International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Optimizing Energy Efficiency on Task Allocation for Cyber Foraging in a Transient Mobile Cloud System

Tiako Fani Ndambomve[a]*, Felicitas Mokom[b], Kolyang Dina Taiwe[c]

[a,c]*LaRI Lab, University of Maroua, P.O. Box 814 Maroua, Cameroon*
[a,b]*School of Information Technology, Catholic University Institute of Buea, P.O. Box 563 Buea, Cameroon*
[b]*IEEE Computational Intelligence Society Member,Association for Computing Machinery (ACM) Member*
[a]*Email: tiakofani@cuib-cameroon.net*
[b]*Email: fmokom@cuib-cameroon.net*
[c]*Email: dtaiwe@yahoo.fr*

Abstract

In this research, we address the essential problem of achieving energy-efficient task allocation, which is a vital building block of cyber foraging on a transient mobile cloud. The goal is to minimize the total energy consumption for collaborative task executions among mobile devices in a multi-hop mesh network constructed on a mobile agent-based framework. Accordingly, we propose an energy-efficient task allocation problem formulation that takes into account the required restrictions. Next, we develop an optimal task allocation solution based on the modification of the Kuhn-Munkres algorithm by leveraging on the structural properties of the problem. We further evaluate the effectiveness of the suggested task allocation scheme through numerical study on a simulated system. The simulation reveals a performance gain on energy consumption reduction over other widely used task assignment algorithms.

Keywords: Transient mobile cloud; Multi-hop mesh network; Energy-efficient task allocation; Kuhn-Munkres algorithm; Mobile agent-based framework.

Received: 6/27/2024
Accepted: 8/27/2024
Published: 9/5/2024

---

* Corresponding author.

1. Introduction

With their ever-evolving capabilities, mobile devices are now able to access an incredible amount of data and information via apps, email, computations and storage on-the-go, with the concept of Mobile Cloud Computing (MCC). The advent of mobile cloud computing has paved the way for quicker, more affordable solutions that provide security and flexibility. Huawei envisions that by 2025, the number of terminal devices connected to the Internet will be close to 100 billion, and a large majority of these devices will be mobile phones [1]. This reveals the prominent need for Mobile Cloud Computing as it enables users to work more effectively, given that mobile devices are still limited in compute and energy resources [2], so much so that they are unable to completely execute computation intensive applications on their own.

Among the various types of MCC are Transient Mobile Clouds (TMC) [3]. These are temporal clouds that enable nearby mobile devices to form an ad hoc network and advertise their capabilities as cloud services. This is achieved by leveraging the idle resources of nearby mobile devices (processing, storage, and communication), creating a distributed cloud infrastructure that can be used to deliver services to users [4]. These services can be offered to other devices in the temporary smartphones ad hoc network. One of the main advantages of transient mobile clouds is that they can be set up quickly and easily, without the need for a central cloud infrastructure or a fixed network. This makes them particularly useful in situations where traditional cloud computing may not be available or feasible, such as in remote locations, crowded areas with limited connectivity, or during disasters [9]. Meanwhile, their main disadvantage is the ephemeral nature of the cloud structure, given that it often changes due to the mobile devices entering and leaving the network [5]. Transient mobile clouds can be used to provide a variety of services, including temporal data storage, content distribution, and computation offloading. They can also be used to improve the performance of existing cloud services by providing additional resources or by caching frequently used data. Furthermore, because the mobile devices are close to one another in a geographically constrained environment, they are more likely to share interests and have social and context awareness, which is useful for a variety of applications [4]. This awareness is more difficult to use with distant clouds. By utilizing the collective resources of the group, devices are no longer constrained by their local hardware and software capabilities. TMC systems harness the ubiquitous nature of mobile devices along with their ever-increasing sets of capabilities in providing a rich computing platform. Overall, transient mobile clouds are an innovative approach to cloud computing that leverages the resources of mobile devices to create a flexible and dynamic cloud infrastructure.Among the various services provided by transient mobile clouds, Computation Offloading is a key activity. It seeks to increase the functionality of mobile systems by sending some of the computational work to distant devices in the cloud network. This process is carried out by assigning a part of the mobile devices' tasks, distributing them through the network and running them on the assigned devices [6]. We focus our interest on Task assignment, which is a well-studied problem with many proposed algorithms [7 - 16]. In these algorithms, each device provides a cost estimate for each task it can run. The aim of a task assignment algorithm is to find the assignment with the lowest overall cost in time and/or energy consumption. This will permit the requesting device to execute most of its tasks efficiently without depending on Internet resources. Therefore, designing efficient task assignment strategies is one of the ways to enhance the computation offloading experience of mobile users in the Transient mobile cloud systems.

2. Overview of Techniques Related to Transient Mobile Clouds

Transient mobile clouds have the potential to alleviate the internet connectivity delay present in most MCC solutions, and more precisely Mobile Edge Cloud Computing solutions, through the creation of an ad hoc network by nearby edge devices [17] and the subsequent offloading of some tasks (Cyber Foraging) to selected devices. Substantial study has been done in the field of transient mobile clouds as a result of the recent major increases in the numbers, kinds, and capabilities of mobile devices [20 - 25]. For this purpose, three aspects are considered in setting up these clouds: Network architecture, Code partitioning and offloading, Tasks scheduling mechanisms.

2.1 Network architecture

Transient mobile clouds currently have a number of restrictions that relate to their growth on a local network with a high number of devices and the control of the network instability brought on by the constant mobility of the devices. Given that a considerable quantity of mobile devices is needed to constitute an efficient cloud system, the authors in [26] argued that setting the devices into a multi-hop ad-hoc network will permit to have access to greater resources even in case the devices requested to serve are very distant from the requesting device. Terry Penner [20] designed a framework for Assignment and Collaborative Execution of Tasks on Mobile Devices in a Transient Cloud. That cloud utilizes the collective capabilities of the devices present, along with their social and context awareness, and that cannot be provided efficiently by the traditional clouds. In [24], they provided a real implementation on the Android platform using the Wi-Fi Direct framework. However, the work had the following limitations as per the network architecture:

- In Wi-Fi Direct, only one device (called the Group Owner) acts as a router, and all of the other peer devices that connect to it create a single hop network. Also, any device in a group can only be a client in another group. However, this limits the possibility of expansion of the network and the size of the network to be a maximum of one hop from the Group Owner.
- Secondly, because of the fact that every device must connect to the Group Owner, a significant disadvantage is that if the Group Owner leaves, the group is torn down and a new group must be established from scratch. This makes Wi-Fi Direct unsuitable as a basis for multi-hop networking.
- Thirdly, the group owner is the hub device of the network that is responsible for maintaining the network's state and it needs to have a stable connection with the client devices in the network to route tasks (code) and data.

Therefore, their solution may hardly work on a multi-hop ad-hoc (mesh) network (MANET) with a very unstable connectivity between devices, which is normally the typical network for a realistic Transient Cloud [18]. Meanwhile, it is crucial to consider this network topology and its characteristics to produce an application framework which responds to the needs and the features of the TMC structure.

Having adopted a multi-hop mesh network for the transient cloud, there is need to adapt to it a protocol that require minimal memory for routing table, less computing resources and generate less protocol control overhead, given the limited resources on mobile devices. The latest amendment of the 802.11 standard [28] provides the

Independent Basic Service Set (IBSS) mode that can be used for ad-hoc networking. It is commonly referred to as ad-hoc mode because it does not require any infrastructure to be in place. It can be used as a basis for mesh networking. In this mode, all nodes play similar roles, and any node can communicate directly with any other node within the network, as those nodes are also set to the IBSS mode, share the same Service Set Identifier (SSID) and are within its radio range. The IBSS mode itself, however, does not offer multi-hop capabilities. There is no provision for path discovery and selection, nor for relaying packets to nodes out of the radio range of the sender. In IBSS-based ad-hoc networks, these functions must be accomplished by an additional protocol, like BATMAN [27], usually at the network layer. Among all the existing protocols for MANETs, the Better Approach to Mobile Ad-hoc Network protocol (BATMAN) is used more often, reason being that it does not need to maintain full path to the destinations. Each node only collects and maintains the information about the best next hop towards all other nodes in the network. Every node collects this information through hello packets broadcast periodically. This makes the protocol suitable for storage constrained devices. Also, since this protocol depends only on hello packets to know the availability of nodes and does not broadcast topology change messages, the control overhead is low. For the purpose of this work, we have built a TMC simulation based on the BATMAN protocol on Repast Symphony. In the Figure 1 for example, we considered 14 devices that are constituted into a TMC network through the BATMAN protocol. Here, a device has access to other devices via a direct connection or a multi-hop (indirect) connection, and every connection may have a weight w_{ij} that represents the communication cost between the devices D_i and D_j.

