IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 3, 2019, accepted July 8, 2019, date of publication July 18, 2019, date of current version August 13, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2929683*

# Data and Task Offloading in Collaborative Mobile Fog-Based Networks

**RADU-IOAN CIOBANU[1], CIPRIAN DOBRE[1], (Member, IEEE), MIHAELA BĂLĂNESCU[2], AND GEORGE SUCIU, JR.[2], (Member, IEEE)**

[1]Faculty of Automatic Control and Computers, University Politehnica of Bucharest, RO-060042 Bucharest, Romania
[2]Beia Consult International, RO-041386 Bucharest, Romania

Corresponding author: Radu-Ioan Ciobanu (radu.ciobanu@cs.pub.ro)

This work was supported in part by the NETIO project Tel-MonAer under Grant 53/05.09.2016, cod SMIS2014+ 105976, and in part by the University Politehnica of Bucharest, through the PubArt "Program for Supporting the Publishing of Articles and Scientific Communications" Program.

**ABSTRACT** The rapid growth of mobile devices in recent years has led to the rapid progress of mobile computing. However, this has exposed certain limitations that have, first, been addressed by mobile cloud computing. Once the Internet-of-Things devices have started being put online, a new step in the evolution of mobile networks was taken through the addition of edge and fog computing, where small nodes at the edge of the network take up some of the load on the cloud backend. Nevertheless, even this model has shown some limitations, which is why in this paper, we address the problem of off-loading data and computations from a mobile device to the cloud to fog nodes, or to other mobile devices in the vicinity. The novelty of our proposal is the addition of a layer composed exclusively of mobile devices that collaborate in an opportunistic fashion, as a first resort when needing some computations to be off-loaded. Through a thorough analysis using the MobEmu mobile network simulator, we show that our solution is able to reduce total computation time by as much as 19%, decrease the cloud usage with up to 40%, and reduce battery consumption with more than 6%.

**INDEX TERMS** 4G mobile communication, clouds, collaborative work, edge computing, mobile nodes, peer-to-peer computing, social computing, wireless communication.

## I. INTRODUCTION

Mobile devices are becoming our everyday companions, and, whether they are smartphones or wearables, they represent an essential part of our life. This has led to the rapid progress of mobile computing, which has become a very important aspect in the development of IT, as well as commerce and other domains. However, despite the increasing usage of mobile computing, exploiting its full potential is difficult due to its inherent problems such as resource scarcity, frequent disconnections, and mobility. The appearance of cloud computing has represented a huge opportunity for the development of mobile services.

With the development of the cloud computing concept, mobile cloud computing (MCC) has been introduced as a potential technology for mobile services [1]. It can address these problems by executing mobile applications on resource providers external to the mobile device. More commonly, the term mobile cloud computing assumes running an application on a remote resource-rich server, while the mobile device acts like a thin client connecting to the server through Wi-Fi or 3G/4G. Some examples of this type are Facebook's location-aware services, Twitter for mobile, and mobile weather widgets.

Due to the fact that the consumer and enterprise market for mobile applications is set to reach $188.9 billion by 2020,[1] there is a lot of interest in designing efficient methods of managing mobile networks. Thus, in time, other methods have been developed starting from the mobile cloud computing concept in order to improve the usage and performance of the mobile network and all its devices by increasing speed, while at the same time reducing latency and costs.

Mobile edge computing (MEC) proposes moving resources (both computing and storage) at the base stations of cellular networks [2]. The purpose is to lower the mobile core usage and to reduce latency, which have increased due to the high amount of data traffic required by many

The associate editor coordinating the review of this manuscript and approving it for publication was Liqun Fu.

[1]https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/.

mobile applications that rely on data and services hosted remotely. To match these increasing demands, operators need to enhance and upgrade the capacities of existing network resources continuously. Moreover, they are forced to integrate new technologies (such as LTE Advanced) into their infrastructure in order to provide a good quality of experience for users, since they can provide higher bandwidth capacities and lower latency. However, improving existing resources and integrating new technologies may add significant operational costs, which is why mobile edge computing is considered a technology that is suitable for addressing this issue in certain scenarios. It aims to reduce network load by shifting computations from the cloud to the edge of the network.

Fog computing extends cloud computing by providing virtualized resources and engaged location-based services to the edge of the mobile networks so as to better serve mobile traffic [3]. The major difference between cloud computing and fog computing is the support of location awareness. The cloud ''resides'' in a centralized place and serves as a centralized global portal of information, whereas fog computing extends the cloud to reside at the user's premises and is dedicated to localized service applications. The idea of fog computing is to place lightweight cloud-like facilities in the proximity of mobile users. The fog can therefore serve mobile users with a direct short-fat connection as compared to the long-thin mobile cloud connection. The main advantages of fog computing are enhancing service quality to mobile users and improving the network efficiency (by avoiding the back-and-forth traffic between cloud and mobile users). However, while bringing certain advantages such as low latency, energy saving and context awareness [4], several challenges in fog systems still need to be addressed, including creating human-driven distributed systems, ensuring security and privacy, as well as scalability [5].

Mobile crowd sensing (MCS) presents a new sensing paradigm based on the power of mobile devices [6]. The sheer number of user-companioned devices (including mobile phones, wearable devices, and smart vehicles) and their inherent mobility enable a new and fast-growing sensing paradigm: the ability to acquire local knowledge through sensor-enhanced mobile devices (location, noise level, traffic conditions, etc.) and the possibility to share this knowledge with devices in proximity. The information collected on the ground and with the support of the cloud where data processing takes place make MCS a versatile platform that can often replace static sensing infrastructures. A formal definition of MCS specifies that it is a new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices, which is then aggregated and fused in the cloud for crowd intelligence extraction and people-centric service delivery.

All these components combine together into the paradigm of Drop Computing [7], which assumes that mobile nodes can offload data and computations to the cloud, to fog devices, and also to other neighboring nodes through close-range protocols. Such a network requires a set of new and improved offloading solutions, that are able to adapt to conditions and select one or more of the available offloading methods. Thus, our contributions in this paper are as follows. We first present the Drop Computing paradigm and place it in the context of cloud, fog and mobile computing. We then present novel data and computation offloading solutions for Drop Computing-based networks, that are able to increase user QoE (quality of experience), reduce costs for application and service developers, and decrease the rate of battery consumption. Through thorough and valid experimentation on synthetic and real-life scenarios, we are able to show that our proposed mechanisms are indeed able to improve upon the aforementioned metrics. Furthermore, we discuss the potential of Drop Computing and its compatibility with new and future communication technologies, and present some real-life use cases.

The rest of this article is structured as follows. In Section II, we discuss related work in the area of mobile offloading, and then we present solutions similar to our proposal, highlighting their drawbacks and showing how our solution attempts to address them. Then, Section III contains details about our proposed techniques for offloading in various scenarios, while Section IV offers a thorough analysis of the proposed mechanisms. Finally, in Section V we draw some conclusions and present future work.

## II. RELATED WORK

The cellular industry has experienced a large growth in the past few years, especially in terms of data traffic, which, already surpassing voice traffic, is continuously increasing by an order of magnitude every year [8]. This is creating challenges for the existing cellular networks, which brings the need of mobile data offloading. This refers to the delivery of data meant initially for mobile cellular networks onto other interfaces, through the use of complementary network technologies, with the purpose of alleviating congestion and making better use of available network resources. The goal is to maintain a certain level of quality of service for users, while reducing the cost and impact of medium and large-scale services on the mobile network. It is predicted that mobile offloading will become a top industry segment very soon, as traffic on mobile networks continues to rapidly grow.

