Task Allocation for Mobile Cloud Computing in Heterogeneous Wireless Networks

Zongqing Lu, Jing Zhao, Yibo Wu and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
E-mail: {zongqing, juz139, yxw185, gcao}@cse.psu.edu

***Abstract*—The ubiquity of mobile devices creates a rapidly growing market for mobile applications. Many of these applications involve complex processing tasks that are difficult to run on resource constrained mobile devices. This leads to the emergence of mobile cloud computing, in which cloud-based resources are used to enhance the computing capabilities of mobile devices. In this paper, we consider heterogeneous wireless networks in which multiple resource-rich computing nodes can be used as mobile clouds, and mobile devices can upload computation extensive tasks to these mobile clouds. The goal is to minimize the average task response time through determining whether to upload a task, and to which cloud the task should be uploaded. We formalize this task allocation problem, which is proved to be a NP-hard problem, and propose both offline centralized approach and online distributed approach to address this problem. Simulation results show that our approaches outperform others in terms of task response time in various scenarios.**

I. Introduction

The past few years have witnessed an explosive growth of mobile devices (e.g., smartphones, tablets). The ubiquity of mobile devices is accompanied by greater demand for mobile applications such as image processing, speech and optical character recognition, and natural language translation. Many of these applications involve complex processing that requires high computing speed, memory and battery lifetime. However, despite advances in manufacture technology, current mobile devices are still much more resource constrained compared with traditional desktop computers. In many cases, it is hard for these mobile devices to execute these computation extensive applications locally.

Recently, the combination of cloud computing and mobile computing leads to a new research area called *mobile cloud computing*, in which resource-constrained mobile devices use cloud-based resources to enhance their computing capabilities [1], [2]. For example, Satyanarayana [3] proposes a *cyber foraging* approach in which mobile devices upload tasks to some nearby resource-rich computing devices like traditional non-mobile computers. It has been implemented in many systems, such as Spectra [4], Chroma [5] and Scavenger [6]. However, resource-rich computing devices do not always exist in vicinity, which restricts the deployment of such cyber foraging approach. With access to Internet, some other approaches (e.g., MAUI [7] and CloneCloud [8]) upload tasks to the remote cloud infrastructure. In scenarios without Internet access, a group of mobile devices (connected by WiFi or Bluetooth) can form a mobile cloud to cooperatively run their tasks [9], [10], [11]. However, these works do not consider and exploit the heterogeneity of these mobile nodes in the network; i.e., some mobile nodes have low computation power whereas other mobile nodes have much higher computation power.

In this paper, we consider heterogeneous wireless networks in which multiple resource-rich computing nodes can be used as mobile clouds, and mobile devices can upload computation extensive tasks to these mobile clouds. For example, in battlefields, mobile clouds can be mobile nodes like computers inside tanks that have much higher computation power than mobile devices carried by solders, or they can be powerful computation devices at command center or other moving vehicles. The goal is to minimize the average task response time through determining whether to upload a task, and to which mobile cloud the task should be uploaded. The task response time may be affected by several factors such as processing delay, communication delay and queuing delay at mobile clouds. Always uploading tasks to the nearest mobile cloud may reduce the communication delay, but its queuing delay may be long if that mobile cloud has to process many requests. Thus, we have to carefully consider all three factors to minimize the task response time.

We first formalize the task allocation problem in heterogeneous wireless network, and prove that it is NP-hard. Based on the technique of linear programming relaxation [12], we propose an offline centralized approach to solve it. Then, we propose an online distributed approach in which the information of future tasks are not known in advance. The basic idea is for each mobile node to collect some network information and make task allocation decisions by themselves. Through extensive simulations, we investigate the tradeoff between performance and overhead, and demonstrate that our approaches can significantly outperform other approaches in terms of task response time.

The remainder of the paper is organized as follows. Section II provides an overview of our approaches and the problem formulation. Section III and Section IV describe the offline centralized algorithm and the online distributed algorithm, respectively. Evaluation results are presented in Section V and Section VI concludes the paper.

II. Overview

A. The Big Picture

We consider a heterogeneous wireless network formed by mobile clouds and mobile devices based on ad hoc connections. Since mobile devices usually have limited moving speed (e.g., they are carried by soldiers in battlefields), we assume their relative position to mobile clouds may not change too much during a short period of time, so the network can be modeled as a graph shown in Figure 1. Each mobile device generates a number of tasks over time and may upload them to some mobile clouds. It is possible that a mobile cloud is busy processing some existing tasks and has to put the new task into a waiting queue. In our model, each mobile cloud maintains a queue to buffer pending tasks which cannot be immediately processed. Tasks in the queue are processed on a first-come-first-serve basis, and the queuing delay may differ from one mobile cloud to another.

As shown in Figure 1, suppose a new task is generated by the mobile device encircled. The task can be completed by itself or uploaded to mobile cloud A or B. To upload the task, the task response time may be affected by several factors such as the processing delay at the mobile cloud, the communication delay from the mobile device to the mobile cloud, and the queuing delay at the mobile cloud. In this example, allocating the task to mobile cloud A incurs less communication delay than mobile cloud B, since there are fewer hops between mobile cloud A and the task generator. However, as mobile cloud A has more neighboring nodes (than mobile cloud B), mobile cloud A generally has higher task queuing delay. Thus, these three factors should be jointly considered to find the optimal task allocation.

In this paper, we address the task allocation problem under two settings: centralized setting and distributed setting.

1) Centralized Setting: Suppose all task information is known as a priori. Then, the task allocation problem can be reduced to the NP-hard scheduling problem [13] which schedules a number of jobs (tasks) on multiple nodes. We can design a centralized scheme based on the technique of linear programming relaxation [12]. Specifically, the problem can be formulated as an integer linear programming problem, and then the integer constraint (each variable must be 0 or 1) can be replaced by a weaker constraint (each variable belongs to [0,1]). The resulting relaxation is a linear program, which can be solved by GLPK [14] in polynomial time. Finally, we transform the solution to meet the integer constraint, and then derive the task allocation decisions.

