(\Gamma^K_i + 1) C_i$, $i = \text{argmin}((\Gamma^K + 1)\mathbf{C})$.

Now for the solution determined by the algorithm, we can write:

$$C^{K+1}_{algo} = \max\{X^{K+1}_i C_i | i = 1, ..., N\}, \quad (16)$$

which in turn can be simplified to

$$C^{K+1}_{algo} = \max_{i=1,...,N} \left[\{(X^K_i + 1) C_i\} \bigcup \{(X^K_j + 0) C_j | \forall j \neq i\} \right], \quad (17)$$

which, combined with Eq. (15), returns $C^{K+1}_{algo} = C^{K+1}_{opt}$. Thus, according to Theorem IV.1, the $K+1$ tasks assignment determined by the algorithm is guaranteed to be optimal, which concludes the proof. □

V. Numerical Results

Using the testbed described in Section II, we compared the task distribution found using the algorithm presented in Section IV-B with both the greedy and uniform task distribution algorithms described previously. Each trial was started after the correct task distribution, if applicable, was found and was stopped when the last result of the computation was received. We note that, when optimizing for energy, the optimization problem presented in Section IV can be simplified to minimize the total energy consumption of the task assignment. Nevertheless, the cost-optimal task distribution presented in Section IV minimizes the maximum energy consumption required for assigning tasks to each device, thus fairly distributing the tasks throughout the network.

A. Performance Evaluation

For all the results presented in this section, we consider a homogeneous network and define the cost vector $\boldsymbol{C} = [C_1, C_2, \ldots, C_N]$ of the optimization problem in Eq. (3) such that $C_i = D_i(t, c, r)$, where $t = 13$B, $r \in \{10\text{KB}, 100\text{KB}, 1\text{MB}\}$ and c is determined so that $e_i(c) \in \{10\text{ms}, 100\text{ms}, 1\text{s}, 10\text{s}\}$. While we do not include the energy consumption at each device (i.e., E_i in Eq. (2)) in the definition of the cost vector $\boldsymbol{C}$, when our algorithm finds multiple task assignments that result in the same delay cost, we select the task assignment that minimizes the overall network energy consumption. Finally, we note that different definitions of the cost vector $\boldsymbol{C}$ are also possible. As an example, the cost C_i defined above can additionally include a level of reliability $R_i \in [0, 1]$, thus becoming $C_i = D_i(t, c, r)/R_i$.

1) Offloading Results: In order to validate our model, in Figures 2-4 we plot the time required to compute a set of 50 tasks as a function of different computation/communication time ratios, for both an Android implementation of the task distribution as well as a simulation of the task distribution using the model described in Section III. In these figures, the dashed lines represent the simulated performance of the task distribution obtained through the iterative algorithm described

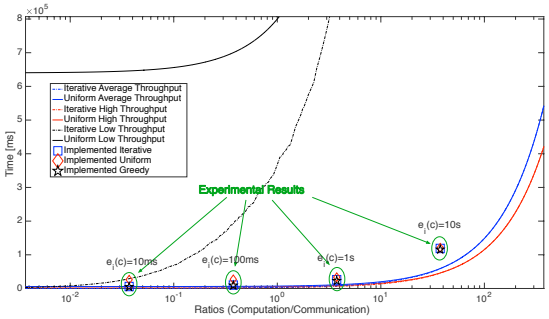


Fig. 2. Implementation and Simulation results for different computation/communication ratios. Result size is 1 MB.

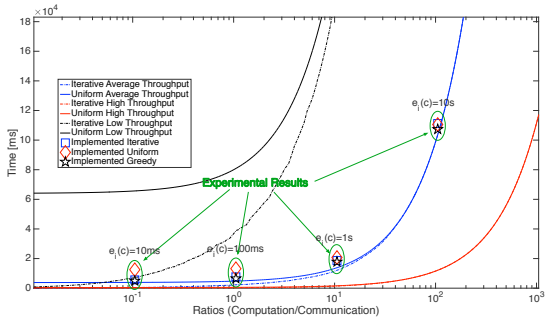


Fig. 3. Implementation and Simulation results for different computation/communication ratios. Result size is 100 KB.

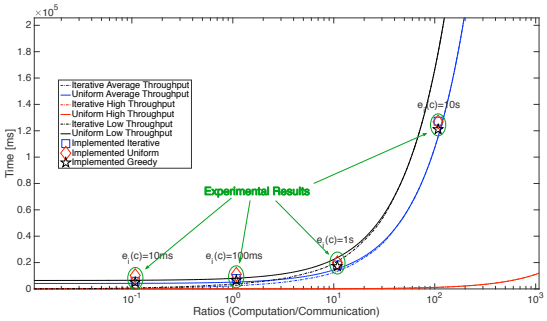


Fig. 4. Implementation and Simulation results for different computation/communication ratios. Result size is 10 KB.