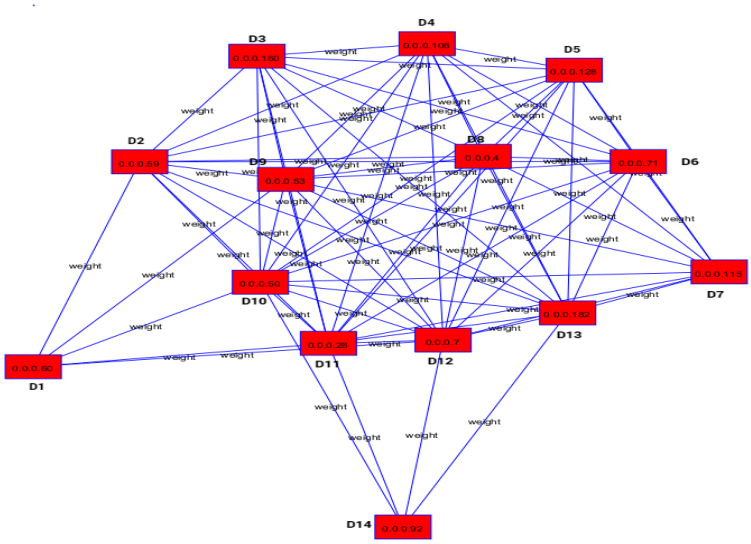


Figure 1: Devices constituting a Multi-Hop Mesh Network using the BATMAN protocol

2.2 Code partitioning and offloading

Code partitioning and offloading constitute another issue on Transient mobile clouds [17] and they need to be properly addressed to match the requirements of such clouds set up on a multi-hop mesh network, otherwise the distributed computing process will be hardly effective for the Transient mobile Cloud.

2.2.1 Code partitioning

With respect to code partitioning, there are several approaches. Bowen and his colleagues. [3] proposed a taxonomy of these approaches based on their strategies to break out application execution components into non-offloadable and offloadable modules. A partitioning algorithm could be classified into either of two categories [29, 30]: (1) static: when it is fixed at development phase based on the static analyzer and dynamic profiling; and (2) dynamic: when it includes runtime information from different profilers, log files, and the interaction among components of the running application. Therefore, static approaches are often based on pre-specified annotation where developers annotate the candidate methods for offloading and leave the remaining to the framework. While the resulting partitions of static algorithms do not change in all executions, those of dynamic algorithms could be updated before each execution to adapt to environmental conditions and optimize offloading objectives. Because of that, though dynamic algorithms may have high overhead during the offloading process, we still opt for using a dynamic approach with Operating System (OS) partitioning considering the various and changing capabilities of the devices in the transient cloud and instability of the network structure.

2.2.2 Code offloading

One of the most typical methods to analyze and model code offloading for an application running on a smartphone is to use its call graph as generated by the OS, which is a Directed Acyclic Graph (DAG), that represents the relationship between the computational components of that application [6, 31, 32]. Normally, the nodes of the graph represent the procedures/functions of a program and the directed edges represent the communication/invocations dependency between them, as shown in Figure 2. Here, two dependency levels can be identified. A module cannot be executed until the execution of its parents is completed. On the other hand, all the modules in the same dependency level can be executed in parallel. The operating system on the smartphone generates this DAG for the application with each of the nodes (processing block) being light enough to be easily transported through the MANET and to be quickly executed. Furthermore, advanced analysis on the DAG can permit to group the modules into two parts: one part which is to run locally and the other part which is to be offloaded to the TMC devices. For example, some processing blocks with special hardware needs, such as camera, accelerometer, etc. should be running locally, while the blocks that need more of computation are offloaded.

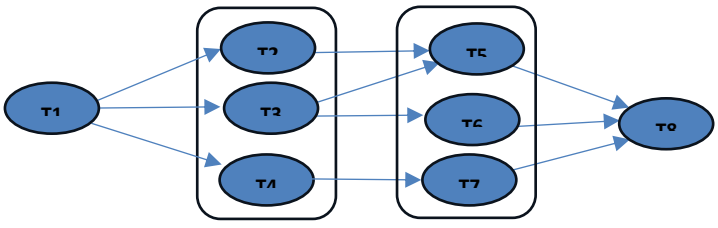


Figure 2: DAG with two identified dependency levels

In [26], the study presents a transitory cloud-based framework that makes use of multi-agent systems to enable dynamic code offloading as well as to make it easier for an individual code block to be packaged in a Mobile Agent (MA), to travel around and be executed on one or more devices in the network. In Figure 1, under various

conditions, D_1 may offload a task for execution to D_7 by packaging it in a MA. That agent can hop through D_{11}, D_{12}, D_{13} to get to its destination; this depending on whether a route to D_7 was recorded during the establishment of the network. Upon arrival at D_7, if the task in the MA does not complete its execution there, it is repackaged in a MA alongside its current status and offloaded again to another device. It continues like that till the task is completed and returns to its origin. Therefore, as the number of hops and transitions of the MA increases, the communication cost and the energy consumed increase likewise. We adopt this multi-agent system strategy as presented in [26]. When designed and implemented, these intelligent mobile agents would work alone or with others, to successfully adapt to the frequent arrival and departure of devices in and out of the network [33, 34].

2.2.3 Offloading Decision

In considering a user application as a set of *M* processes forming a DAG, all the processes in the same dependency level can be executed in parallel. As such, the process that takes most time for execution ultimately gives the level execution time as it covers the execution time of all the processes in that level. Hence, greater number of processes in a dependency level decreases the overall execution time of the application. The total execution time of the application is the summation of execution times of all the levels and data transmission times between the modules. To determine the appropriate devices for the offloaded execution, given that the total execution time of the offloaded processes helps to define the performance of the TMC System, the receiving devices must have to fulfill the selection criteria. These criteria consider the computation power, energy, data rate, the memory usage and the disk usage and associativity time [39]. In several approaches [35,38], a device will be selected as a *receiver* if it can execute the task in less time compared to the *sender* device and it maintains a minimum data-rate with the *sender*. Moreover, it must have sufficient energy to execute the task and send the result back to the *sender*. During this execution and communication time, the two devices remain connected. However, this will not be suitable for this transient mobile cloud given that the devices are not necessarily directly connected to each other in the multi-hop mesh and they may move out of the network due their mobility.In our approach based on [26], each device, through its MAZ, is responsible for maintaining the list of resources available on other devices it knows in the whole network (by a direct or indirect link given that it is a multi-hop mesh network). Some resources that are monitored here are the *network bandwidth*, the *processor usage*, the *queuing delay*. In our framework, each device monitors these three resources to make sure it has the latest usage information of other devices. In this regard, for the amount of energy, we consider a threshold value that needs to be available on a mobile device before it can receive an offloaded task, so that the device can still run its own compulsory activities (the minimal level of energy present on a device that exempts it from participating in the Transient cloud). In the same light, we may also consider a threshold for the amount of memory. With this information, each mobile device can make allocation decisions locally. Also, the queuing delay of a mobile device is frequently updated whenever a new task is received, so each mobile device has to periodically broadcast its queuing delay to inform all the devices of its status. By increasing the frequency of broadcast, the information collected by the tasks' generators will be more accurate. Then, the MAZ on the *sender* will most likely make good allocation decisions, but at the cost of more control message overhead.

The authors in [12] state that for offloading to be effective, the computation part of program must be significantly larger than its communication part. This implies that when one uses cyber foraging to improve response time or

energy consumption, the offloading mechanism would be more effective for applications requiring more computation than communication. We illustrate a simple flowchart of cyber foraging on a requesting device in our TMC (See Figure 3). Therefore, large tasks requiring higher execution times make offloading more effective because the benefits of computation on a more powerful and faster surrogate outweigh the cost of communication. However, the constraints on mobile devices are due to their mobility as it restricts the size of tasks to be offloaded [18, 26, 32]. That is, if a task is too large to complete its execution before leaving the area in the networking coverage of a surrogate, offloading becomes very complex and time-consuming solutions such as check pointing and process migration would often be used. Therefore, a suitable offloading approach must specially consider the mobility nature of mobile devices and manage a trade-off between mobility and task size. We introduce a hand-over strategy to further manage this trade-off. In this scenario therefore, the user might prefer to lower the bar of latency and constraints to favor general/local energy savings, as it is the case in our work. Also, in the TMC system we propose, the time and the energy spent for one task is the sum of the time and energy on each transition point (surrogate device) added to the communication time between transited surrogates in the network.