2) Distributed Setting: In practice, the information of future tasks is not known as a priori, and we propose a distributed approach in which each mobile device makes allocation decisions locally. More specifically, when a new task is generated, the task generator makes a greedy allocation decision to minimize the response time of that task. To achieve this, the task generator has to collect the information about processing delay, communication delay and queuing delay of mobile clouds. Here the queuing delay of a mobile cloud is frequently updated whenever a new task is received, so each mobile cloud has to periodically broadcast its queuing delay to inform all task generators (mobile devices). By increasing the frequency of broadcast, the information collected by the task generator will be more accurate. Then, the task generator will most likely make good allocation decisions, but at the cost of more control message overhead. The tradeoff between performance and overhead will be investigated in the performance evaluation section.

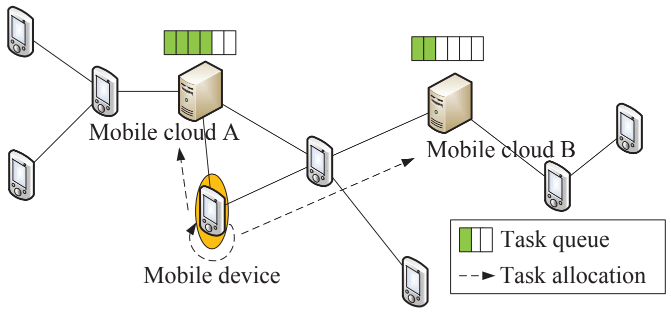


Fig. 1: Network Scenario

B. Problem Formulation

We first describe the network model and the task model, and then formulate the task allocation problem.

1) Network Model: The network is modeled as a graph $G(V, E)$, where each node $i \in V$ corresponds to either a mobile cloud or a mobile device, and an edge $e \in E$ represents the link between the two end nodes if they fall in the communication range. Each edge is associated with the communication link delay. In general, for any two nodes i_1 and i_2, the communication delay between them is modeled by the length of the shortest i_1-i_2 path in G where the edge weight is the communication link delay. Note that it can be obtained by Dijkstra's shortest path algorithm [15].

2) Task Model: There are a number of tasks generated in the network. Let $p_{i,j}$ be the processing delay of task j if it is processed by node i. A task can be processed by either its generator or a mobile cloud who has much shorter processing delay than the generator, and the processing delay by any other node (except the generator or mobile cloud) is infinity long. Let $T_{i,j}$ be the time if task j is completed by node i, then

$$T_{i,j} = t_j + 2d_{i,j} + q_i + p_{i,j}$$

where t_j is the generation time of task j, $d_{i,j}$ is the communication delay between node i and the generator of task j, and q_i is the queuing delay at node i.

Let T_j be the complete time of task j. Then,

$$T_j = \min_{i \in V} T_{i,j}$$

and the response time of task j is $T_j - t_j$.

We formulate the task allocation problem as follows.

Definition 1: ***Task Allocation Problem***
Given a set of tasks (denoted by U) to be completed by nodes in a heterogeneous wireless network, how to allocation tasks to minimize the average task response time $\frac{1}{|U|}\sum_{j \in U}(T_j - t_j)$? Note that $\frac{1}{|U|}\sum_{j \in U}(T_j - t_j) = \frac{1}{|U|}\sum_{j \in U} T_j - \frac{1}{|U|}\sum_{j \in U} t_j$ and $\frac{1}{|U|}\sum_{j \in U} t_j$ is a constant, so we simplify the objective to minimize the average task complete time $\frac{1}{|U|}\sum_{j \in U} T_j$.

III. Offline Centralized Approach

In this section, we first reduce the task allocation problem to the scheduling problem for proving the NP-hardness, and then propose a centralized algorithm based on linear programming relaxation.

A. Reduction to the Scheduling Problem

Definition 2: ***Scheduling Problem***
Given a set of jobs (denoted by U) to be completed by a number of nodes. Each job j is associated with a positive weight w_j, a nonnegative processing time and a nonnegative release time (i.e., the earliest possible start time) on each node. The jobs must be processed without interruption, and a node can process at most one job at a time. Let T_j denote the complete time of job j, how to schedule jobs to minimize the average job complete time $\frac{1}{|U|}\sum_{j\in U} w_j T_j$?

Theorem 1: The scheduling problem is NP-hard [13].

The task allocation problem can be reduced to the scheduling problem as follows: each task corresponds to a job, and the task arrival time corresponds to the job release time. Our goal is to minimize the average task complete time, which corresponds to minimizing the weighted job complete time in the scheduling problem by assigning unit weight to every task. Since the scheduling problem is NP-hard, we have the following theorem.

Theorem 2: The task allocation problem is NP-hard.

B. Algorithm Description

1) Linear Programming Relaxation: Let n denote the number of nodes and m denote the number of tasks. We formulate our problem as the following integer linear programming problem.

$$\textbf{minimize } \frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{h}\tau_{k-1}x_{i,j,k} \quad (1)$$

subject to

$$\sum_{i=1}^{n}\sum_{k=1}^{h}x_{i,j,k}=1 \quad (2)$$

$$\sum_{j=1}^{m}\sum_{k=1}^{h'}p_{i,j}x_{i,j,k}\le\tau_k, h'=1,2,\ldots,h \quad (3)$$

$$x_{i,j,k}=0 \text{ if } \tau_k<t_j+d_{i,j}+p_{i,j} \quad (4)$$

$$x_{i,j,k}\in\{0,1\} \quad (5)$$

Here we divide the time horizon into the following intervals: $[1,1]$, $(1,2]$, $(2,4]$, …, $(2^{h-2},2^{h-1}]$, where h is the smallest integer such that

$$2^{h-1}\ge\max_{i,j}(t_j+d_{i,j})+\sum_{j=1}^{m}\max_{i,p_{i,j}<\infty}p_{i,j}$$

That is, 2^{h-1} is the latest possible time when all tasks are completed. For conciseness, let $\tau_0=1$, $\tau_k=2^{k-1}$ $(k=1,2,\ldots,h)$, so the kth interval can be denoted by $(\tau_{k-1},\tau_k]$ $(k=1,2,\ldots,h)$.