The main reason for advancing mobile offloading is the increase in data traffic on cellular networks, which is causing congestion and affecting user experience. This increase can be attributed to several factors [9]. Firstly, the increase in number of high-end devices like smartphones or tablets has led to a spike in traffic (e.g., a smartphone can generate up to 35 times more traffic than a basic feature phone). Another important factor is the increase of average traffic per the devices themselves, which happened mainly because the network speeds have increased, while batteries have become more powerful. Thirdly, the increase in mobile video content has also led to congestion, because it has much higher bit rates than other content types. Furthermore, large screen sizes and mobile video optimization also contribute to the growth

of video traffic. As a fourth factor, we should mention the availability of mobile broadband services at low prices and speeds similar to those of fixed broadband.

Data offloading over Wi-Fi has caught traction in recent years, due to the ubiquitousness of wireless access points. There are considered to be two types of data offloading over Wi-Fi [9]: on-the-spot and delayed. On-the-spot offloading refers to using spontaneous connectivity to Wi-Fi and transferring data on the spot; when users move out of the Wi-Fi coverage, they discontinue the offloading and all the unfinished transfers are transmitted through cellular networks. Most of the smartphones which give priority to Wi-Fi over the cellular interface in data transmissions can be expected to currently achieve on-the-spot offloading. In delayed offloading, each data transfer is associated with a deadline and, as users come in and out of Wi-Fi coverage areas, they repeatedly resume data transfer until the transfer is complete. If the data transfer does not finish within its deadline, cellular networks finally complete it. Most smartphones with Wi-Fi are already performing on-the-spot offloading by default, but delayed offloading is relatively new. Its notion is very close to that of delay-tolerant networks where applications can tolerate some amount of delays. It is true that users want the data immediately, but network carriers can provide monetary incentives for users to transfer with longer deadlines.

A recent approach to offloading mobile data traffic using opportunistic communications has been proposed [8]. Most of the information delivered over mobile networks comes from content service providers and may include multimedia newspapers, small computer games, weather reports, and so on. The service providers can benefit from the delay-tolerant nature of such applications and may deliver the information only to a small group of users (target users). The target users can further disseminate the information to subscribed users when their mobile phones are in proximity and can communicate opportunistically using Wi-Fi or Bluetooth technology. Apart from these two technologies, device-to-device (D2D) communication using cellular resources can also be employed for such opportunistic communication. Such an offloading approach is attractive as there is little or no monetary cost associated with it. However, it is challenging due to a number of factors such as the heterogeneity of data traffic from service providers (varying in delay and content size), varied user demands and preferences for data traffic, incentives for target users, and battery and storage constraints of mobile devices.

Several frameworks for offloading data in mobile networks have been proposed recently, but they have several important limitations that we aim to address with our solution. Firstly, Huerta-Canepa and Lee propose an offloading framework where static identical devices are able to work together at parts of a series of common tasks [10]. However, the fact that the nodes are not mobile, they are all the same, and they can only communicate directly when in range (and not over multiple hops) are some downsides of this solution that are solved by the offloading mechanisms proposed in Section III.

A solution that also employs the cloud as an alternative to device-to-device communication is mCloud [11], but it has the drawback that it expects mobile network carriers to offer incentives for participation. Furthermore, it has the same limitation as all the other solutions, namely that it does not allow the mobile nodes to communicate opportunistically. A framework also called mCloud [12] further introduces the third dimension of offloading, composed of edge devices. In mCloud, node are thus able to offload over Wi-Fi, Wi-Fi Direct (but again in a single-hop fashion), Bluetooth and 3G, and the interface selection is performed using a multi-criteria optimization approach that considers several context parameters such as battery consumption or resource availability.

Verbelen et al. introduce the notion of cloudlets [13], which are groups of nodes located in the same network that can collaborate to help with computations (either for a node inside the cloudlet, or even for an external device). There are elastic cloudlets built specifically in datacenters, and ad-hoc cloudlets that get created when multiple devices that need to offload tasks are connected to the same network at the same time. The main drawback of this solution is that there needs to be a central service that is aware at all times of all the cloudlets and their locations, which defeats the purpose of decentralization. Furthermore, this would be even harder to implement if we took into account the fact that mobile nodes move around a lot and thus change their cloudlet membership very often.

## III. DATA AND TASK OFFLOADING MECHANISM

In this section, we first present the Drop Computing paradigm that our offloading mechanisms are implemented and tested on, and then we propose our solutions for various scenarios.

### A. DROP COMPUTING

The data and computation solution that we present here is based on the Drop Computing paradigm [7], [14], as shown in Figure 1. The main idea of Drop Computing is the introduction of an additional crowd computing layer below the classic fog/edge model, where mobile nodes can collaborate between each other without the need for access to the infrastructure. At this layer, the interactions between nodes are guided by social connections, which add a new and interesting dimension to distributed computing. Basically, this bottom layer is an opportunistic network [15] governed by the store-carry-and-forward paradigm.

Opportunistic networks (ONs) are a natural evolution of mobile ad hoc networks (MANETs), where most (or sometimes all) of the nodes are mobile wireless devices. In the case of ONs, routing information is not stored, due to the volatile nature of the links between nodes. ON nodes have no knowledge of the network when they join it, they are only aware of other nodes that they come in close contact with, depending on the radius of their communication mechanism (e.g., Wi-Fi Direct, Bluetooth, NFC, etc.). Secondly, the existence of a path between two devices is not guaranteed, since the topology is unknown (even if it would be known, a node

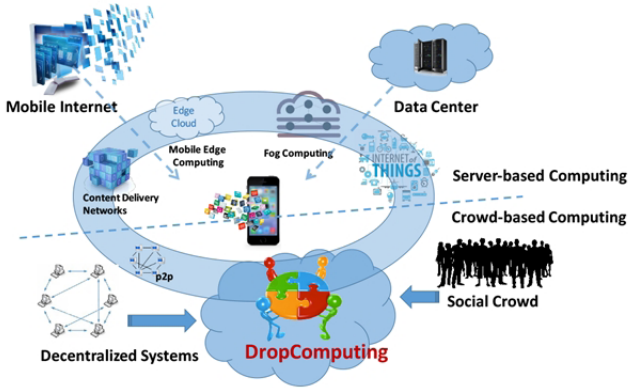


**FIGURE 1.** The Drop Computing paradigm.

can exit the local network anytime it wants to, or go idle for an undefined period of time.)

In such circumstances, traditional TCP/IP protocols or any other legacy Internet routing approach based on topological information will fail to accomplish the network needs. ONs address this issue through a store-carry-forward approach in which a node that wants to relay a message will store it and carry it around the network until the destination node is reached, or another node with a high chance of encountering the recipient is found. In this case, the message will be passed from the sender to the intermediate node, which now becomes the new sender.

However, opportunistic networks by themselves still have several unresolved challenges, which is why Drop Computing employs them only as a first layer in the communication process. The first challenge, and one of the greatest problems that must be tackled when developing routing algorithms for opportunistic networks, is the high degree of unreliability: not having to rely on any topology means that the routing will somehow have to constantly adapt to the new changes in topology, and also determine the best routes.

Secondly, since contacts between nodes are hardly predictable in real life (due to the complexities of human mobility patterns), some form of opportunistic decisioning must be taken in order to maximize the rate of messages reaching their destinations. Ideally, in order to correctly select the next hop, the future state and behavior of the network must be known, but this is not the case for ONs.