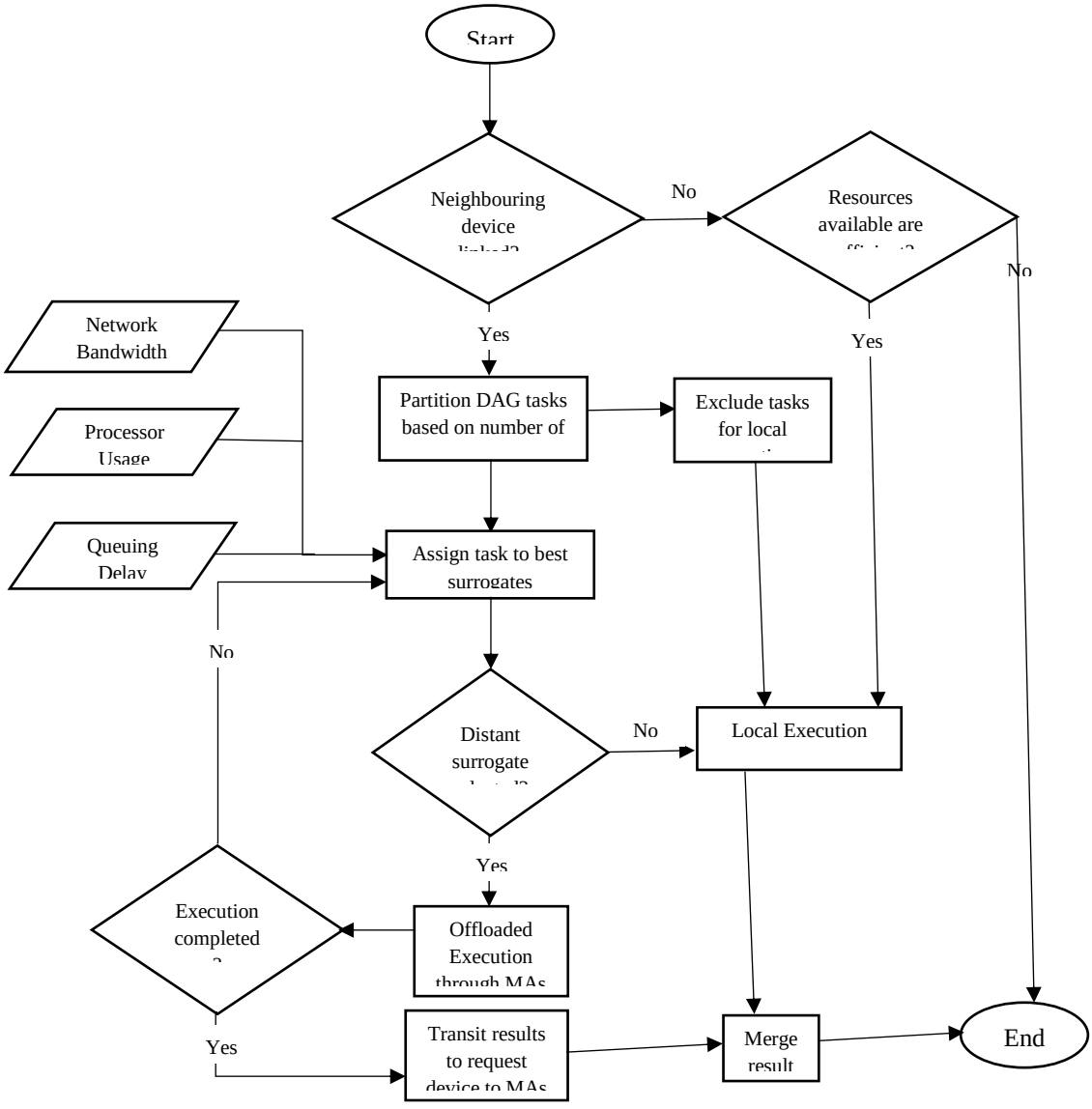


Figure 3: Flowchart of cyber foraging on a requesting device in the TMC system

2.3 Tasks scheduling mechanisms

When energy efficiency is taken into account, task scheduling becomes much more complicated but crucial. Execution overhead and scalability are major concerns in current research on energy-efficient task scheduling [7]. Because the cloud nodes need to respond quickly to real-time tasks and ensure the mobility management of nodes at the same time, the optimal task scheduling strategy needs to be selected to meet the low latency requirements of users. Also, based on the needs and the features of the TMC system, the algorithm for task assignment and execution should be derived to align with the framework described in [26], while taking into account some constraint conditions such as response time deadline, dependency between task, data transmission and energy cost. Actually, the objective of Task allocation scheme is to select an appropriate node for execution of a task as aiming to reduce time and/or energy consumption. The task allocation process usually consists of two steps: **(1)** Selection of nodes on which estimated task execution time is less than the task deadline and **(2)** Given the list of nodes selected in (1), allocate a task to a node on which estimated energy consumption is minimum. To estimate task execution time and energy consumption, models proposed in [31, 32, 40, 41] have been readjusted to the TMC context in view of the above problems. We present them in **Section 3** below.

In a nutshell, as presented in Figure 4, the user application plane builds the various tasks partitions, the framework plane packages the tasks in MAs and allocates them and finally the MAs are distributed to various devices over the network. Results of tasks are received via the planes in the reversed order, back to the user application.

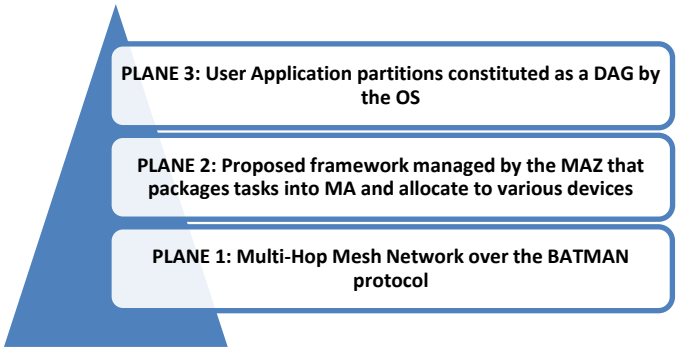


Figure 4: Stack Approach to tasks allocation and Execution on a TMC system

3. Formulation of the Task Execution Problem

As a global perspective, we first introduce the model to describe the resources of a TMC device for mobile task processing as follows. We consider the connectivity graph $G1 = \{N, E\}$, where the set of devices N is the vertex set and $E = \{(i,j) : e_{ij} = 1, \forall i, j \in N\}$ is the edge set where $e_{ij} = 1$ if devices i and j can establish a direct link between themselves in the multi-hop mesh network.

3.1 Evaluating Energy Consumption

The computation cost constitutes the energy required by a given task to process on various devices. The communication cost is the energy required for transferring the agent, its data and control information between the

mobile devices in the TMC System and then, the resulting data sent back to requesting device. Therefore, we have the considerations below.

For a given mobile node n_i that has been partitioned into tasks to execute either onsite or otherwise, we consider the following elements and their notations:

$\{N_i\} = \{N_i\}_A \cup \{N_i\}_U$. This is the set of n mobile nodes (devices) known by $\mathbf{n_i}$ in the TMC.

The set of mobile nodes can be seen as a union of the **sender node** n_i, the set $\{N_i\}_A$ of devices to which tasks of n_i have been allocated and the set $\{N_i\}_U$ of devices unused by tasks of n_i.

$\{T_i\} = \{T_i\}_o \cup \{T_i\}_L$ with $\{T_i\}_o = \{T_i\}_{par} \cup \{T_i\}_{Seq}$. It is the set of m tasks of a given node n_i.

This set can be seen as the union of the set $\{T_i\}_L$ of tasks that are executed locally and the set $\{T_i\}_o$ of tasks that offloaded through agents. This later set is grouped into tasks that are executed parallelly in $\{T_i\}_{par}$ and others that are executed sequentially in $\{T_i\}_{Seq}$ as denoted by the operating system.

The device n_i has its CPU working frequency W_i (the total computation capacity in CPU cycles per unit time). Let μ_i be the current load of node n_i (the percentage of currently occupied processing capacity) since device i may have some background load and/or run some unoffloadable tasks. Then the available processing capacity of n_i is $C_i = (1-\mu_i)W_i$. With this note, we have the considerations given in **Table 1**.