In the program, $x_{i,j,k}$ indicates whether task j is scheduled to complete on node i within the interval $(\tau_{k-1},\tau_k]$. If $x_{i,j,k}$ is 1, τ_{k-1} is used as an approximation of the actual complete time to calculate the average task complete time (the objective function (1)). Constraint (2) ensures that only one node is scheduled to complete each task. Constraint (3) ensures that the complete time requirement indicated by $x_{i,j,k}$ is satisfied. Constraint (4) ensures that each task j that completes by time τ_k on node i must have $t_j+d_{i,j}+p_{i,j}\le\tau_k$.

Since the program has an integer constraint (Constraint (5)), there is no polynomial time algorithm to find the solution. Thus, we relax Constraint (5) to Constraint (6) and use GLPK [14] to solve the linear program (1)-(4),(6) in polynomial time.

$$x_{i,j,k}\ge 0 \quad (6)$$

2) Rounding: Let $x^*_{i,j,k}$ denote the solution for the linear program (1)-(4),(6). However, $x^*_{i,j,k}$ may be fractional, and hence does not represent a feasible solution to the task allocation problem. We apply the rounding technique [16] to transform $x^*_{i,j,k}$ into integers by constructing a weighted bipartite graph $\mathcal{G}(\mathcal{U},\mathcal{V},\mathcal{E})$. One side of the bipartite graph has the tasks, i.e.,

$$\mathcal{U}=\{u_j|j=1,2,\ldots,m\}$$

The other side has the nodes, i.e.,

$$\mathcal{V}=\{v_{i,s}|i=1,2,\ldots,n;s=1,2,\ldots,s_i\}$$

where $s_i=\lceil\sum_{j=1}^{m}\sum_{k=1}^{h}x_{i,j,k}\rceil$, and the s_i vertices $\{v_{i,s}|s=1,2,\ldots,s_i\}$ correspond to node i.

For conciseness, let $x_{i,j}$ denote $\sum_{k=1}^{h}x_{i,j,k}$. The edges of the bipartite graph correspond to node-task pairs (i,j) such that $x_{i,j}>0$. More specifically, the edges connecting vertices $\{v_{i,s}|s=1,2,\ldots,s_i\}$ are constructed as follows:

1) Find the smallest index j_1 such that $\sum_{i=1}^{j_1}x_{i,j}\ge 1$.
2) For $j=1,2,\ldots,j_1-1$, if $x_{i,j}>0$, add an edge $(u_j,v_{i,1})$ with weight $x_{i,j}$.
3) Add an edge $(u_{j_1},v_{i,1})$ with weight $1-\sum_{i=1}^{j_1-1}x_{i,j}$. This ensures that the total weight of edges connecting $v_{i,1}$ is at most 1.
4) If $\sum_{i=1}^{j_1}x_{i,j}>1$, add an edge $(u_{j_1},v_{i,2})$ with weight $\sum_{i=1}^{j_1}x_{i,j}-1$.
5) In a similar way, continue adding edges to $v_{i,2},v_{i,3},\ldots,v_{i,s_i}$.

Based on the matching theory [17], there exists a matching in bipartite graph $\mathcal{G}(\mathcal{U},\mathcal{V},\mathcal{E})$ which matches all tasks in $\mathcal{U}$. We use the Hungarian algorithm [18] to find a maximum weighted matching whose total edge weight is the maximum among all matchings. If an edge $(u_j,v_{i,s})$ is in the matching, we set $\bar{x}_{i,j}$ to 1; otherwise, $\bar{x}_{i,j}$ is set to 0. This minimizes the difference between the fractional solution $x_{i,j}$ and the rounding result $\bar{x}_{i,j}$.

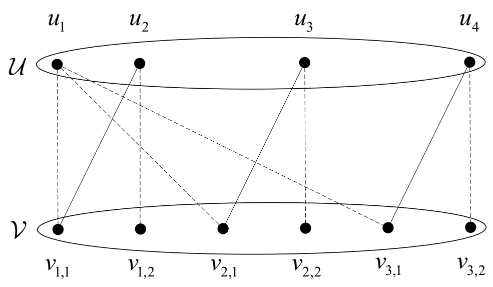


Fig. 2: An example bipartite graph. The weight of solid edge is 2/3, and the weight of dashed edge is 1/3.

Algorithm 1 Offline Centralized Algorithm

```
Input: task set $U$, node set $V$, and $t_j$, $d_{i,j}$, $p_{i,j}$ for $\forall i \in V, j \in U$
1: Calculate $h$ and $\tau_k$, $k = 0, 1, \ldots, h$
2: Solve linear program (1)-(4),(6) by GLPK
3: Construct weighted bipartite graph $\mathcal{G}$ based on the fractional solution $x_{i,j,k}$
4: Run Hungarian algorithm to find a maximum weighted bipartite matching $\bar{x}_{i,j}$
5: Output the task allocation decisions
```

We use an example to illustrate the aforementioned procedure. Suppose we are given the following $(x_{i,j})$.

$$(x_{i,j}) = \begin{bmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 0 \\ 1/3 & 0 & 0 & 1 \end{bmatrix}$$

The corresponding bipartite graph is shown in Figure 2. The maximum weighted matching is $(u_1, v_{1,1})$, $(u_2, v_{1,2})$, $(u_3, v_{2,1})$, $(u_4, v_{3,1})$, which corresponds to the following $(\bar{x}_{i,j})$.

$$(\bar{x}_{i,j}) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3) Deriving the Task Allocation Decisions: A task j is allocated to node i if $\bar{x}_{i,j} > 0$. If there are multiple tasks allocated to node i, they are scheduled in a non-descending order of the task arrival time.