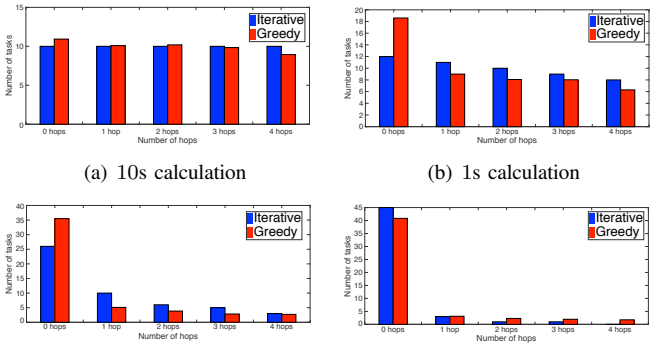
in Section IV-B, while the continuous lines represent the simulated performance of a uniform task distribution for different values of transmission throughputs. In particular, we set $T_l^{\text{tx}} = T_{\overline{l}}^{\text{tx}} \in \{72\text{Mbps}, T_r^{avg}, 0.5\text{Mbps}\}$ in Eq. (1), where T_r^{avg} is the average experimental throughput measured for the different data size r (i.e., $T_{10\text{KB}}^{avg} = 0.87$Mbps, $T_{100\text{KB}}^{avg} = 8.4$Mbps and $T_{1\text{MB}}^{avg} = 30$Mbps). The squares, diamonds and stars, instead, represent the results obtained by implementing the Iterative, Uniform and the Greedy task distribution policies in our real network of Android devices, respectively.

The results in Figures 2-4 show that when the computation takes about 40 times longer than the communication, a uniform task distribution provides the same performance as the optimal task distribution. Below this point, there are clear benefits in using the task distribution found with our iterative algorithm, with gains that become more evident when the time spent computing is comparable to the communication time. When the communication takes significantly longer than the computation, the optimal task distribution can complete the set of tasks about twice as quickly as the uniform task distribution (i.e., result size of 1MB and computation time of 10ms). Moreover, these results validate the model presented in Section III, and show that the simulated results can provide a good approximation of the implementation results so long as the appropriate average link throughputs are used. Hence, our iterative algorithm can be used to explore the performance of computation offloading in multi-hop ad hoc networks.

The simple Greedy task distribution described in Section II is able to adapt to the instantaneous variations in computational time (due to, e.g., operating system operation unrelated to the actual task execution), as well as to the channel impairments that can severely affect the actual throughput of the communication links. As a result, the Greedy task distribution is able to attain performance very close to the simulated performance of the optimal task assignment with the high WiFi Direct throughput. When minimizing the total completion time, thanks to its adaptability to the instantaneous system variations, the Greedy task distribution outperforms the implementation of the optimal task assignment, thus making it the de-facto choice for real device implementations.

To gain further insight into the differences between the Greedy task distribution and our iterative algorithm, in Figure 5 we compare the average task assignments that were used in each case. As can be seen in Figure 5(a), both the iterative and Greedy task distribution schemes start assigning tasks in a uniform way when the computation time (10s) is much longer than the time spent communicating. When the situation is reversed (i.e., communication is much longer than the 10ms computation), instead, in some cases no tasks are actually assigned to the furthest node (see Figure 5(d)). Overall, these results show that the throughput used to compute the optimal task assignment is over estimated, which is particularly evident by the fact that the device that is generating the tasks (i.e, the device at 0 hops), is most of the time computing fewer tasks than when using the Greedy approach.

2) Benefits of Offloading to Multi-hop Neighbors: The results presented in the previous section show that a Greedy task assignment can adapt to changing computation and communication environments and hence can return all the tasks in the least amount of time. However, our iterative algorithm provides a task distribution that is close to that provided by the Greedy algorithm. Additionally, the results in the previous section show that the simulated results match the implementation results and, as a consequence, simulations based on the model from Section III can be used to provide insight into the performance of the system. Thus, in what follows we use the analytical model and the iterative algorithm described in Sections III-IV to further explore the gains that can be achieved by extending the distributed computation to



(a) 10s calculation (b) 1s calculation

(c) 100ms calculation (d) 10ms calculation

Fig. 5. Task distribution for the Greedy and Iterative approach with result size fixed to 1 MB.

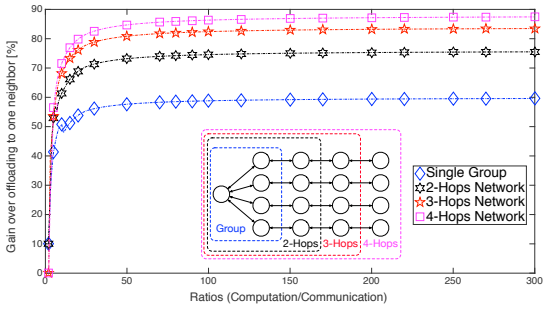


Fig. 6. Simulated measurement indicating the percent gain over offloading to only a single neighbor.

all the available network resources.