Another type of concern addresses the storage limitations that most of the existing hand-held smart device have: this directly impacts the number of messages one node can store. The bigger the storage space is, the more transient messages can exist in the network at a given time. If a node's message buffer gets full, it cannot receive any other messages unless some of the existing ones are dropped. This can eventually lead to the congestion of the network, in which case the network simply ''freezes'' and a waiting interval has to pass in order for the overwhelmed nodes to drop some of the messages.

Given the fact that the majority of mobile devices run using a limited amount of battery available, and that network operations are known for being power-demanding, the routing algorithms have to take into consideration the short amount of time in which data transfers should take place and the protocol being used: typically, Bluetooth is considered to be a low-power, low-energy protocol, whereas Wi-Fi or broadband technology such as 3G and 4G are much more energy-consuming.

Two other issues are privacy and security, which are unavoidable as the data being sent make their way throughout the network with the help of stranger nodes acting as forwarders. To prevent then from being tempered with, encryption methods must be employed that must assure both the integrity and the secrecy of the content being sent. Each routing algorithm designed for an opportunistic network must address those issues in order to provide a sustainable communication channel between parties.

In the case of Drop Computing, these issues are addressed by only employing the opportunistic network as part of a layered architecture. On top of this crowd-based computing, Figure 1 shows that there is also a component for server-based computing. This is where the fog and edge nodes are located, which can be employed by Drop Computing nodes when in range and when the nearby nodes cannot help with a request in a satisfying fashion. The fog and edge nodes are generally static devices that are located at the edge of the network, and that are used to alleviate the load on the cloud backend, and to increase response times for the requests due to their positioning closer to the devices being served.

If the edge and fog devices are not sufficient, then the cloud is employed. Thus, it can be observed that the Drop Computing paradigm has several levels of data and computation offloading: the ad-hoc opportunistic network composed of other nearby devices, the fog and edge nodes, and the cloud itself. It is in this context that we propose a solution for offloading in the next section.

### B. PROPOSED SOLUTION

On top of the Drop Computing paradigm presented in the previous section, we now propose offloading mechanisms that aim to improve the quality of experience (QoE) for users, the costs for developers, as well as battery consumption (which is of the utmost importance when dealing with mobile networks composed of battery-powered devices). In this paper, we focus on task computation as an offloadable action, but the solutions we propose can be easily mapped onto the offloading of data.

In [7], we introduced the Drop Computing paradigm and proposed a simple offloading model that only takes into account the cloud and other mobile devices located in close proximity. In [14], we extended this solution by adding fog nodes and presenting a more complex offloading mechanism and highlighting its advantages. In this paper, we take this one step further, by proposing a unified offloading solution for Drop Computing that takes into consideration all levels of the architecture presented in Section III-A and thoroughly analyzing it in various scenarios.

As previously seen, there are three kinds of entities in Drop Computing: mobile devices, fog nodes, and the cloud. This leads to four scenarios of offloading, and in the remainder of this section we propose a method for each of them:

- device-to-device (D2D) only - there are only mobile nodes in the network, so they can either compute the tasks themselves, or spread them in the opportunistic network composed of neighboring nodes
- cloud only - mobile devices do not communicate with each other, they can only compute the tasks themselves or employ the cloud when they consider this necessary (assuming unlimited resources in the cloud)
- D2D and cloud - in this scenario, mobile nodes can compute the tasks themselves or offload them opportunistically to other devices or to the cloud
- D2D, cloud and fog - in addition to the previous scenario, fog nodes are added here, which are located at the edge of the network, between the opportunistic layer and the cloud backend.

### 1) D2D ONLY OFFLOADING

This type of offloading is very similar to the basic idea of opportunistic routing or dissemination, where a source node sends a message that needs to reach a particular destination or a set of interested nodes. However, when dealing with task offloading, the communication needs to be bi-directional. First, the node that generates a task (its owner) needs to move it towards a good executor. Then, after the task is computed, it needs to make its way back to the owner, not necessarily on the same path (especially taking into account the fact that nodes in an opportunistic network are extremely mobile and thus do not spend much time in the same place). Another difference from opportunistic networks is the next hop selection criteria. In ON routing, the destination of a message is known in advance, so nodes that are encountered are analyzed based on their suitability of delivering the message closer to the destination (as shown by popular solutions such as BUBBLE Rap [16], PROPHET [17], ONSIDE [18], MLSOR [19], RANK [20], dLife [21], SPRINT [22], etc.). For ON dissemination, next hops are selected based on the tags of a message and of the interests of the encountered nodes (well-known proposals include ContentPlace [23], SocialCast [24], SANE [25], ONSIDE [18], etc.). In the D2D offloading case, when a task needs to be computed, devices are selected based on their resources and load, so different heuristics need to be employed. Once a task is computed, the logic of bringing it back to the owner behaves exactly like opportunistic routing (with a destination known in advance), so well-known algorithms can be employed for this stage.

Our proposed mechanism for offloading through D2D is presented in Algorithm 1. There are two main functions that need to be implemented for all four proposed solutions: *onDataExchange* and *onTick*. The former method is called whenever two nodes encounter and start a data exchange (from the standpoint of a node $A$ encountering a node $B$), whereas the latter is called on every clock tick. The *onTick* function is straightforward, as shown at lines 1-12 in Algorithm 1: each node simply computes as many tasks as it can from its list. This may include not only tasks that belong to the current node, but also tasks that other devices have offloaded. Whenever a task is completed, if it belongs to the current node then it is marked as solved and sent to the application level, otherwise it is put into a list of computed tasks, so it can then be delivered to its owner node.

**TABLE 1. Helper functions for the offloading mechanisms.**

| Helper function | Purpose |
|---|---|
| areBalanced(node1, node2) | checks if the two nodes have their tasks balanced |
| areSociallyConnected(node1, node2) | checks if the two nodes are socially connected |
| balance(node1, node2, drop) | balances the tasks of the two nodes; if the third parameter is true, then the nodes will not keep the tasks that they send each other |
| canCompute(node, task) | checks whether the node can compute the task in the current time tick |
| compute(node, task) | the node computes as much of the task as it can in this time tick |
| getContacts(node1, node2) | returns the number of contacts between the two nodes |
| getExecuted(node) | returns the list of executed tasks carried by the node |
| getOwner(task) | returns the owner of the task |
| getTasks(node) | returns the list of non-executed tasks carried by the node |
| hasExpired(task) | checks if the current task has expired |
| hasFinished(task) | checks if the current task has finished being computed |
| isFogNode(node) | checks if the current node is a fog node |
| markAsSolved(node, task) | marks the task as solved at the node |
| send(task, node1, node2) | sends the task from the first to the second node |
| sendToCloud(task) | sends the task to be executed in the cloud |

When two nodes $A$ and $B$ meet, the first step is to exchange information about completed tasks. If a completed task from node $A$ belongs to $B$, then a data exchange is performed, so $B$ now knows that its task was completed, although not necessarily by $A$ (lines 16-18). If a completed task's owner is not node $B$, then $B$ will only get the information about the completed task from $A$ if it is socially connected (on a network like Facebook, for example) with the task's owner (this is seen in lines 19-21 of Algorithm 1).