The entire flow of offline centralized algorithm is summarized in Algorithm 1.

IV. Online Distributed Approach

In this section, we first describe how to collect the required information, and then propose a distributed algorithm.

A. Information Collection

Suppose a new task j is generated by node i_0. Node i_0 makes the task allocation decision based on the following information of each candidate node i (i.e., either node i_0 or a mobile cloud):

- *Communication delay ($d_{i,j}$):* As aforementioned in Section II-B1, $d_{i,j}$ can be obtained by running Dijkstra's shortest path algorithm on graph G. However, such centralized algorithm is difficult to implement in a distributed environment where node i_0 do not have the entire network information. This issue is addressed as follows. Similar to routing discovery in AODV [19], node i_0 discovers a number of paths to node i, during which the delay information of each path is also collected. Node i_0 records the path with the minimum delay, whose value is assigned to $d_{i,j}$.
- *Queuing delay (q_i):* As aforementioned in Section II-A, q_i is frequently updated whenever a new task arrives at node i. Thus, node i_0 has to be informed of the up-to-date q_i periodically. This will be achieved by the following information dissemination protocol.
- *Processing delay ($p_{i,j}$):* As in several prior work [7], [8], [9], $p_{i,j}$ can be obtained based on the node profile of node i and the execution profile of task j. Here a node profile includes the execution speed which is estimated by running benchmarks [20]; an execution profile includes the CPU cycles required for running the task. A node profile remains relatively stable over a long period of time. In case of an update, the node will share with all other nodes by the following information dissemination protocol.

Protocol 1 Information Dissemination Protocol

```
For each mobile cloud:
1: if node profile is updated then
2:     Broadcast message $(node\_id, profile, timestamp)$ to all neighboring nodes
3: end if
4: if $timestamp - last\_timestamp \geq T$ then
5:     If $queuing\_delay \neq last\_queuing\_delay$, broadcast message $(node\_id, queuing\_delay, timestamp)$ to all neighboring nodes
6:     $last\_queuing\_delay \leftarrow queuing\_delay$
7:     $last\_timestamp \leftarrow timestamp$
8: end if
For each mobile device:
1: if message $(node\_id, profile, timestamp)$ is received and timestamp is newer then
2:     Broadcast the message to all neighboring nodes and update $(node\_id, profile, timestamp)$ in record
3: end if
4: if message $(node\_id, queuing\_delay, timestamp)$ is received and timestamp is newer then
5:     Broadcast the message to all neighboring nodes and update $(node\_id, queuing\_delay, timestamp)$ in record
6: end if
```

The formal description of the information dissemination protocol is shown in Protocol 1, which consists of two parts: at mobile cloud side and at mobile device side.

- *At mobile cloud side:* Whenever the node profile is updated, the mobile cloud broadcasts the up-to-date node profile through message $(node\_id, profile, timestamp)$.

Algorithm 2 Online Distributed Algorithm at Node i_0

```
Input: task $j$ and the collected information
1: $V_0 \leftarrow \{i_0\} \bigcup$ the set of mobile clouds
2: for each node $i$ in $V_0$ do
3:     $T_{i,j} \leftarrow t_j + 2d_{i,j} + q_i + p_{i,j}$
4: end for
5: $i_1 \leftarrow$ the node whose $T_{i,j}$ is the smallest among the nodes in $V_0$
6: Send task $j$ to node $i_1$ along the predetermined shortest $i_0$-$i_1$ path
```

Here *node_id* denotes the ID which is unique to each node, and *timestamp* denotes the time when sending the message. Each mobile cloud also checks whether the queuing delay is different from the last recorded value every T time units. If there is an update, the mobile cloud broadcasts the up-to-date queuing delay to all neighboring nodes. The effect of T on network performance in will be investigated in Section V.

- *At mobile device side:* When receiving a message about node profile (or queuing delay), the mobile device checks whether the timestamp of the message is newer compared with the recorded information. If so, it updates the recorded information and re-broadcasts the message, in order to inform all other mobile devices about the update.

B. Algorithm Description

When a new task is generated, the task generator tries to make a greedy allocation decision to minimize the complete time of that task. More specifically, it uses the collected information to estimate the complete time for each possible task allocation. Then, the task is sent to the node with the minimum complete time. Here the generator includes in the task the predetermined shortest path (as aforementioned in Section IV-A) as a subfield, so other nodes in the network know to which node to forward the task. Through this source routing approach, the communication delay is minimized and thus the quality of task allocation is ensured. The online distributed algorithm is summarized in Algorithm 2.

V. Performance Evaluations

In this section, we evaluate the performance of the designed solutions through extensive simulations, which includes two parts: the first part compares the offline and online approaches, and the second part compares the online approach with other approaches based on the NS-3 network simulator.

A. Comparison between Offline and Online Approaches

1) Simulation Setup: To obtain the future task information required for the offline approach, we set up a simple simulation scenario and then compare it with the online approach. In our simulations, we randomly place 50 mobile devices and 5 mobile clouds in a 3000m $\times$ 3000m square area. The transmission range is 250m. If two nodes are within the transmission range, there is a link between them. The default wireless bandwidth is set to 24Mbps. As in previous study [9], large tasks can be divided into multiple uniform small tasks. The task size, i.e., the size of code and data, is set to 2Mb. For each task, the processing delay by a mobile device is 20s, while the processing delay by a mobile cloud is 2s. Here we do not investigate the scenarios that different mobile clouds have different computing speed, which will be left as future work. The tasks are generated in a homogeneous Poisson process. Due to the memoryless property of Poisson process, the task generation interval follows an exponential distribution.