Table 1: Parameters considered for the TMC system

Parameter	*Description*
D_{ij}	*Data Rate from n_i to n_j over the network*
a_i :	*Processing Energy Consumption on n_i per time unit;*
b_i :	*Transmission Energy Consumption on n_i per time unit;*
$P_t(n_i)$:	*Transmission power of n_i;*
$P_r(n_j)$:	*Reception Power of n_j*
B_j :	*Average bandwidth on node n_i*
β:	*Factor of the throughput and packet loss*
	For a given $t_k(n_i)$ (task k of node n_i)
I_{ik} :	*Input data size of task k*
O_{ik} :	*Output data size of task k*
S_{ik} :	*Size of task k (amount of required computing)*
Q_{ik} :	***Queuing delay of task k***

3.1.1 Local Execution of a given task k on a node n_i

$$\text{Execution Time: } ET^{e}_{iik} = \frac{S_{ik}}{C_i} + Q_{ik} \qquad (1)$$

Energy used during Computation: $EC^{e}_{iik} = a_i ET^{c}_{iik}$ (2)

Time spent for data communication (given that the task may require some input data or send some output through the network):

$$DT^{d}_{ik} = \frac{I_{ik}+O_{ik}}{D_i} + \beta \quad (3)$$

Energy used during Data communication: $EC^{d}_{iik} = a_i DT^{d}_{ik}$ (4)

Total Energy Consumption for the execution of task k locally on a given node: $EC^{l}_{iik} = EC^{e}_{iik} + EC^{d}_{iik}$ (5)

3.1.2 Offloaded Execution of task k of node n_i in an agent

$N[t_k]$ is the ordered set of all the nodes n_j that the MA having task *k* goes through in order to complete its execution after leaving the node n_i. Therefore, $\{N\}_A = \bigcup_{k=1}^{n} {}^{N[t_k]}$. (6)

Supposing that to get to a node n_j, the MA hops through the nodes $\{n_{A1}, n_{A2}, \ldots, n_{Ak}\}$as determined by the BATMAN protocol, we have that the energy used for the data transfer of the agent having task *k* from n_i through various n_j nodes is:

$$EC^{d}_{ijk} = \sum_{j\in\{N[t_k]\}} \left\{ \left(\frac{p_t(n_{A1})+p_r(n_{A2})}{D_{A1,A2}} I_{ik} + \frac{p_t(n_{A2})+p_r(n_{A1})}{D_{A2,A1}} O_{ik} \right) + \cdots + \left(\frac{p_t(n_{Ak})+p_r(n_j)}{D_{Ak,j}} I_{ik} + \frac{p_t(n_j)+p_r(n_{Ak})}{D_{j,Ak}} O_{ik} \right) \right\} \quad (7)$$

The energy used to execute the offloaded agent's task through various n_j nodes is:

$$EC^{e}_{ijk} = \sum_{j\in\{N[t_k]\}} \left(a_j \frac{S_{ik}}{C_j} + b_j \frac{(I_{ik}+O_{ik})}{D_{j,next}} \right) \quad (8)$$

where *next* indicates the succeeding device on the route back to n_i to deliver the output.

The time used for executing the offloaded agent's task is:

$$ET^{e}_{ijk} = \sum_{j\in\{N[t_k]\}} \left(\frac{S_{ik}}{C_j} + \frac{(I_{ik}+O_{ik})}{D_{j.next}} + Q_{ik} \right) \quad (9)$$

where *next* indicates the succeeding device on the route back to n_i to deliver the output.

The total energy needed for the offloaded agent having task *k* from n_i through various n_j nodes is

$$EC^{o}_{ijk} = EC^{e}_{ijk} + EC^{d}_{ijk} . \quad (10)$$

The total energy needed for all the tasks of n_i: $EC(n_i) = E_{part} + EI + E_{merging} + \sum_{t_k\in\{T\}_l} \left(EC^{l}_{iik}\right) + \sum_{t_k\in\{T\}_o, n_j\in\{N\}_A} \left(EC^{o}_{ijk}\right)$ (11)

where $EI = p_{Idle} \times \sum_{t_k \in \{T\}_{o,n_j \in \{N\}_A}} ET^e_{ijk} - a_i \times \sum_{t_k \in \{T\}_l} (ET^e_{iik})$ (12);

with *EI* being the energy spent by node n_i when it is idle, that is the amount of energy spent on locally executed tasks subtracted to the amount of energy it saves on offloaded tasks. E_{part} is the energy used to partition the application into tasks according to its DAG. $E_{merging}$ is the energy used to merge the results of various tasks executions. These last two values can be approximated during the simulation.

The total time needed for all the tasks of n_i to complete their execution:

$$ET(n_i) = \sum_{t_k \in \{T\}_l} \left(ET^l_{iik}\right) + \sum_{t_k \in \{T\}_{seq}} \left(ET^e_{ijk}\right) + Max_{t_k \in \{T\}_{par}} \left(ET^e_{ijk}\right) \quad (13)$$

where $n_j \in \{N\}_A$

3.2 Mathematical model of the energy minimization problem

By taking into account the above formulations, our objective is to minimize the overall energy consumption of the task executions by all the devices n_i, which is formally described as follows:

$$min \sum_{i=1}^{n} \sum_{k=1}^{x_i} \lambda_{iik} EC^l_{iik} + (1 - \lambda_{iik}) \sum_{i=1, j \neq i}^{n} \sum_{k=1}^{m_i - x_i} \lambda_{ijk} EC^o_{ijk} \quad (14)$$

over x_i

subject to the constraints:

$$\lambda_{ijk} = 0, \ \forall e_{ij} \notin E, \ \forall k \in M \quad (15)$$

$$\sum_{j \in N} \lambda_{ijk} = \varepsilon_i, \ \forall i \in N, \ \forall k \in M \quad (16)$$

$$\sum_{j \in N} \lambda_{ijk} \leq 1, \ \forall j \in N, \ \forall k \in M \quad (17)$$

$$\lambda_{ijk} \in \{0,1\} \quad (18)$$

$$x_i \in [1,m] \quad (19)$$

$$x = \sum_{i=1}^{n} x_i \ , \ i \in N \quad (20)$$

Where,

$\boldsymbol{\varepsilon_i}$: binary indicator that is 1 if device *i* has a task to be executed and 0 otherwise

$\boldsymbol{\lambda_{ijk}}$: binary decision variable for task allocation, which is 1 if the task k of device *i* is offloaded to execute on device *j* and 0 otherwise (λ_{iik} local execution on device *i*).

m_i : Number of tasks generated by the DAG on Device *i*

Constraint (15) warrants that the task allocations are determined according to the feasible TMC connectivity.

Constraint (16) denotes that if a device has an offloadable task, this task should be assigned.

Constraint (17) denotes that during a task offloading round a device will execute at most one task at a time (which could either be its own task or an offloaded task from a nearby device).

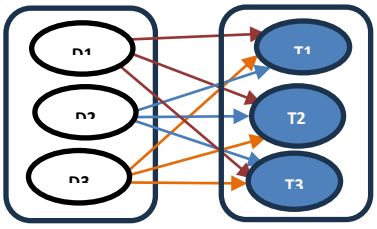
Constraint (18) follows from the context.

Constraint (19) denotes the tasks to be executed locally on each device *i*.

Constraint (20) denotes the number of locally executed tasks in the whole TMC system.

3.3 Adapted Kuhn-Munkres (Hungarian) Method for Optimal Task Assignment

We next propose the optimal solution for the problem in **section 3.2** via a task assignment policy. At first glance, one might regard this optimization problem as the classical NP-hard task allocation problem, meaning that finding and concluding the best answer in a finite period is practically impossible [30, 42]. It has been demonstrated in the literature, that this kind of task allocation problem can be solved optimally using the Kuhn-Munkres (Hungarian) method [5, 25, 35]. It is one of the methods used in assigning tasks to each worker device in the system that achieves an optimal assignment in which the aggregate cost is minimized, and whereby every task must be assigned to only one device. It considers a bipartite graph for the possible allocations (See Figure 5 (a)), derived from a 2-dimensional square matrix (See Figure 5 (b)) in which the rows D_i represent the devices and the columns T_i represent the tasks. An entry c_{ij} denotes the cost of assigning task j to device i. Its running complexity is $O(n^3)$, where n is the number of tasks, and equal to the number of worker devices.



(a)

C_{ij}	*T1*	*T2*	*T3*
D1	5	8	6
D2	7	9	7
D3	6	4	4

(b)

Figure 5: (a) A bipartite graph of all possible allocations, (b) A matrix of edge weights

However, when dealing with a TMC system, there often arise the need to manage more tasks than devices at a time, and then more devices than tasks at another time. It varies based on tasks that complete their execution, devices that run out of resources and/or, devices that move into or out of the transient cloud. In those situations, we may want to assign multiple tasks to the same device, or a device might not be assigned any task. Therefore, unlike the Classical Hungarian method, it is clearly important for the assignment algorithm to achieve certain properties such as load balancing and/or collocating tasks. Hence, we propose a modification to the Hungarian method to make it suitable for TMC systems.

3.3.1 Cost allocation in the graph for a given device

The Hungarian algorithm assumes the existence of a bipartite graph, ***G2 = {D, T, E}*** where *D* is the set of devices that the sender has access to through the network, *T* is the set of offloadable tasks of the sender, and *E* is the set of edges. The edge weights may be stored in a matrix. Given that each device runs the allocation scheme for the tasks it has, the devices it can access and on the basis of the data it has gathered in the TMC system, this matrix is built in the algorithm according to the following cases:

Case 1*:* To indicate that the device *i* can do a local allocation of the task, for an edge that connects device *i* to its own tasks, we set the weight $c_{ik} = EC^{l}_{iik}$. This means that for a device *i* running the allocation scheme, row *i* will have the energy consumption of local execution for each corresponding offloadable task of this device.

Case 2: For an edge that connects tasks of device *i* with another device *j*, we set the weight $c_{jk} = EC^{o}_{ijk}$. This means that for a device *i*, row *j* will have the energy consumption of offloaded execution from device *i* to device *j* for each corresponding offloadable task of the device *i*.