In particular, in order to highlight the benefits of offloading the computation to all the mobile devices in a network, in Figure 6 we compare the gain in time to complete all 50 tasks relative to a task distribution that only utilizes one additional device (i.e., single-hop task distribution as considered in the literature [9]–[11]) with: a) offloading the computation to all of the nearest neighbors of the device that is generating the tasks (i.e., Single Group), and b) offloading the computation to a multi-hop network (i.e., 2-hops, 3-hops and 4-hops network). As shown in Figure 6, in this simulation we consider a tree-like network with the root node generating the tasks having four children nodes, and each subsequent child servicing one additional node. This network extends for 2, 3 or 4 hops, resulting in a total of 4 devices in the source node's group that can be used for group task distribution, and 8, 12 and 16 devices that can be used for network task distribution.

Finally, Figure 6 clearly shows the benefit of offloading to a multi-hop network. In particular, offloading to the larger network provides up to 30% faster computation time than offloading the tasks only to the first group, and a gain of up to 88% against offloading to only a single device (as is currently supported in the literature).

VI. Conclusions

In this paper we explored the benefits of enhancing mobile to mobile computational offloading in multi-hop cooperative networks. By implementing a multi-hop computational offloading system, we were able to implement different task distribution algorithms and verify the accuracy of an analytic model and implemented different task distribution strategies. Using this model, we were able to show the overall benefit of enabling offloading to multi-hop neighbors in a network, which can be quite large when the time to compute is much higher than the time to communicate the data. In our future work, we will explore the inter-relation between relay nodes and nodes that are assigned tasks. In this way, we can avoid routing data through computation bottlenecks, as well limiting the number of tasks assigned to nodes that are pertinent for facilitating communication.

References

[1] "Harris RF-3590-RT." [Online]. Available: http://rf.harris.com/capabilities/c4isr/ruggedizedtablet.asp

[2] "MIL-STD-810G." [Online]. Available: http://everyspec.com/MIL-STD/MIL-STD-0800-0899/MIL-STD-810G_12306

[3] M. T. Flynn, M. F. Pottinger, and P. D. Batchelor, "Fixing intel: A blueprint for making intelligence relevant in afghanistan," DTIC Document, Tech. Rep., 2010.

[4] 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 *Proc. of ACM MobiSys*, 2010, pp. 49–62.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. of ACM EuroSys*, 2011, pp. 301–314.

[6] A. Khan, M. Othman, S. Madani, and S. Khan, "A survey of mobile cloud computing application models," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 1, pp. 393–413, First 2014.

[7] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proc. of ACM MobiSys*, 2007, pp. 272–285.

[8] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Sept. 2009.

[9] N. Fernando, S. Loke, and W. Rahayu, "Mobile crowd computing with work stealing," in *Proc. of NBiS*, Melbourne, Australia, Sept. 2012.

[10] T. Penner, A. Johnson, B. Van Slyke, M. Guirguis, and Q. Gu, "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," in *Prof. of IEEE GLOBECOM*, Dec. 2014, pp. 2801–2806.

[11] C. Funai, C. Tapparello, H. Ba, B. Karaoglu, and W. Heinzelman, "Extending volunteer computing through mobile ad hoc networking," in *Prof. of IEEE GLOBECOM*, Dec. 2014, pp. 32–38.

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

[13] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for vehicle-based disruption-tolerant networks." in *Proc. of IEEE INFOCOM*, Apr. 2006, pp. 1–11.

[14] B. Raj, T. Jayakumar, and B. Rao, "Non-destructive testing and evaluation for structural integrity," *Sadhana*, vol. 20, no. 1, pp. 5–38, 1995.

[15] Wi-Fi Alliance, P2P Task Group, "Wi-Fi Peer-to-Peer (P2P) Technical Specification, Version 1.2," Dec. 2011.

[16] C. Funai, C. Tapparello, and W. B. Heinzelman, "Supporting multi-hop device-to-device networks through WiFi direct multi-group networking," *ArXiv e-prints*, Dec. 2015.

[17] C. Casetti, C. F. Chiasserini, L. Curto Pelle, C. Del Valle, Y. Duan, and P. Giaccone, "Content-centric routing in Wi-Fi Direct multi-group networks," *ArXiv e-prints*, Dec. 2014.

[18] "Asus Nexus 7 (2013)." [Online]. Available: http://www.asus.com/Tablets_Mobile/Nexus_7_2013/

[19] D.-M. U. Derigs and U. Zimmermann, "An augmenting path method for solving linear bottleneck assignment problems," *Computing*, vol. 19, no. 4, pp. 285–295, 1978.

[20] R. E. Burkard and U. Zimmermann, "Weakly admissible transformations for solving algebraic assignment and transportation problems," in *Combinatorial Optimization*. Springer, 1980, pp. 1–18.