After exchanging information about completed tasks, the nodes decide which set of tasks each of them keeps to compute or further move in the network. By comparing the tasks that each of the two nodes carry, this is an attempt to solve the following optimization problem (where $A$ and $B$ are the nodes, $N.t$ is the set of uncomputed tasks carried by a node $N$, $N.cptu$ is the number of cycles per time unit that a

**Algorithm 1** D2D Only Offloading Mechanism (Helper Functions Are Presented in Table 1)

```
1:  procedure onTick(A)
2:      for all tasks t in getTasks(A) do
3:          if canCompute(A, t) then
4:              compute(A, t)
5:              if hasFinished(t) and getOwner(t) = A then
6:                  markAsSolved(A, t)
7:              end if
8:          else
9:              return
10:         end if
11:     end for
12: end procedure
13:
14: procedure onDataExchange(A, B)
15:     for all tasks t in getExecuted(B) do
16:         if getOwner(t) = A then
17:             send(t, B, A)
18:             markAsSolved(A, t)
19:         else if areSociallyConnected(A, getOwner(t))
    then
20:             send(t, B, A)
21:         end if
22:     end for
23:
24:     if getContacts(A, B) ≥ 2 then
25:         familiar = true
26:     else
27:         familiar = false
28:     end if
29:
30:     if ¬areBalanced(A, B) then
31:         balance(A, B, familiar)
32:     end if
33: end procedure
```

**Algorithm 2** Cloud Only Offloading Mechanism (Helper Functions Are Presented in Table 1)

```
1:  procedure onTick(A)
2:      for all tasks t in getTasks(A) do
3:          if canCompute(A, t) then
4:              compute(A, t)
5:              if hasFinished(t) then
6:                  markAsSolved(A, t)
7:              end if
8:          else
9:              break
10:         end if
11:     end for
12:
13:     for all tasks t in getTasks(A) do
14:         if hasExpired(t) then
15:             sendToCloud(t)
16:         end if
17:     end for
18: end procedure
```

node $N$ can perform, while $T.c$ is the dimension of a task in cycles):

$$\begin{aligned} &\text{minimize } \Big|\sum_{T\in A.t} T.c \times A.cptu - \sum_{T\in B.t} T.c \times B.cptu\Big| \\ &\text{subject to } A.t \cap B.t = \emptyset \end{aligned} \tag{1}$$

This is a bound constrained optimization problem that should balance the computation load on the two nodes by minimizing the difference between the total computation durations of the tasks carried by the two nodes, calculated as the product between the number of cycles per task and the duration of a cycle per node. In Algorithm 1, this step is performed by the *balance* method as seen at line 31 (the *areBalanced* method invoked at line 30 simply checks if the difference between the loads on the two devices is higher than a given threshold). After the minimization problem is solved, the nodes exchange the necessary tasks between each other, in order to remain with the most optimal task set.

For solving the problem, the *balance* function employs a greedy approach: node $A$ selects the task that takes the longest for it to compute, then $B$ chooses one or multiple tasks that take a similar amount of time, and so on.

When node $A$ transfers a task to node $B$, it has two options. It can either keep the task even after transfer (in which case it can work on computing it in the future or pass it along to other nodes) or it can delete it (and then only node $B$ decides what to do with the task). In order to make this decision, we analyze the familiarity between the two encountering nodes, as seen at lines 24-28 of Algorithm 1. Thus, if there have been at least two contacts between $A$ and $B$ in the last time window, then the *familiar* variable is set to true and the sending node does not keep the task. This happens because we consider that, when two nodes are familiar, there is an inherent trust in each other. Furthermore, this may also mean that, since $A$ and $B$ meet often, they will both encounter a similar set of neighbors.

### 2) CLOUD ONLY OFFLOADING

The second type of offloading is when devices do not communicate with each other opportunistically, but they only have access to the cloud. This is actually the classic case of mobile cloud computing [1], so our proposal for this scenario, as shown in Algorithm 2, is relatively straightforward. More specifically, the first part (lines 2-11) is very similar to the one for D2D offloading from Algorithm 1 (the only difference is that, for this case, a node can only compute tasks that belong to itself, so it simply marks them as finished and sends them to the application level upon execution). Whenever a task expires, the cloud is employed to help with the computations. Expiration times depend on the size of the task and the computation expectancy of the user of the particular application

**Algorithm 3** D2D and Cloud Offloading Mechanism (Helper Functions Are Presented in Table 1)

```
1:  procedure onTick(A)
2:      for all tasks t in getTasks(A) do
3:          if canCompute(A, t) then
4:              compute(A, t)
5:              if hasFinished(t) and getOwner(t) = A then
6:                  markAsSolved(A, t)
7:              end if
8:          else
9:              break
10:         end if
11:     end for
12:
13:     for all tasks t in getTasks(A) do
14:         if hasExpired(t) then
15:             sendToCloud(t)
16:         end if
17:     end for
18: end procedure
19:
20: procedure onDataExchange(A, B)
21:     for all tasks t in getExecuted(B) do
22:         if getOwner(t) = A then
23:             send(t, B, A)
24:             markAsSolved(A, t)
25:         else if areSociallyConnected(A, getOwner(t))
    then
26:             send(t, B, A)
27:         end if
28:     end for
29:
30:     if getContacts(A, B) ≥ 2 then
31:         familiar = true
32:     else
33:         familiar = false
34:     end if
35:
36:     if ¬areBalanced(A, B) then
37:         balance(A, B, familiar)
38:     end if
39: end procedure
```

that is using the offloading mechanisms (generally in terms of quality of experience).

#### 3) D2D AND CLOUD OFFLOADING

The D2D and cloud offloading mechanism presented in Algorithm 3 is basically a combination of the D2D and cloud only solutions shown in Algorithms 1 and 2. More specifically, nodes compute the tasks they have in their lists upon each tick (lines 2-11) and send them to the application layer or store them to be disseminated to other nodes. Upon a D2D contact, the nodes first exchange information about executed tasks (lines 21-28), and then attempt to balance their tasks (as shown at lines 30-38 in Algorithm 3). The advantage of this solution over the two versions presented in Sections III-B.1 and III-B.2 is that more tasks will be computed because they are sent to the cloud when they expire, but at the same time the costs for the developers will be lower, because the cloud will not be used as much as in the cloud-based solution.

#### 4) D2D, CLOUD AND FOG OFFLOADING

When adding fog nodes into the network, the logic needs to be changed in order to account for and take advantage of these devices. Our proposal is shown in Algorithm 4, and it can be observed that the behavior of the *onTick* method is the same as for the D2D and cloud approach: devices compute themselves as many tasks as they can at each tick, and they offload to the cloud all the tasks that expire upon a tick.

The first different approach of this solution occurs when exchanging executed tasks. In this case, if a mobile node encounters a fog device, it sends its entire list of executed tasks. This happens because fog nodes have larger storage space and they are placed in locations where they have many contacts with other devices, which increases the chance of an executed task to find its owner. This is shown at lines 26-28 of Algorithm 4.

There is also a special case regarding tasks that have not been fully computed yet, as observed at lines 40-53 of Algorithm 4. In this situation, a mobile node will pick one task from its list, offload it to the fog node, wait for the executed task, and then repeat this action until the two nodes are no longer in contact or all of the mobile device's tasks are executed. Upon the execution of each task, the mobile node sends it to the application level or stores it for future delivery, depending on the case. If the mobile node is not the task's owner, the fog node also keeps the result in its list of executed tasks, in order to improve the task result delivery. By employing the fog devices, mobile nodes take full advantage of their higher computing power whenever they are in range.

## IV. ANALYSIS

In this section, we present a thorough analysis of the behavior of the offloading mechanism proposed in Section III. We start by describing the various scenarios that we simulated and the way they are implemented, and then we analyze the results obtained by our solution, highlighting its advantages and the places where it can be improved.