Case 3: If the number of tasks is less than the number of devices, some of them cannot be assigned any job, so that we have to introduce one or more dummy tasks of size zero to convert the unbalanced assignment problem to the balanced assignment problem that can now be solved classically. The devices assigned with the dummy tasks are assigned are then left out of the final result.

Case 4: If the number of tasks is more than the number of devices, some of them cannot be assigned any device. However, if we introduce one or more dummy devices to balance the problem, the assignment algorithm will cause the starvation of larger tasks. Therefore, we rather iteratively select an equal number of tasks to assign to the devices until all tasks are assigned. The selection may be based on the priority given by the queuing delay, the level of dependency in the DAG, the remaining time for the connection due to the mobility.

Case 5 (Load Balancing): If a task has already been assigned to a given device *j*, device *i* should have the ability to inflate the costs of other tasks by some factor (i.e., multiply the weight on already assigned devices by 2) when considering device *j* again for the assignment. This is in other to avoid overworking and congestion, long waiting time, and even distribution of tasks.

3.3.2 The modified algorithm

Here, we modify the Hungarian Algorithm on a bipartite graph as proposed by [44, 45] for the reasons already presented.

Adapted Hungarian Algorithm (AH_Algo)

Given that the variables u_i are assigned to each node d_i and dual variables v_j are assigned to each node t_j, the minimization of the assignment problem is feasible when $u_i + v_j \leq c_{ij}$.

Input: A bipartite graph G2 = {D, T, E} (where |D| = n, |T| = m) and an n × m matrix of edge costs C

Output: An optimum matching graph M

Let n = number of devices (nodes) and m the number of agents

If m >= n then do

r ← n

While (r>=0 and m>=n) do

Select n tasks according to Case 4

*Run the **AssignTask()** algorithm for the n tasks and n nodes*

Remove from A the agents that have been assigned

Run Case 5 for Load Balancing

m ← m – n, r ← m%n

Endwhile

*Run the **AssignTask()** algorithm for the r tasks and n nodes according to Case 3*

*Else Run the **AssignTask()** algorithm for the m tasks and n nodes according to Case 3*

***AssignTask()**:*

Let q = min (n,m)

1. Perform initialization of the matching:

(a) Begin with an empty matching, $M_0 = \emptyset$.

(b) Assign feasible values to the dual variables αi and v_j as follows:

$\forall d_i \in N, u_i = 0, \ \forall t_i \in T, v_j = min_i(c_{ij})$

2. Perform q stages of the section in ***Phase****.*

3. Output the matching after the q^{th} stage: $M = M_q$.

Phase*:*

An alternating path: a path through the graph such that each matched edge is followed by an unmatched edge and vice-versa

An augmenting path: an alternating path that begins and ends with an exposed node.

A Hungarian tree: All alternating paths originating from a given unmatched node.

Searching for an augmenting path in a graph involves exploring these alternating paths in a breadth-first manner, and the process can be called growing a Hungarian tree.

An Equity Graph is made up of edges where $u_i + v_j = c_{ij}$.

1. Designate each free (unmatched) node in N as the root of a Hungarian tree.

2. Grow the Hungarian trees rooted at the free nodes in the equality subgraph. Designate the indices i of nodes di encountered in the Hungarian tree by the set $I*$, and the indices j of nodes tj encountered in the Hungarian tree by the set $J*$. Go to step(4) if an augmenting path is found. Otherwise, if the Hungarian trees cannot be grown further, continue to step(3).

3. Modify the dual variables u and v as follows to add new edges to the equality subgraph. Then resume step(2) to continue the search for an augmenting path with

$\theta = 1/2 \ min \ (c_{ij} - u_i - v_j), i \in I*, j \notin J*$

$u_i \leftarrow u_i + \theta$ *if* $i \in I*$ *and* $u_i \leftarrow u_i - \theta$ *if* $i \notin I*$

$v_j \leftarrow v_j - \theta$ *if* $j \in J*$ *and* $v_j \leftarrow v_j + \theta$ *if* $j \notin J*$

4. Augment the current matching by exchanging matched edges with unmatched edges along the selected augmenting path. That is, M_k (the new matching at stage k) is given by $(M_{k-1} - P) \cup (P - M_{k-1})$, where M_{k-1} is the matching from the previous stage and P is the set of edges on the selected augmenting path.

3.3.3 Dynamic Reallocation of tasks

Given the plausible unpredictability of user mobility and the variability of devices' resources states, optimal allocation decisions must be made dynamically at runtime in order to adapt to new conditions and so on. Mobility alone will cause the quick change of some costs c_{ij} in the matrix for the mobile devices (workers) that move. Therefore, dynamic reallocation seems more appropriate, but it has an associated higher communication overhead, which should be taken under control [46, 47]. In the *Adapted Hungarian Algorithm* used here, the costs on a given column or row is updated in the matching matrix. In this case, if a change happens on a device, there is no need to restart the algorithm from start, since the current matrix represents the optimal assignment before the change of costs in one node. It is then more effective to use the Dynamic Hungarian algorithm proposed by [48]. It uses the current optimal assignment matrix to produce a new complete matching matrix M representing an optimal solution to the problem with the changed edge costs.

Optimized Version of the algorithm to integrate dynamic reallocation (Dynamic_Hungarian)

Input:

• An optimal solution to the above assignment problem, comprising a complete matching $M*$, and the final values of all dual variables ui and vj .

• k cost changes, each of which can be a row $i*$, a column $j*$, or a single entry $ci*j*$ of the cost matrix.

Output:

A new complete matching, M, representing an optimal solution to the problem with the changed edge costs.

Perform initialization: For each of the k cost changes

• If a single value $ci*j*$ of the cost matrix changed from cold to cnew:

(a) If cnew > cold and $di*$ is matched to $tj*$, then remove the edge $(di, tj*)$ from the matching $M*$.*

(b) Otherwise, if cnew < cold and $uj + vi* > cnew$*

– Assign $uj = minj (ci*j − vi)$*

– If $di*$ is not matched to $tj*$, remove the edge $(di, mate(di*))$ from the matching $M*$.*

[Note: We may stochastically decide to modify $di*$ rather than $tj*$ in this case. If $tj*$ is modified, then the edge $(mate(tj*), tj*)$ should be removed from the matching instead of $(di, mate(di*))$] .*

• Otherwise, if a row $i*$ of the cost matrix changed:

(a) If $di*$ is matched, remove the edge ($di*$, mate($di*$)) from the matching $M*$.

(b) Assign $ui* = minj\ (ci*j - vj)$

• Otherwise, if a column $j*$ of the cost matrix changed:

(a) If $tj*$ is matched, remove the edge (mate($tj*$), $tj*$) from the matching $M*$.

(b) Assign $vj = mini(cij* - ui)$*

2. Let $k*$ be the number of edges removed from the matching in the initialization phase of the algorithm (note $k* \leq k$). Perform $k*$ iterations of Phase from the AH_Algo.

Output the resulting matching M.

3.3.4 Evaluating the Send_Over benchmark for a given agent hosting task T_k originating from a node n_i

With the continual movement of the mobile users and the progressive change of status of the resources in the devices over time, the costs c_{ij} may change often, and this even faster if the connected devices move in divergent directions or out of reach. For this reason, there is need to check at every time slice T_s, the devices that have received offloaded agents and to measure the remaining connection time T_c on the links that they have in the network. We have that: $\boldsymbol{T_c = (D-D_{ms})/S_m}$, $\boldsymbol{D_{ms} = C * T_{RTT}}$, $\boldsymbol{C = 300 * 10^6\ m/s}$, with $\boldsymbol{D}$: Max Distance Coverage Range of the Wifi = [80 – 200]m (in practice), $\mathbf{D_{ms}}$: Average sampling distance between 2 given devices, $\boldsymbol{T_{RTT}}$: Round Trip Time interval, $\boldsymbol{C}$: Standard speed of the wave, $\boldsymbol{S_m}$: Current Speed of the device.

An agent executes its tasks on a device as long as that device has sufficient resources and the connection time is valuable. When the conditions of execution are no longer satisfactory, the agent needs to move to another device on the basis of the allocation algorithm. Here, we consider the energy level of the device measured at every time slice T_s and the connection time, in order to determine the Send_Over benchmark (which is the threshold for the agent to move to another device). The lowest limit energy level for any device to be considered for participation in the network may be 40%. As indicated in Table 2, we consider 2 metrics (Contact Time, Energy used) measured for every time period T_s to decide on the Send_Over of the agent concerned (see line 1 to line 4). For this purpose, we assume some practical values including $T_s = ET(n_i) \times \frac{1}{\partial}, with\ \partial = ln(b_i) + ln(T_c)$.