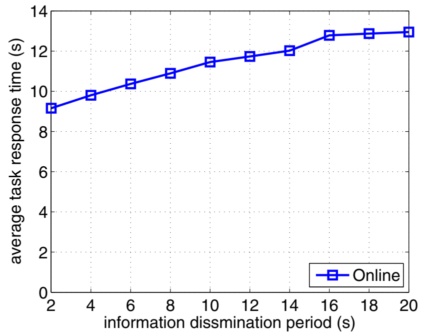
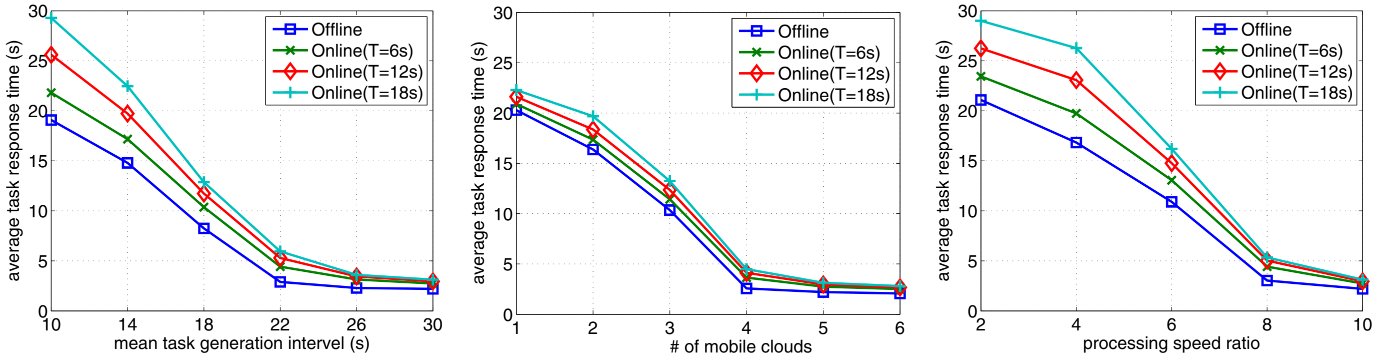


Fig. 3: Effect of information dissemination period

2) Tradeoff in Information Dissemination: Figure 3 shows the effect of information dissemination period T on the average task response time. Here the mean task generation interval is set to 18s. The average response time increases as T increases. As T increases, the accuracy of the information collected by the task generators decreases. Then, these nodes are less likely to make good allocation decisions, which leads to the increase of average response time. When T increases from 2s to 10s, the average response time increases by 25%. When T further increases from 10s to 20s, the average response time only increases by 13%.

3) Performance Comparisons: Figure 4 shows the performance of offline and online approaches in various settings. Since the computational complexity of the offline approach increases significantly as the number of tasks increases, the offline approach is evaluated using a trace of 500 tasks. The offline approach is compared with the online approach under different settings of information dissemination period T. Note that the mean task generation interval is set to 30s in default.

Figure 4a shows the effect of task generation interval on the average task response time. As shown in the figure, with the increase of task generation interval, the average task response time decreases for both offline and online approaches, because longer task generation interval results less queuing delay at mobile clouds. In addition, the online approach with smaller information dissemination period T has better performance. Figure 4b shows the effect of the number of mobile clouds on the average task response time. As expected, the increase of mobile clouds reduces the number of tasks processed at each mobile cloud, and thus decreases the average task response time. When the number of mobile clouds further increases, there is no queuing delay at mobile clouds and thus the task response time that only includes processing delay and communication delay becomes stable. Figure 4c shows the



(a) effect of mean task generation interval (b) effect of # mobile clouds (c) effect of processing speed ratio

Fig. 4: Performance of offline and online approaches when 500 tasks are generated in various settings, where the default processing delay at mobile devices is 20s, the default processing delay at mobile clouds is 2s, the default number of mobile clouds is 5, and the default value of mean task generation interval is 30s.

effect of processing speed ratio on the average task response time. Here we fix the processing delay by a mobile device to 20s, and denote the *processing speed ratio* as the ratio between the processing time by a mobile device and that by a mobile cloud. Similarly, the average task response time decreases when the processing speed ratio increases, because smaller processing delay leads to less queuing delay at mobile clouds.

As illustrated in Figure 4, the offline approach outperforms the online approaches; among the online approaches, smaller information dissemination period corresponds to better performance.

B. Comparisons between Online Approach and Other Approaches

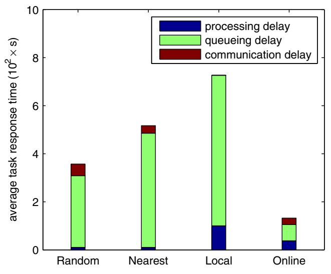
1) Simulation Setup: To simulate more realistic network environments, we use NS-3 simulator to more accurately capture the characteristics of heterogeneous wireless networks. The simulation setup is similar as before. We randomly place 50 mobile devices and 5 mobile clouds in 3000m $\times$ 3000m area. Nodes move according to the random waypoint mobility model with a speed randomly chosen between 2 and 5m/s for mobile devices and between 5 and 10m/s for mobile clouds, and there is no pause time. WiFi ad hoc mode is used for wireless communication with a rate 24Mb/s rate (802.11a) and a Friis loss model. The transmission power is set to 17 dBm. The information dissemination protocol is implemented with broadcast interval 6 seconds to collect the information required for the online approach. Tasks are generated in Poisson processes and the mean task generation interval for each mobile device is randomly chosen between 40s and 60s or between 60s and 120s or between 120s and 180s. The task size is set to 1Mb or 2Mb. Tasks are transmitted through TCP connections. The processing delay at mobile clouds is set to 10s, and the processing delay at mobile devices is set to 50s or 100s. Each simulation runs for one hour.

We compare our online approach with the following approaches:

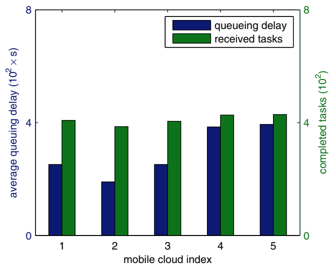
- *Random:* The task generator randomly selects a mobile cloud to allocate the task.
- *Nearest:* The task generator selects the mobile cloud with the minimum communication delay for task allocation.
- *Local:* All tasks are completed by the task generator itself (i.e., not uploaded to any mobile cloud).