### A. EXPERIMENTAL SETUP

This section addresses the experimental setup of our simulations. We first describe the MobEmu framework that was used for all our tests, and then go into detail regarding each scenario and the behavior of all entities involved (mobile nodes, fog devices, the cloud, etc.).

#### 1) MobEmu

MobEmu [26] is an open-source mobile network simulator[2] that can replay a mobility trace or a synthetic mobile

[2]Its code is freely available under MIT license at https://github.com/raduciobanu/mobemu.

**Algorithm 4** D2D, Cloud and Fog Offloading Mechanism (Helper Functions Are Presented in Table 1)

```
procedure onTick(A)
    for all tasks t in getTasks(A) do
        if canCompute(A, t) then
            compute(A, t)
            if hasFinished(t) and getOwner(t) = A then
                markAsSolved(A, t)
            end if
        else
            break
        end if
    end for

    for all tasks t in getTasks(A) do
        if hasExpired(t) then
            sendToCloud(t)
        end if
    end for
end procedure

procedure onDataExchange(A, B)
    for all tasks t in getExecuted(B) do
        if getOwner(t) = A then
            send(t, B, A)
            markAsSolved(A, t)
        else if areSociallyConnected(A, getOwner(t)) then
            if isFogNode(A) then
                send(t, B, A)
            end if
        end if
    end for

    if getContacts(A, B) ≥ 2 then
        familiar = true
    else
        familiar = false
    end if

    if ¬areBalanced(A, B) and ¬isFogNode(B) then
        balance(A, B, familiar)
    else if isFogNode(B) then
        for all tasks t in getTasks(A) do
            if canCompute(B, t) then
                send(t, A, B)
                compute(B, t)
                send(t, B, A)
                if hasFinished(t) and getOwner(t) = A then
                    markAsSolved(A, t)
                end if
            else
                break
            end if
        end for
    end if
end procedure
```

node interaction model and allow the users to implement their own logic on top of these interactions. It started out as a tool for testing opportunistic network routing and dissemination behavior on large-scale scenarios (having implementations for solutions such as Epidemic [27], Spray-and-Wait [28], BUBBLE Rap [16], Interest Spaces [29], etc.), but it has evolved to allow for the existence of a cloud backend, edge and fog devices, and other specialized network entities that need to be simulated in various mobile network-related
scenarios.

Although other opportunistic network simulators existed at the time MobEmu was created (such as ONE [30], DTN2,[3] or OMNet++,[4]) they had several disadvantages, such as lack of support for opportunistic dissemination, community detection, social connections, context data, etc., which is the reason that MobEmu was implemented.

It has a very intuitive interface, where a *Node* class is used to simulate a network entity, and two methods need to be implemented to guide the behavior of a mobile node: *onTick* (called at every clock tick of the simulation) and *onDataExchange* (called when two nodes are in contact and start communicating). Aside from this, each node has several additional components that can be attached, such as *Battery* (to simulate battery consumption, as detailed in Section IV-A.5), *Network* (for network transfer behavior using various close-range protocols), *Community* (for node community grouping), etc.

#### 2) TRACES

For testing the solution proposed in Section III, we used one real-life mobility trace collected in an office environment (where the number of mobile nodes and their interactions are fixed and cannot be controlled in the simulation) and the HCMM mobility model [23], which simulates device contacts based on approximations of the human behavior (and which allows for a high control on the number of nodes in the simulation and their grouping, the size of the simulation area, the number of contacts, etc.).

The real-life trace (entitled "UPB 2015") was collected for a period of several weeks in an office in Bucharest, but for the experiments we perform here, we selected a sample obtained in a 7-hour interval of a single day (since it is a working environment where the same employees work and interact, the trace is very similar on a daily basis). For collection, we employed the HYCCUPS Tracer [31], which is an Android application that collects information about the behavior of a mobile device, including battery consumption, CPU behavior (frequencies, hotplugging, sleep states, etc.), memory management, user activity, and also contacts between nodes. For contact tracing, HYCCUPS uses the

[3]https://sites.google.com/site/dtnresgroup/home/code/dtn2documentation.

[4]https://omnetpp.org.

**TABLE 2. Testing scenarios in MobEmu. For UPB 2015, the parameters marked with "N/A" cannot be controlled when running the trace.**

| Parameter | UPB 2015 | HCMM |
|---|---|---|
| Environment | Office | Synthetic |
| Number of mobile nodes | 6 | 40 |
| Number of static fog nodes | 2 | 2 or 4 or 10 |
| Contacts | 361 | 1624 or 25412 |
| Duration | 7 hours | 15 hours |
| Node speed | N/A | 1.25 - 1.5 m/s |
| Simulation grid dimensions | N/A | 400x400 m |
| Node transmission radius | N/A | 10 m (Bluetooth) |
| Number of communities | N/A | 1 or 4 |
| Number of traveler nodes | N/A | 1 per community |

AllJoyn framework[5] to detect interactions over Wi-Fi access points and Bluetooth.

On the other hand, the Home-Cell Mobility Model (HCMM) is a synthetic model that generates user interactions that try to mimic real-life contacts between humans carrying mobile devices, by following the caveman model [32]. The advantage of HCMM is that it allows us to configure the behavior of the mobile network through various parameters, as shown in Table 2.

As shown in Section III, our solution assumes that there are also fog devices in the network, which can alleviate some of the load placed on mobile devices or the cloud. For this reason, support for such devices was needed in MobEmu. For HCMM, we simply added the desired number of static fog nodes in the simulation, and generated contacts whenever mobile devices were in range, which allowed us to vary the number of fog nodes. It can be seen in Table 2 that, for our experiments, we tested with 2, 4 and 10 static fog nodes. On the other hand, since UPB 2015 is a real-life trace and we wanted to test with real data, we placed two static nodes in locations at our faculty (in a large amphitheater and in a laboratory) and configured their Wi-Fi interfaces in monitor mode and listened for Wi-Fi beacons from other devices in range [14]. We then mapped the most seen nodes to the 6 nodes in the UPB 2015 trace (choosing the 7-hour interval with the most contacts) and ran our experiments this way.

#### 3) TESTING SCENARIOS

In order to highlight the advantages brought forth by our proposed solution, we devised several scenarios. Firstly, we test a scenario where there is only device-to-device communication. In this situation, if a node needs to compute a task, it can either perform this computation itself, or it can request help from other nodes in the network when they are in range. Requests for computations can be forwarded between the nodes in a multi-hop opportunistic fashion, so a task may end up being solved by a node that is never in contact with the task's owner. In this situation, the difficulty lies in correctly choosing which node to forward the task to, and then which node to use as a relay for the result of task, in order for it to make its way back to its original owner.

For the second scenario, we assume that nodes do not communicate with other devices, but they have access to the cloud (so this is the classic mobile cloud computing scenario). In this case, they can either compute the tasks themselves or they can employ the cloud backend for help.

The third scenario sees the mobile nodes able to communicate with each other (D2D), while also having access to the cloud, which means that they need to carefully decide when to request help from their neighbors (e.g., when the other devices in range have the necessary capabilities and they are predicted to be encountered again shortly) and when to go to the cloud.

Finally, the fourth test case is the complete scenario that the solution we presented in Section III attempts to address, namely when nodes can compute the tasks themselves, ask for help from neighbors, go to the fog devices when in range, or compute the tasks in the cloud.