Table 2: Metrics used for the Send_Over

Energy Consumed	**Connection Time**		**Send_Over Benchmark**
1) Low *I* Average	High		Low
2) Low *I* Average			
3) Low *I* Average *I* High	Average		Average
4) High			
	Low		High
	Low *I* Average *I* High		High
Low = [0,25[	Low = [0, T_s + 5[		
Average = [25,50[	Average = [T_s + 5, T_{rt} + 10[		
High = [50, ---[	High = [T_{rt} + 10, ---[		
E_Level	**Energy Level in the Battery (%)**		
Threshold	Low	Average	High
40	41- 60	61-80	81- 100

To design the Send_Over algorithm, we consider the following values:

T_{rt} : Remaining Time to avail the Service

E_{level} : Energy Level in the Battery

T_s : *Time slice after which connection time and energy used is measured*

SS: Average received signal strength = **$AVG_{i \varepsilon Ts}$ RSSI[i]**

RSSI[1,…, i] : Array of signal strength values collected overtime interval T_s.

Essentially, the *Send_Over* is triggered when its benchmark is high. However, If *Send_Over* benchmark = *Average* and *E_Level* = *low* and $SS < SS_{Threshold}$ and $T_{rt} > T_c$, then *Send_Over* is equally triggered.

Send_Over Algorithm

Input:

Agent id carrying Task k

Start:

T_{rt} = Initial total Execution Time

While $T_{rt} > 0$

*For i ←1, **T_s** Measure RSSI[i]*

Compute Energy level, SS, Contact Time

Get Send_Over_Benchmark

*If (Send_Over = High) and (SS < **$SS_{Threshold}$**) then AH_Algo()*

*ElseIf (Send_Over = Average) and ((**E_Level = Low**) or (SS < **$SS_{Threshold}$**)) and ($T_{rt} > T_c$) then AH_Algo()*

Else do nothing

EndIF

EndWhile

4. Simulation and Results

4.1 General Considerations

To evaluate the effectiveness of the solution proposed, we implement a simulation environment suitable for this framework's setting with the help of mobile agents. We used the **Repast Symphony 2.9.1** software which is an agent-based modelling and simulation system. It is open source, mainly java-based, richly interactive, expert focused. The operating environment is an Intel Core i5 with a 4.6 GHz CPU and 8 GB memory laptop. We initially considered 20 devices (nodes) constituted into a TMC system as described above, and they are assumed to be moving in and out of a 200 m long surface area at different positions. Given the simulation environment, we can assign to all nodes the same battery level and enough energy to run the whole experiment as these deplete over time. We considered a threshold for the amount of energy that needs to be available on a mobile device before it can receive an offloaded task, so that the device can still run its own compulsory activities (the minimal level of energy needed for a device to participate in the system). We applied the Random Gauss-Markov (RGM) [3] model which is often used in modelling mobility for mobile cloud augmentation. RGM moves a mobile node in time intervals, such that, at each time interval, the next location d_{next} and speed s_{next} are calculated based on its current location d_{pre} and speed s_{pre} . We have that:

$$\boldsymbol{s_{next} = \alpha s_{pre} + (1-\alpha) s_{avg} + \sqrt[2]{(1-\alpha^2) s_{ran}}} \quad \text{and} \quad \boldsymbol{d_{next} = \alpha d_{pre} + (1-\alpha) d_{avg} + \sqrt[2]{(1-\alpha^2) d_{ran}}}$$

where α is the tuning parameter to vary the randomness. s_{avg}, d_{avg} represent the mean values, and s_{ran}, d_{ran} are two random variables from a Gaussian distribution. RGM avoids the sudden change of direction issue by letting past states influence future states.

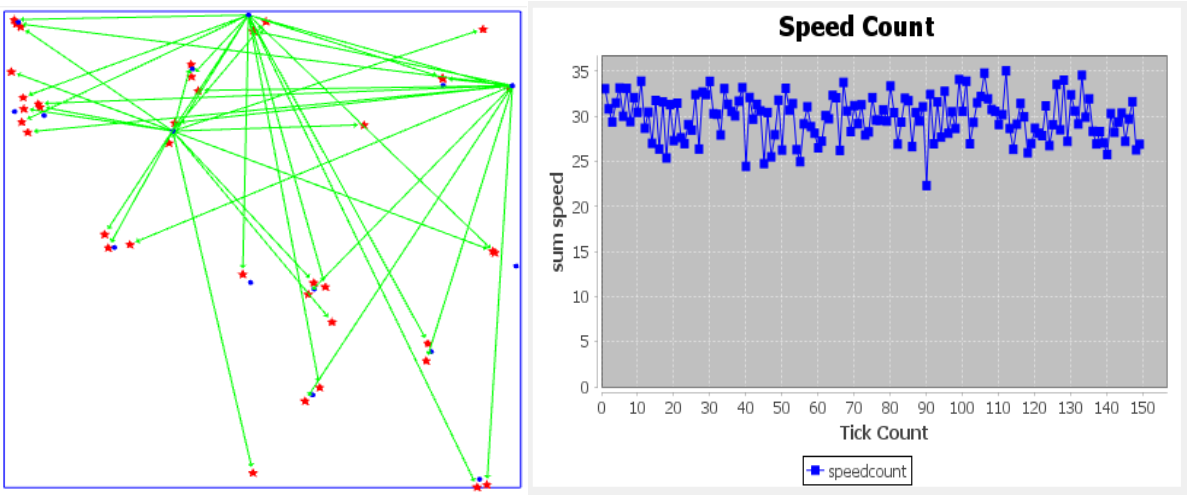
Every node generates its own DAG of maximum 20 tasks independently following a Poisson arrival process. Task characteristics (i.e., complexity, data size, etc.) are selected randomly within the range of chosen max values. For example, we may vary the task sizes from 0 MFLOP to 60 MFLOP and the input data sizes from 0 to 20MB. Other important simulation parameters used for the devices are given in the Table 3.

Table 3: Simulation Parameters

Parameters	**Max Values**
Transmission power	100 MW
Reception power	25 MW
CPU processing capacity	1.5 GHz
Data transmission rate	20 Mbps
Bandwidth	20 MHz

Using this mobile agent simulator, we constructed a prototype, while considering the characteristic values of the network protocol (BATMAN), to represent all the components of the 3 planes designated in Figure 5. This prototype was implemented with over 7000 lines of code in Java.

Figure 5 (a) shows a view of the mobile devices (blue dots), their agents moving towards the devices where they have been assigned (red stars) and the green arrows shows the link between a device and the agents carrying its tasks. Figure 5 (b) illustrates the variability of the speeds of the devices as they move. Every device runs the *AH_Algo* as it starts the allocation of its tasks. Once this is done, during its movement, it will run the *Dynamic_Hungarian* algorithm in order to manage the few changes in cost. If a device is getting out of reach, it will run the *Send_over* algorithm. To handle cases of packet loss, after a required execution and transfer time, supplemented by an average delay rate, if a given agent has not returned to the sender with a result, the MAZ may resend an agent with the same information for re-execution especially if the priority of the tasks hosted in it was critical.



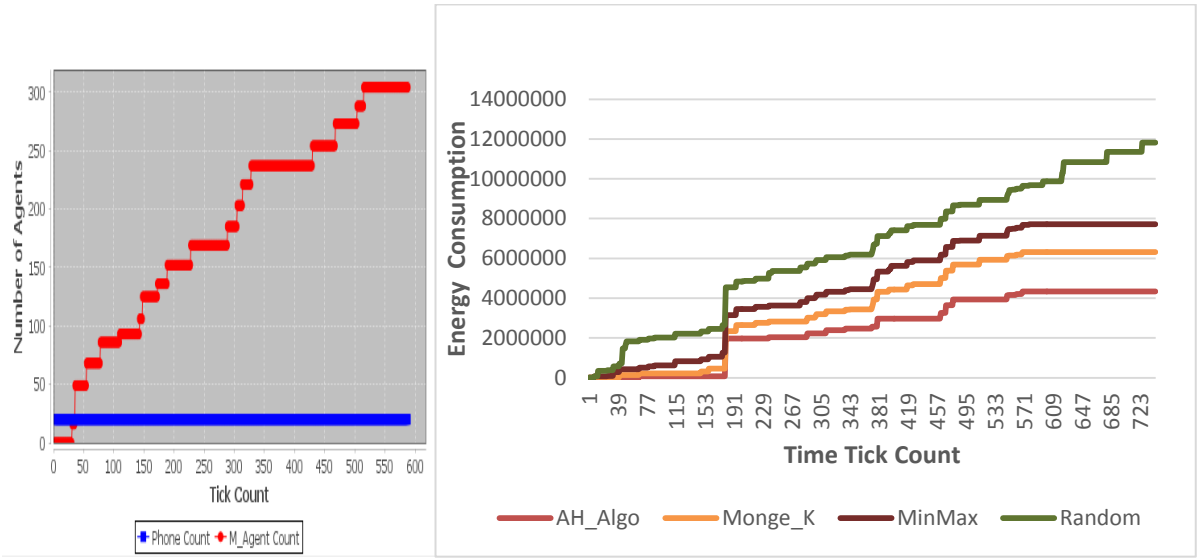
(a) **(b)**

Figure 5: (a) Simulation view of the devices and allocated agents (b) Variability of the devices' speeds over time