2) Performance Comparisons: Figure 5 shows the performance of these approaches when the processing delay at mobile devices and mobile clouds is 100s and 10s, respectively, and the mean task generation interval is randomly chosen between 60s and 120s for each mobile device. Figure 5a compares the four approaches in terms of average task response time which consists of processing delay, queueing delay, and communication delay. Among them, *Online* performs the best and the task response time is much less than the other three approaches, while *Local* performs the worst. For all these approaches, the proportion of the queuing delay in the task response time is large, but the queuing delay of *Online* is much less than other approaches. *Random* has the highest communication delay, because *Random* allocates tasks to a randomly selected mobile cloud without considering the communication delay and hence the communication delay of the task can be very long when the task generators is far from the selected mobile cloud. Since *Online* processes some tasks locally (no communication delay), it has less communication delay than $Nearest$ on average and the processing delay is also between 10s and 100s. *Local* does not incur any communication delay since it only processes tasks locally. *Online* outperforms others in terms of average task response time because it considers the processing delay, communication delay and queue delay jointly while the other three approaches could not.

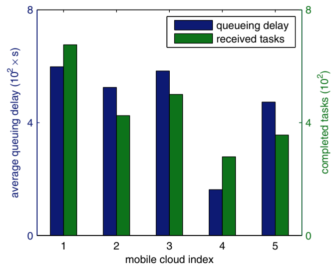
Figures 5b, 5c and 5d show the average queuing delay and the number of received tasks at each mobile cloud (also locally processed tasks for *Online*) for *Random*, *Nearest* and *Online*, respectively. From these figures, we can see *Random* allocates tasks more equally than *Nearest* and *Online*. *Nearest* allocates each task at the mobile cloud with the minimum



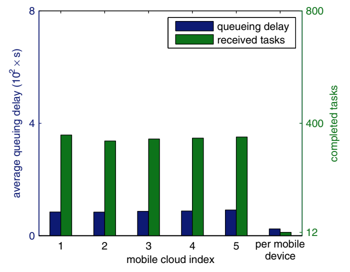
(a) average task response time



(b) queuing delay and # of tasks received at each mobile cloud for *Random*

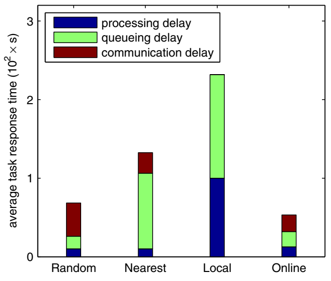


(c) queuing delay and # of tasks received at each mobile cloud for *Nearest*

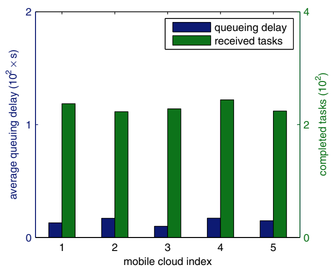


(d) queuing delay and # of tasks received at each mobile cloud for *Online*

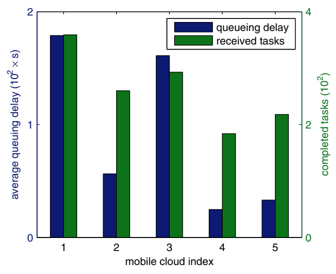
Fig. 5: Performance of *Random*, *Nearest*, *Local* and *Online* when the task size is 1Mb, the processing delay at mobile devices and mobile clouds is 100s and 10s, respectively, and the mean task generation interval is between 60s and 120s.



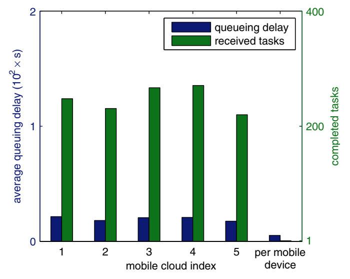
(a) average task response time



(b) queuing delay and # of tasks received at each mobile cloud for *Random*



(c) queuing delay and # of tasks received at each mobile cloud for *Nearest*



(d) queuing delay and # of tasks received at each mobile cloud for *Online*

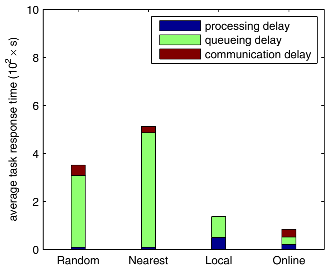
Fig. 6: Performance of *Random*, *Nearest*, *Local* and *Online* when the task size is 1Mb, the processing delay at mobile devices and mobile clouds is 100s and 10s, respectively, and the mean task generation interval is between 120s and 180s.

communication delay and thus tasks can gather at some specific mobile cloud if there are many mobile devices nearby, e.g., mobile cloud 1 in Figure 5c. *Online* allocates tasks evenly on each mobile cloud (about 350 tasks at each mobile cloud) and allocates 12 tasks at each mobile device on average, as shown in Figure 5d. Moreover, the average queuing delay at mobile devices is much less than that at mobile clouds (i.e., about 20 seconds at mobile devices and 80 seconds at mobile clouds.

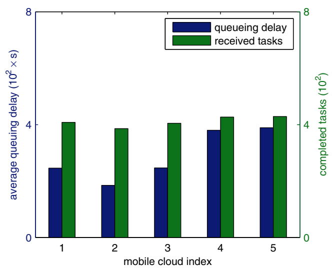
Figure 6 shows the performance of these approaches when tasks are generated at each mobile devices with the mean interval between 120s and 180s. Since less tasks are generated compared to Figure 5, the queueing delay at mobile clouds and mobile devices decreases and thus the average task response time of all these approaches drops significantly as shown in Figure 6a, where *Online* still outperforms others. For *Random* and *Nearest*, as shown in Figures 6b and 6c, the distribution of tasks at each mobile cloud is similar as in Figures 5b and 5c, however the number of tasks and the queuing delay at each mobile cloud decrease. For *Online*, as shown Figure 7d due to the reduction of queuing delay at mobile clouds, *Online* are more likely to allocate tasks at mobile cloud and very few tasks are allocated at mobile devices (less than 1 task per mobile device).