We execute these four scenarios for the two traces presented in Section IV-A.2, in order to see how our solution behaves for different environments. Furthermore, we run our solution on two versions of the HCMM scenario. In the first version, the 40 nodes are grouped into 4 communities, which means that nodes inside a community will tend to encounter each other a lot, whereas interactions between nodes from different communities are rarer. In this scenario, each community also has a traveler node, which moves through the other communities with a higher probability than regular nodes, acting thus as an inter-community connection. Aside from this scenario, we also test with all nodes in a single large community, which increases the contact duration of any two nodes while decreasing the overall number of contacts (since the two devices spend more time together and move apart rarer), thus potentially making the D2D-only case more optimal.

#### 4) MEASURED METRICS

For each scenario, we measure several metrics that we consider to be important in deciding whether the solution we propose is suitable. The first metric is computation duration, which is the time passed between the moment a task is generated and the moment that its owner has the result (that was either computed locally, on another mobile or fog node, or in the cloud). This metric is closely related to a user's quality of experience, since the quicker a task is solved, the more satisfied the user is. Then, the second metric is the cloud usage time, which needs to be improved from the standpoint of the entity that offers the mobile service. For example, if we are talking about a mobile application where CPU-intensive tasks are offloaded to the cloud, a large number of such offloads leads to higher costs for the application developers. Ideally, a good solution should be able to minimize both the computation duration and the cloud usage time, thus satisfying the users and application developers alike. We also compute the usage time per device (i.e., when the node

[5]https://openconnectivity.org/developer/reference-implementation/alljoyn.

itself is performing computations), as well as the fog node computation time, in order to see how often these nodes are employed.

Another important metric when talking about networks composed of mobile devices is the battery consumption. For this reason, in our experiments we measure how much battery is consumed overall by a mobile device (as a percentage of the fully-charged battery), but also how much is consumed for each of a device's actions (i.e., exchanging data with other devices, with the fog nodes, or with the cloud, and computing tasks). In Section IV-A.5, we discuss the battery model employed in our experiments.

#### 5) BATTERY MODEL

Several battery models for embedded systems have been proposed over the years, including kinetic, stochastic or Markov process-based models [33]. However, MobEmu currently opts for a more simpler model, but one that we believe is closer to reality if configured correctly. More precisely, for each type of device that MobEmu is able to simulate, we have used real-life data (generally collected from GSM Arena[6]) to estimate the lifetime of a mobile device when used normally (i.e., when no data transfers and task computations are performed). Based on the current trace's sample time, the initial battery is set so that, at each clock tick, it decreases linearly by one unit, and the time it takes for the battery level to get from maximum value to zero is equal to the battery lifetime as indicated by GSM Arena information.

Then, for each type of action that is bound to consume more battery (transferring data and computing tasks), we apply a multiplier, so that, for example, while a task is being computed, the battery will decrease by the value of the multiplier (instead of one unit) per time tick. If there are several battery-consuming actions being performed simultaneously at one point (for example, a device is sending some tasks to the cloud, while at the same time computing other tasks), the battery multipliers are added, so both actions contribute to the consumption of battery.

In order to select the battery consumption multiplier values (which are presented in Table 3), we performed several experiments on multiple mobile devices (a Samsung Galaxy S7, an iPhone 5S and a Google Pixel) where we observed how much lower a device's lifetime is when transferring data on Wi-Fi (for fog node communication), 4G (for cloud communication) and on Bluetooth (for D2D communication), and also when running a task that keeps one core in 100% usage. The values obtained for the three devices we tested were relatively similar, so we approximated them as shown in Table 3. We should mention that, when performing the battery experiments that led to these values, we took into account the fact that, when smartphones keep their CPUs in high frequencies for longer periods of time, they tend to overheat and end up consuming even more power, so we allowed for cool-down intervals between short periods of running tasks or transferring data.

[6]https://www.gsmarena.com.

**TABLE 3. Battery consumption multipliers used for testing in MobEmu.**

| Consumer | Multiplier value |
|---|---|
| D2D data transfer | 1.65 |
| Fog node data transfer | 2.87 |
| Cloud data transfer | 5.74 |
| Task computation | 2.70 |

#### 6) TASKS

In order to analyze our proposed solution in multiple scenarios, we also varied the kinds of tasks that a node needs to execute. Consequently, there are three types of computational tasks in our experiments: small (1 Mcycle), medium (1000 Mcycles) and large (10000 Mcycles). Every time a node remains out of tasks to execute, it generates a new set. Tasks are organized into task groups, which pertain to the same application, actually representing one computation that is split into multiple offloadable components. A task group consists of between 1 and 20 tasks of each type. For all our experiments (as presented in Section IV-A.3), we limit the total number of tasks per scenario to 60000.

Each individual task from a task group has an expiration time. If no result arrived for that task (from a mobile or a fog node) before the expiration time, then the task is sent to the cloud for computing, so that it does not delay the application and affect the user's experience. This is why, when offloading a task to another node, the proposed offloading solution needs to take into account the time left prior to the task's expiration, in order to correctly decide where it is worth to offload the task or if it is not better to compute it in the cloud. For small tasks, the expiration time is 1 millisecond, while for medium and large tasks it is one second and ten seconds, respectively.

When tasks are executed in the cloud, we assume that there is an unlimited number of single-core virtual machines that have 3.3 GHz CPUs. When executing tasks on mobile nodes, we only use one core for task computation, in order to not affect the QoE of the owner of the mobile device. When offloading a task, all its data need to be transferred. In our experiments, we assume that all tasks (regardless of type) have 5 MB.

#### 7) MOBILE DEVICES

MobEmu is able to simulate several types of mobile devices (smartphones, in this case), having support for LG G5, HTC One M9, iPhone 5S, iPhone 6S, Samsung Galaxy S4, Samsung Galaxy S5, Samsung Galaxy S6, and Samsung Galaxy S7. A device is characterized by its battery lifetime and the duration of computing a megacycle on one CPU (information obtained from GSM Arena), as we assume that transfer speeds are given by the protocols employed, and not necessarily by the hardware.

For all our experiments in this article, we test with half of the devices being iPhone 5S and the other half Samsung

**TABLE 4. Mobile and fog device characteristics.**

| Device | Mcycle duration | Full charge duration |
|---|---|---|
| iPhone 5S | 0.769 ms | 44 h |
| Samsung Galaxy S7 | 0.535 ms | 59 h |
| Fog | 0.200 ms | N/A |

Galaxy S7. We chose this combination because there is a relatively significant advantage regarding CPU power for the Samsung device, and we wanted to see what effect this disparity has on our solution. The parameters for the two types of devices, along with the fog nodes, can be observed in Table 4. Regarding the fog nodes, we assumed that they are static nodes always connected to a power outlet (thus no battery limitations), which can perform task computations on four cores simultaneously.

### B. RESULTS

This section presents the results obtained when running experiments on the scenarios described in Section IV-A. We begin by analyzing the behavior of our solution on the HCMM model when the nodes are grouped into four social communities, then we only set one community for the same environment, and finally we show the results obtained on the UPB 2015 mobility trace. All tests in this section have been performed 5 times each, with average values being shown in the charts.

#### 1) HCMM RESULTS (FOUR COMMUNITIES)

Figure 2 shows the computation duration and cloud usage time for the case when the 40 nodes in the HCMM simulation are split evenly into four communities. Firstly, it can be observed that, for the D2D-only case, the computation duration is much higher than for all the other three scenarios. This is caused by the fact that nodes in this scenario do not have access to a cloud backend, so all computations must be performed by themselves or by the other devices in range. This means that, when a task expires, there is no other backup option as is the case for the other tests. Furthermore, because this is an environment where nodes are split into communities, only nodes from the same community can be good offloaders for a task, because there is a low chance that nodes from different communities encounter each other (so the number of potential devices to offload to is a quarter of the network). Moreover, the D2D case is the only situation where less than the maximum number of tasks is not reached. As we specified in Section IV-A.6, we limit the total number of tasks per scenario to 60000, but in the case of D2D-only offloading, only 45782 tasks end up being computed. This shows that having a cloud backend can drastically improve the computation, but this is done at the cost of renting cloud resources by the application or service providers.