4.2 Performance Evaluation

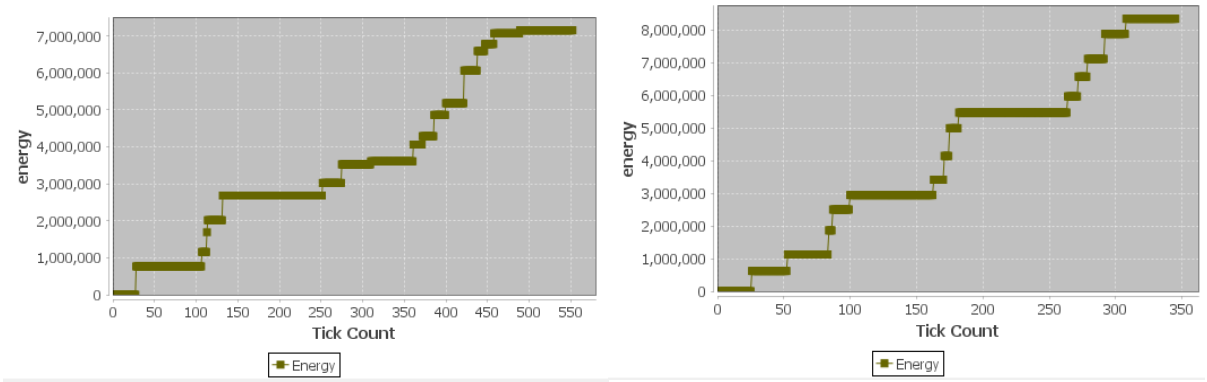
In the simulation with 20 devices, about 300 tasks are generated and encapsulated in agents (see Figure 6 (a)). We compare our updated Kuhn Munkres task assignment policy to three other schemes that are also used in these scenarios: the Monge Kantarovich Algorithm, the MinMax algorithm and the Random Allocation algorithm. We run 50 rounds of these task assignment schemes to obtain the average energy consumed up until the execution of all the tasks. The feasible network connectivity among the devices also varies from round to round, which depends on the devices' positions while the users move. We depict the energy consumed of different schemes in Figure 6 (b). We observe that our proposed scheme (AH_Algo) can save sufficiently more energy compared to other schemes. Also, with the AH_Algo, the tasks' execution is completed on average a few time ticks earlier than with the other schemes, considering the constraints depicted in the TMC system, as it stops increasing at time tick 569. Furthermore, Figure 8 presents the average load balancing over the 20 devices, each having a number of tasks allocated through a given scheme as listed in the table. We note that the AH_Algo is able to equitably manage the load of tasks assigned to devices better than other schemes. These points demonstrate the superior performance

of our task allocation scheme for a TMC system.



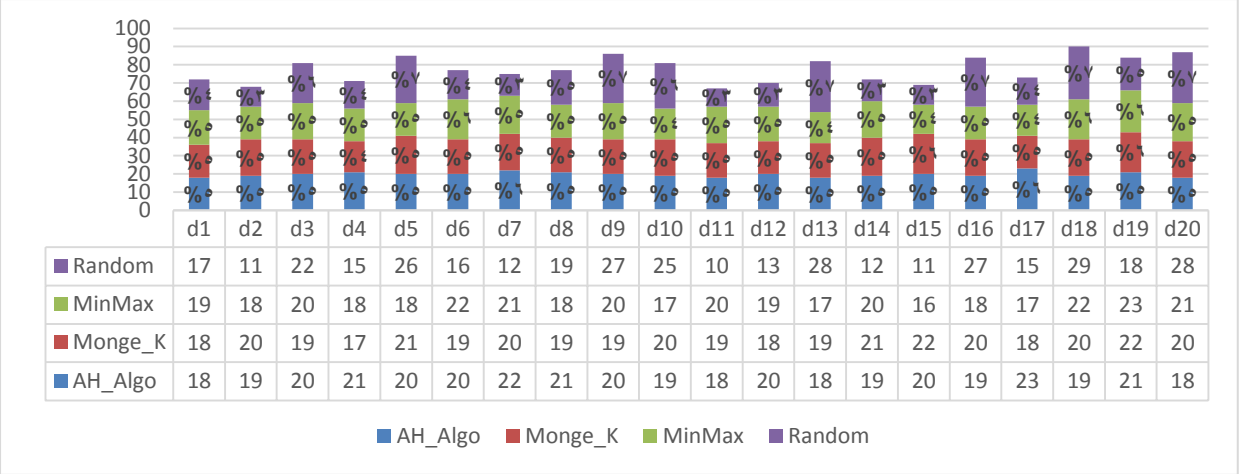
(a) **(b)**

Figure 6: (a) Devices and Mobile Agents count (b) Average energy consumption in the whole system over time per allocation scheme



(a) **(b)**

Figure 7: (a) Energy consumed with 15 devices (b) Energy consumed with 20 devices



	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
Random	17	11	22	15	26	16	12	19	27	25	10	13	28	12	11	27	15	29	18	28
MinMax	19	18	20	18	18	22	21	18	20	17	20	19	17	20	16	18	17	22	23	21
Monge_K	18	20	19	17	21	19	20	19	19	20	19	18	19	21	22	20	18	20	22	20
AH_Algo	18	19	20	21	20	20	22	21	20	19	18	20	18	19	20	19	23	19	21	18

Figure 8: Load Balancing in Task Allocation on the devices

We equally examined the case of running the AH_Algo when there are more or less devices in the TMC system to manage tasks. Figure 7 (a) represents the average energy consumed over time with 15 devices, and Figure 7 (b) does same with 20 devices. We observe that the performance of our algorithm will slightly increase with increasing number of users even for devices that has a high number of tasks. This can be seen in the fact that it takes more time to execute those tasks with less devices though almost the same amount of energy is spent in both cases. The reason is that a larger number of devices would enable a device to have more neighbors, and hence a device can have more opportunities to offload the task to a suitable device and to execute more tasks parallelly. This will decrease the number of idle devices, and hence promote the overall performance. However, the difficulty is that when the TMC system expands, it has the tendency to increase the processing time because there are more devices to consider for the task assignment.

5 Conclusion and Limitations

In this paper, we modeled the tasks allocation problem in TMC system as an energy consumption optimization problem, while taking into account task dependency, data transmission and some constraint conditions such as and cost. We further solved it using an adapted version of the Kuhn-Munkres (Hungarian) Algorithm to fit in the mobile ad hoc cloud networking infrastructure described above. Furthermore, we have considered the Dynamic_Allocation algorithm and proposed a Send_Over algorithm that permit to easily manage the change in energy cost due to the constant movement of devices. A series of simulation experiments are conducted to evaluate the performance of the algorithm and the results obtained are efficient and acceptable compared to what obtains in other prominent tasks allocation algorithms.

6 Future Works

For the future work, we will test the performance of algorithms with much larger task graphs and devise more efficient heuristic algorithms to solve this task scheduling problem. We consider using a lightweight swarm intelligence algorithm permitting the agents to learn from the environment and act autonomously. This may equally help to manage many more devices in the TMC system. Also, we consider carrying out a priority-based

scheduling for the tasks in a DAG, test them with existing real life mobility trace. This will offer a better representation of the devices.

References

[1] W. Xu, "Huawei predicts 100 Billion internet connections globally by 2025" IndoAsian News Service, Aug. 19, 2022.

[2] S. Nath and J. Wu, "Dynamic Computation Offloading and Resource Allocation for Multi-user Mobile Edge Computing" in Proc. IEEE Global Commun. Conf. (GLOBECOM), 2020, pp. 1-6.

[3] B. Zhou and R. Buyya, "Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions", ACM Comput. Surv., vol. 51, no. 1, Art. 13, Jan. 2018, 38 pages.

[4] A. Sciarrone, I. Bisio, F. Lavagetto, T. Penner, and M. Guirguis, "Context awareness over transient clouds" in Proc. IEEE Global Commun. Conf. (GLOBECOM), 2015, pp. 1-5.

[5] M. Guirguis et al., "Assignment and collaborative execution of tasks on transient clouds", Ann. Telecommun, vol. 72, no. 3-4, pp. 251-261, Jul. 2017.

[6] B. Hu, X. Yang, and M. Zhao, "Online energy-efficient scheduling of DAG tasks on heterogeneous embedded platforms", J. Syst. Archit , vol. 140, p. 102894, 2023.

[7] M.W. Tian, S.R. Yan, W. Guo, A. Mohammadzadeh, and E. Ghaderpour, "A New Task Scheduling Approach for Energy Conservation in Internet of Things", Energies , vol. 16, no. 5, p. 2394, 2023.

[8] S. Kadry, K. Abdulkareem, A. Lakhan, M. Mohammed, and A. Rashid, "Deadline Aware and Energy-Efficient Scheduling Algorithm for Fine-Grained Tasks in Mobile Edge Computing", Int. J. Web Grid Serv, vol. 18, no. 1, pp. 1-18, 2022.