Figure 7 shows the performance of these approaches when the processing delay at mobile devices is set to 50s and the mean task generation interval is between 60s and 120s. As shown in Figure 7a, the task response time of *Local* decreases significantly due to the reduced processing delay at mobile devices, compared to Figure 5a. Since the change of processing delay at mobile devices does not affect the task allocation decisions of *Random* and *Nearest*, their performance is similar as in Figure 5. For *Online*, the average task response time also decreases due to the reduced processing delay and queuing delay. As shown in Figure 7d, *Online* allocates less tasks at mobile clouds and more tasks at mobile devices compared to Figure 5d. Due to the reduced processing delay at mobile devices, processing tasks locally is more likely to obtain better task response time when the queuing delay at mobile clouds is high.

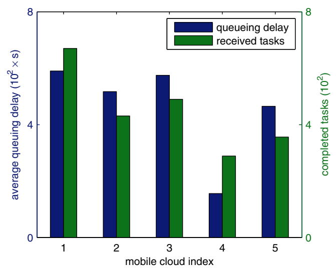
In Figure 8, we set the mean generation interval between 40s and 60s and keep other parameters same as in Figure 7. As shown in Figure 8a, the average task response time of all these approaches increases, compared to Figure 7a, due to the increased queuing delay incurred by more generated tasks. For *Random* and *Nearest*, although the generated tasks increase, the distribution of tasks among mobile clouds is similar with Figures 7b and 7c, respectively. When more tasks are generated, *Online* tends to process more tasks locally due to the increased queuing delay at mobile clouds and



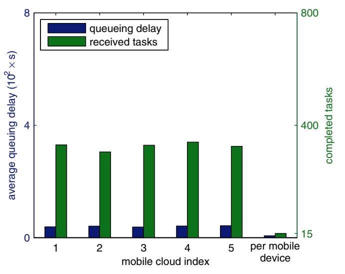
(a) average task response time



(b) queuing delay and # of tasks received at each mobile cloud for *Random*

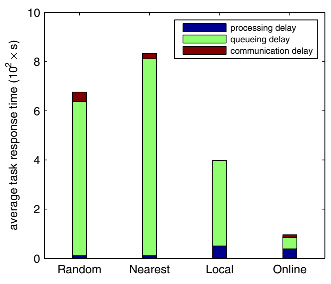


(c) queuing delay and # tasks received at each mobile cloud for *Nearest*

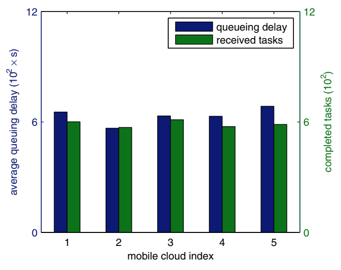


(d) queuing delay and # of tasks received at each mobile cloud for *Online*

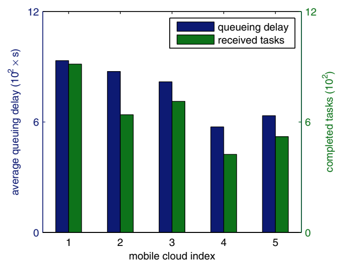
Fig. 7: Performance of *Random*, *Nearest*, *Local* and *Online* when the task size is 1Mb, the processing delay at mobile devices and mobile clouds is 50s and 10s, respectively, and the mean task generation interval is between 60s and 120s.



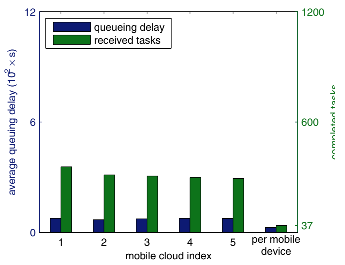
(a) average task response time



(b) queuing delay and # of tasks received at each mobile cloud for *Random*



(c) queuing delay and # of tasks received at each mobile cloud for *Nearest*



(d) queuing delay and # of tasks received at each mobile cloud for *Online*

Fig. 8: Performance of *Random*, *Nearest*, *Local* and *Online* when the task size is 1Mb, the processing delay at mobile devices and mobile clouds is 50s and 10s, respectively, and the mean task generation interval is between 40s and 60s.

the raised communication delay (incurred by the increased network traffic); i.e., 37 tasks on average are allocated at each mobile device in Figure 8d, while 15 tasks are processed locally at each mobile device in Figure 7d.

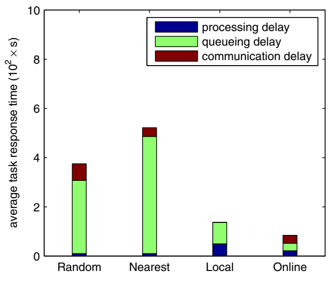
Finally, we investigate how the task size affects the performance of these approaches. Figure 9 shows the results when the task size is set to 2Mb, the processing delay at mobile devices is set to 50s, and the mean task generation interval is between 60s and 120s. Compared to Figure 7a where the task size is 1Mb, the communication delay of *Random* and *Nearest* increases due to the increase of task size, and thus the average task response time also increases, as shown in Figure 9a. Since the task size does not affect *Local*, its performance is the same as in Figure 7a. For *Online*, due to the increase of communication delay to mobile clouds, *Online* allocates more tasks at mobile devices and less tasks at mobile clouds than that in Figure 7d. Moreover, tasks are balanced among mobile clouds as before.

Figure 10 shows the results when the task size is 2Mb and the tasks are generated more frequently (the mean task generation interval is between 40s and 60s). As shown in Figure 10a, due to the increase of task size, the communication delay of *Random*, *Nearest* and *Online* increases, compared to Figure 8a. Moreover, comparing to Figure 9a, since more tasks are generated, the queueing delay increases for all these approaches. Due to the increase of task size, *Online* is more likely to allocate tasks at mobile devices, when the communication delay to mobile clouds is high, so as to achieve better task response time.