Figure 2 also shows that the solution we propose in Section III is able to reduce both computation time (by at least 19.42%) and cloud usage time (with more than 40.24%) when compared to the classic mobile cloud computing case

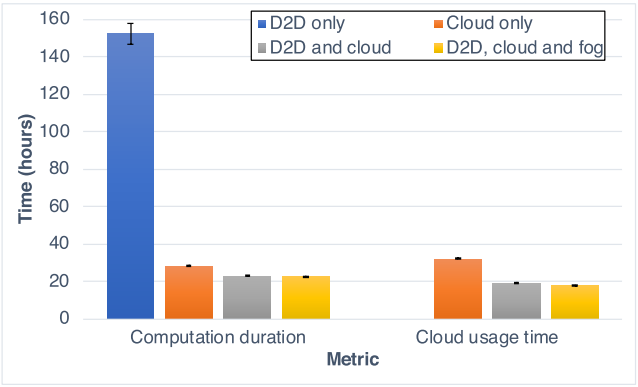


**FIGURE 2. Computation duration and cloud usage time for the HCMM scenario with four communities.**

(i.e., when nodes can only offload their computations to the cloud or perform them themselves). This means that the users are more satisfied (since their computations are performed faster, leading to a better QoE), and the application providers are able to lower their deployment costs in the cloud. It can also be seen that adding two fog nodes further improves the computation duration (by 1.11%) and cloud usage (by 7.8%) when compared to the D2D and cloud scenario. It should also be mentioned that, for all four situations, the mobile devices were used continually (computing tasks for themselves or for other nodes, or transferring data), while the fog nodes were used for a total of 0.54 hours.

Figure 2 shows that adding two fog nodes improves both metrics analyzed, but not by much, because there are not many contacts with them, which leads to few opportunities to employ them. Furthermore, even if a node encounters a fog device quickly, it might not get a chance to encounter it again and get the result of a task back before the task expires. For this reason, we tested with more fog nodes, and the results are shown in Figure 3. It can be seen that, while the computation duration is indeed reduced, the benefits are not that high. There is only an improvement of 0.7% when employing ten fog nodes instead of four. This is most likely caused by the fact that the mobile network in this scenario is extremely dynamic, so mobile devices do not meet the static fog nodes very often, and even then the contact durations are low, which caps the amount of data that can be exchanged. In terms of cloud usage time, the benefits are slightly better (3.8%), but the values are still small, which leads us to draw the conclusion that the number of specialized fog nodes does not have to be too high when compared to the number of nodes in the network, since other mobile devices themselves can take some of the load. For this particular test case, 2 fog nodes (i.e., 5% of the total number of network nodes) is sufficient for a good QoE and reduced deployment costs. Furthermore, fog nodes are also more expensive than simply using the regular mobile nodes, so keeping a low number in this network is an advantage.

For the four cases presented in Figure 2, we also measured the average battery consumption per mobile device type, assuming that each node starts the experiments with a

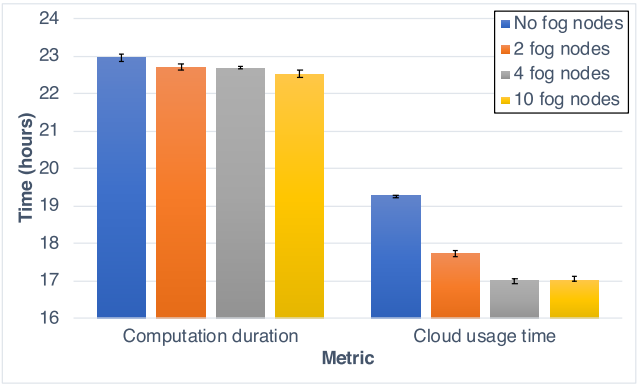


**FIGURE 3.** The influence of the number of fog computing nodes on computation duration and cloud usage time for the HCMM scenario with four communities.

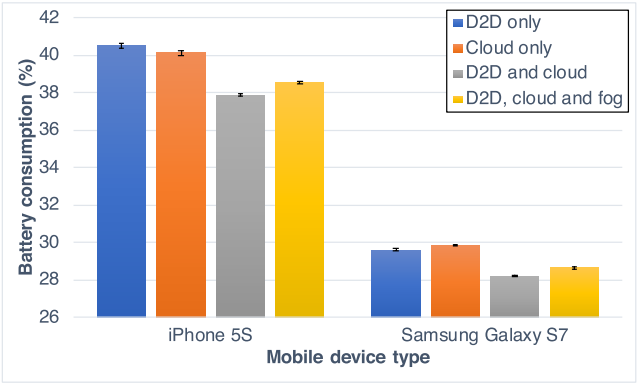


**FIGURE 4.** Average battery consumption per mobile device type for the HCMM scenario with four communities.

full charge. The results are shown in Figure 4, where it can be observed that our proposed offloading solution (with or without fog nodes) is able to reduce the battery consumption when compared to the D2D-only and classic mobile cloud computing cases. For the Samsung Galaxy S7 devices (which in average consume about 25% less battery than the iPhone 5S nodes), our proposed solution decreases battery depletion by 5.4%, whereas for the iPhone 5S nodes our solution brings a reduction of about 6.5%. The numbers may not be that significant in and of themselves, but they should be put into perspective. Namely, our offloading solution is able to reduce computation time (i.e., improve user QoE) and decrease cloud usage (i.e., lower the costs for the developers), while at the same time consuming less battery for the mobile devices, which is an important metric for smartphone users.[7] As another interesting aspect to mention, Figure 4 shows that a mobile device consumes the same amount of power regardless whether there are fog nodes in the network or not.

We wanted to delve deeper into battery consumption, so we extracted the distribution of power consumption per activity in each scenario for the two types of devices, as shown in Figures 5 and 6. The four activities considered are D2D

[7]https://www.forbes.com/sites/antonyleather/2013/12/13/why-battery-life-should-be-the-new-smartphone-battleground/.

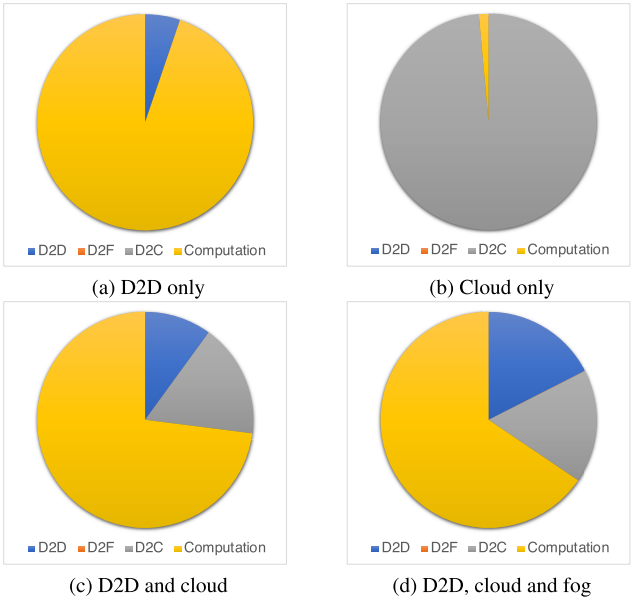


**FIGURE 5.** Distribution of battery consumption per activity for the HCMM scenario with four communities (iPhone 5S devices).

communication (over Bluetooth), device-to-cloud (D2C) communication (using 4G), device-to-fog node (D2F) communication (with Wi-Fi), and task computation. The two sets of charts show that, for all scenarios excluding the cloud-only case, the most battery is consumed by a mobile device when computing tasks (either for its own benefit, or for other nodes in the network). For the mobile cloud computing scenario (shown in Figures 5b and 6b), a node only computes a task until it expires and then sends it to the cloud. Thus, nodes that tend to generate more tasks need to contact the cloud more often (since they cannot deal with all the tasks themselves, and there is no other entity that can help), leading to the behavior shown in Figures 5b and 6b (for both types of devices, 98.5% of the battery consumption belongs to cloud communication).