[9] M. Mscs, "An efficient dynamic decision-based task optimization and scheduling approach for microservice-based cost management in mobile cloud computing applications", Pervasive Mobile Computing, vol. 92, 2023.

[10] A. A. Amer, I. E. Talkhan, R. Ahmed, and others, "An Optimized Collaborative Scheduling Algorithm for Prioritized Tasks with Shared Resources in Mobile-Edge and Cloud Computing Systems", Mobile Netw. Appl , vol. 27, pp. 1444–1460, 2022, doi: 10.1007/s11036-022-01974-y.

[11] C. Jin, J. Xu, Y. Han, J. Hu, Y. Chen, and J. Huang, "Efficient Delay-Aware Task Scheduling for IoT Devices in Mobile Cloud Computing", Mobile Inf. Syst , vol. 2022, Art. 1849877, 10 pages, 2022.

[12] R. Alakbarov, "An Optimization Model for Task Scheduling in Mobile Cloud Computing", IJCAC, vol. 12, no. 1, pp. 1-17, 2022.

[13] J. Guo, Y. Liu, B. Yang, B. Xiao, and Z. Li, "Energy-Efficient Dynamic Computation Offloading and Cooperative Task Scheduling in Mobile Cloud Computing", IEEE Trans. Mobile Comput , vol. 18, no. 2, pp. 319-333, Feb. 2019.

[14] Z. A. Jaaz, S. A. Abdulrahman, and H. M. Mushgil, "A dynamic task scheduling model for mobile cloud computing", in Proc. 9th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI), Jakarta, Indonesia, 2022, pp. 96-100.

[15] P. Singh, P. Singh, S. Rajpoot, and D. P. Singh, "Study and Analysis of Offloading in Mobile Cloud Computing", in Proc. Int. Conf. Technol. Advancements Innovations (ICTAI), Tashkent, Uzbekistan, 2021, pp. 280-284.

[16] X. Liu, J. Liu, and H. Wu, "Energy-Efficient Task Allocation of Heterogeneous Resources in Mobile Edge Computing", IEEE Access, vol. 9, pp. 119700-119711, 2021.

[17] E. Soares, P. Brandão, R. Prior, and A. Aguiar, "Experimentation with MANETs of Smartphones", arXiv:1702.04249v1 [cs.NI], Feb. 2017.

[18] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey", Wireless Commun. Mobile Comput , vol. 17, pp. 1607-1625, 2017.

[19] S. C. Shah, "A Mobile Ad hoc Cloud Computing and Networking Infrastructure for Automated Video Surveillance System", J. Comput. Sci. Tech. Rep , vol. 6, 2017.

[20] M. Guirguis et al., "Assignment and collaborative execution of tasks on transient clouds", Ann. Télécommun, vol. 73, no. 3-4, pp. 251-261, 2018.

[21] I. Bisio, F. Lavagetto, A. Sciarrone, T. Penner, and M. Guirguis, "Context-awareness over transient cloud in D2D networks: energy performance analysis and evaluation", Trans. Emerg. Telecommun. Technol , vol. 28, no. 2, 2017.

[22] A. Sciarrone, I. Bisio, F. Lavagetto, T. Penner, and M. Guirguis, "Context Awareness over Transient Clouds", in Proc. IEEE Global Commun. Conf. (GLOBECOM) , 2015, pp. 1-5.

[23] T. Penner et al., "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," in Proc. IEEE Global Commun. Conf. (GLOBECOM), 2014, pp. 2801-2806.

[24] T. Penner et al., "Demo: Transient clouds", in Proc. Int. Conf. Mobile Comput. Appl. Serv. (MobiCASE), 2014, pp. 153-154.

[25] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting Massive D2D Collaboration for Energy-Efficient Mobile Edge Computing", in Sustainable Green Networking and Computing in 5G Systems: Technol., Economics, Deployment, 2017.

[26] T. F. Ndambomve, F. Mokom, and K. D. Taiwe, "A Dynamic Application Partitioning and Offloading Framework to Enhance the Capabilities of Transient Clouds Using Mobile Agents", Int. J. Comput , vol. 40, no. 1, pp. 109-126, 2021.

[27] M. Shahzamal, "Lightweight Mobile Ad-Hoc Routing Protocols for Smartphones", Macquarie University, Sydney, Australia, Apr. 2018.

[28] IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks—Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-2012, pp. 1–2793, Mar. 2012.

[29] F. Gu, J. Niu, Z. Qi, and M. Atiquzzaman, "Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions", J. Netw. Comput. Appl , vol. 85, pp. 1-23, 2018.

[30] A. Lakhan et al., "Dynamic Application Partitioning and Task-Scheduling Secure Schemes for Biosensor Healthcare Workload in Mobile Edge Cloud", Electronics , vol. 10, p. 2797, 2021.

[31] Q.H. Nguyen and F. K. Hussain, "Smart Mobile Edge Computing for Wireless Sensor Networks: Energy Efficient Task Offloading Strategies", in Proc. IEEE Global Commun. Conf. (GLOBECOM) , 2022, pp. 281-285.

[32] Z. Wei, X. Yu, and L. Zou, "Multi-Resource Computing Offload Strategy for Energy Consumption Optimization in Mobile Edge Computing," Processes , vol. 10, no. 1762, 2022, doi: 10.3390/pr10091762.

[33] R. Tahboub and F. Warasna, "Security issues in Mobile Cloud Computing Frameworks based on Mobile Agents," Deanship of Graduate Studies and Scientific Research, Palestine Polytechnic University, Hebron, Palestine, 2015.

[34] X. Liu, C. Yuan, Z. Yang, and Z. Zhang, "Mobile-agent-based energy-efficient scheduling with dynamic channel acquisition in mobile cloud computing," J. Syst. Eng. Electron. , vol. 27, no. 3, pp. 712–720, Jun. 2016.

[35] Q. Wang, Y. Mao, Y. Wang, and L. Wang, "Computing task offloading based on multi-cloudlet collaboration," Comput. Appl. , vol. 40, pp. 328–334, 2020.

[36] S. Ghasemi-Falavarjani, M. Nematbakhsh, and B. S. Ghahfarokhi, "Context-aware multi-objective resource allocation in mobile cloud," Comput. Electr. Eng. , vol. 44, pp. 218-240, 2015, doi: 10.1016/j.compeleceng.2015.02.006.

[37] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds", ACM Trans. Internet Technol. , vol. 18, no. 2, Art. 23, 25 pages, Jan. 2018, doi: 10.1145/3122981.

[38] V. Balasubramanian, K. Kroep, K. C. Joshi, and R. V. Prasad, "Reinforcing Edge Computing with Multipath TCP Enabled Mobile Device Clouds," 2019.

[39] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting Massive D2D Collaboration for Energy-Efficient Mobile Edge Computing," IEEE Wireless Commun. , Aug. 2017, doi: 10.1109/MWC.2017.1600321.

[40] S. C. Shah, "A Mobile Ad hoc Cloud Computing and Networking Infrastructure for Automated Video Surveillance System," J. Comput. Sci. Tech. Rep. , vol. 6, 2018.

[41] C. Tang, S. Xiao, X. Wei, M. Hao, and W. Chen, "Energy-efficient and Deadline-satisfied Task Scheduling in Mobile Cloud Computing," in Proc. IEEE Int. Conf. Big Data Smart Comput. , 2018.

[42] S. M. Kak, P. Agarwal, and M. A. Alam, "Task Scheduling Techniques for Energy Efficiency in the Cloud", EAI Endorsed Trans. Energy Web , vol. 9, no. 39, Jun. 2022, doi: 10.4108/ew.v9i39.1509.

[43] H. Wu, "Multi-Objective Decision-Making for Mobile Cloud Offloading: A Survey" IEEE Access , Feb. 28, 2018.

[44] H. Cui, J. Zhang, C. Cui, and Q. Chen, "Solving large-scale assignment problems by Kuhn-Munkres algorithm", in Proc. 2nd Int. Conf. Adv. Mech. Eng. Ind. Inform. (AMEII), 2016.

[45] H. W. Kuhn, "The Hungarian method for the assignment problem", Naval Res. Logistics Quart. , vol. 2, pp. 83–97, 1955.

[46] A. Ali, M. M. Iqbal, H. Jamil, F. Qayyum, S. Jabbar, O. Cheikhrouhou, M. Baz, and F. Jamil, "An Efficient Dynamic-Decision Based Task Scheduler for Task Offloading Optimization and Energy Management in Mobile Cloud Computing," Sensors , vol. 21, no. 13, p. 4527, 2021.

[47] S. Nath and J. Wu, "Dynamic Computation Offloading and Resource Allocation for Multi-user Mobile Edge Computing" in Proc. IEEE Global Commun. Conf. (GLOBECOM) , 2020.

[48] G. A. Mills-Tettey, A. Stentz, and M. B. Dias, "The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs", Carnegie Mellon University, Jul. 2007.