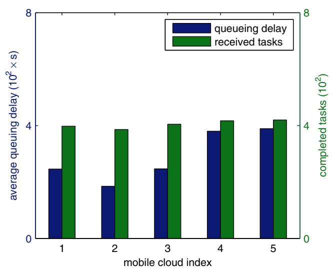
In summary, *Random* allocates tasks randomly at mobile clouds; *Nearest* allocates more tasks at the mobile clouds that have more mobile devices in vicinity; *Local* always allocates tasks locally at mobile devices; *Online* allocates tasks at mobile clouds and mobile device accordingly to minimize the task response time, considering the queuing delay, communication delay and processing delay together. Therefore, *Online* adapts to the variations of parameter settings and outperforms other approaches in all simulation settings.

VI. Conclusions

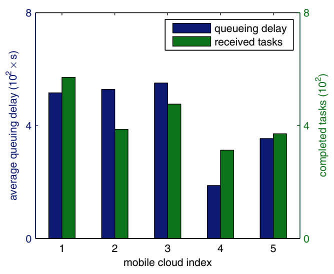
This paper studied the task allocation problem for mobile cloud computing in heterogeneous wireless networks, where multiple resource-rich computing nodes can be used as mobile clouds, and mobile devices can upload computation extensive tasks to these mobile clouds. The objective is to minimize the average task response time of all tasks considering communication delay, queuing delay and processing delay. To address this NP-hard problem, we first designed an offline centralized approach based on the technique of linear programming relaxation and then proposed an online distributed approach.



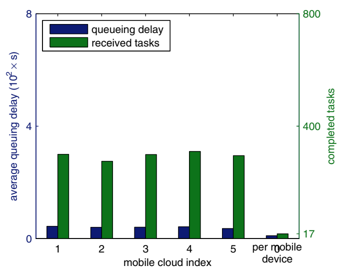
(a) average task response time



(b) queuing delay and # of tasks received at each mobile cloud for *Random*

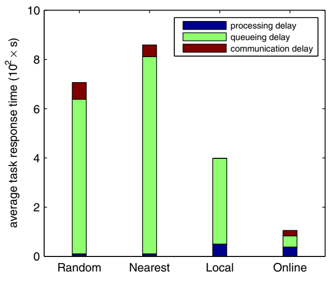


(c) queuing delay and # of tasks received at each mobile cloud for *Nearest*

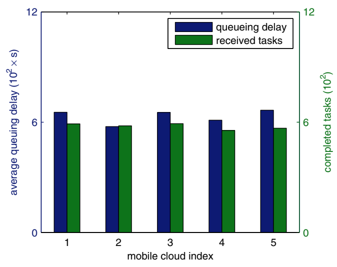


(d) queuing delay and # of tasks received at each mobile cloud for *Online*

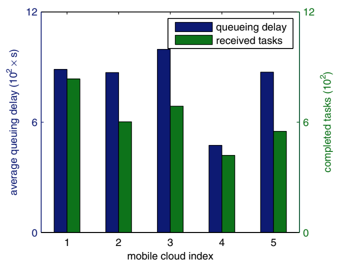
Fig. 9: Performance of *Random*, *Nearest*, *Local* and *Online* when the task size is 2Mb, the processing delay at mobile devices and mobile clouds is 50s and 10s, respectively, and the mean task generation interval is between 60s and 120s.



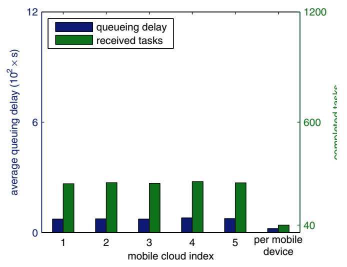
(a) average task response time



(b) queuing delay and # of tasks received at each mobile cloud for *Random*



(c) queuing delay and # of tasks received at each mobile cloud for *Nearest*



(d) queuing delay and # of tasks received at each mobile cloud for *Online*

Fig. 10: Performance of *Random*, *Nearest*, *Local* and *Online* when the task size is 2Mb, the processing delay at mobile devices and mobile clouds is 50s and 10s, respectively, and the mean task generation interval is between 40s and 60s.

Evaluation results show that our approaches outperform others in terms of task response time in various scenarios.

References

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[2] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 369–392, 2014.

[3] M. Satyanarayanan, "Pervasive computing: vision and challenges," *IEEE Personal Communications*, vol. 8, no. 4, pp. 10–17, 2001.

[4] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceedings of IEEE International Conference on Distributed Computing Systems*, 2002.

[5] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services*, 2007.

[6] M. D. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Proceedings of IEEE International Conference on Computer Communications*, 2010.

[7] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services*, 2010.

[8] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proceedings of ACM European Conference on Computer Systems*, 2011.

[9] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices," in *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2012.

[10] Y. Li and W. Wang, "Can Mobile Cloudlets Support Mobile Applications," in *Proceedings of IEEE International Conference on Computer Communications*, 2014.

[11] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proceedings of ACM International Conference on Mobile Computing and Networking*, 2013.

[12] L. A.Hall, A. S. Schulz, D. B. Shmoys, and J.Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Mathematics of Operations Research*, vol. 22, no. 3, pp. 513–544, 1997.

[13] J. K. Lenstra, A. H. G. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[14] GLPK: GNU Linear Programming Kit, "http://www.gnu.org/software/glpk/glpk.html."

[15] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[16] D. B. Shmoys and E. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, no. 1-3, pp. 461–474, 1993.

[17] L. Lovasz and M. D. Plummer, *Matching Theory*. American Mathematical Society, 2009.

[18] H. Kuhn, "The Hungarian Method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, pp. 83–97, 1955.

[19] C. E. Perkins and E. M. Royer, "Ad hoc on-demand distance vector routing," in *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, 1999*.

[20] *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures*. Intel White Paper.