For the D2D-only case, while devices compute continuously as long as there are tasks available, the majority of battery consumption is caused by computation on the nodes, as shown in Figures 5a and 6a (94.75% for the iPhones and 95.72% for the Samsung nodes). This happens because nodes communicate in a probabilistic fashion in this scenario, based on heuristics and context information, so the same task needs to be sent on multiple paths in order to find a node that is willing and capable of computing it, thus ending up potentially being computed multiple times in the network.

When allowing mobile devices to use both the cloud and other nodes (fog or mobile), the distribution of battery consumption is skewed less towards computation, as seen in Figures 5c, 5d, 6c, and 6d. This happens because the nodes try to look for the better option for each task, which might mean using other nodes or the cloud. For the D2D and cloud case, iPhone 5S devices spend 10% of their battery on D2D communication and 16.98% on data exchanges with the cloud. When also adding fog nodes, although the data

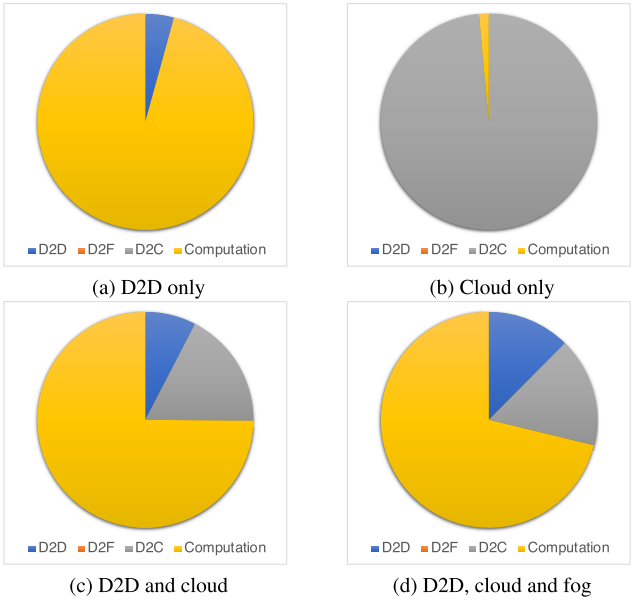


**FIGURE 6.** Distribution of battery consumption per activity for the HCMM scenario with four communities (Samsung Galaxy S7 devices).

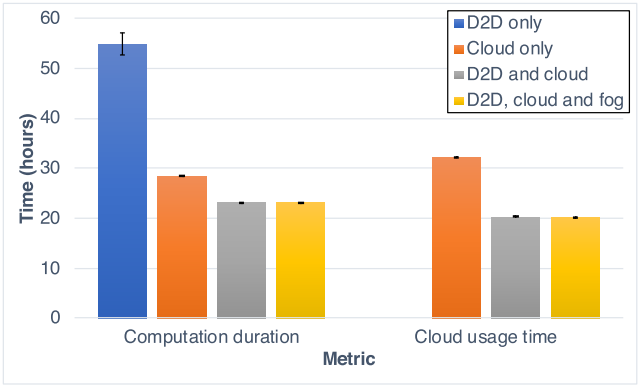


**FIGURE 7.** Computation duration and cloud usage time for the HCMM scenario with one community.

exchanges with them only consume 0.05% for iPhone 5S devices, their introduction in the network has other effects. More specifically, Figure 5d shows that more battery is consumed by D2D communication (17.49%), as the amount of computations performed on a device decreases (with the value being 65.54%). This happens because the results of the tasks computed by the fog devices need to make their way back to their owners, so the nodes need to work harder in ensuring that they are spread optimally in order to reach their owners. Figure 6d shows that similar behavior is also exhibited by the Samsung Galaxy S7 devices, albeit with less D2D usage (since the nodes are more powerful than the iPhones and can compute their own tasks quicker).

### 2) HCMM RESULTS (ONE COMMUNITY)

We have seen in the previous section that the scenario where nodes are only able to communicate using close-range protocols has very high values for the total computation duration,

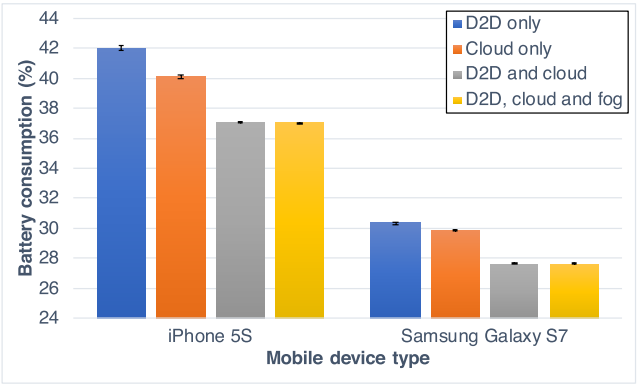


**FIGURE 8.** Average battery consumption per mobile device type for the HCMM scenario with one community.

which is caused by the fact that nodes from different communities encounter each other rarer, so only nodes in the same community can help each other. Furthermore, nodes in this scenario tend to have lower contact durations, so a smaller amount of tasks can be exchanged upon a contact. For this reason, in this section we test our solution on a version of this scenario where all nodes are part of the same community. As shown in Table 2, this leads to a smaller number of contacts (1624 as opposed to 25412), but their durations are higher.

The first effect of having a single community can be observed in Figure 7, namely that the total computation duration for the D2D case is drastically decreased in comparison to the scenario with four communities (from 152.15 to 54.88 hours). Furthermore, the number of tasks that can be computed in this scenario is improved by 26%, ending up very close to the maximum 60000 tasks per simulation. For the other three test cases, the values obtained are similar to the results achieved when testing with four communities: the computation duration is decreased when offloading can be performed in the cloud or on other nodes (fog or mobile), while the cloud usage time is also reduced when compared to the classic mobile cloud computing scenario. When comparing Figure 7 with Figure 2, it can be seen that the computation duration for the one-community scenario is slightly higher than when running with four communities, while the cloud usage time is decreased. This is most likely cause by the fact that, because devices spend more time in contact with each other, they have the tendency to help each other more often, thus reducing the number of accesses to the cloud (and in the process slightly increasing the overall computation duration).

We measured average battery consumption per mobile device type for this scenario as well, as shown in Figure 8. The results show that allowing devices to offload tasks to each other and to fog nodes helps decrease battery consumption by as much as 6.5%. However, what is most important to note when comparing Figure 4 to Figure 8 is that, when the mobile nodes are in the same community and they only have D2D connectivity, they not only compute 26% more tasks 2.7 times faster, but they also consume less power in the process (for

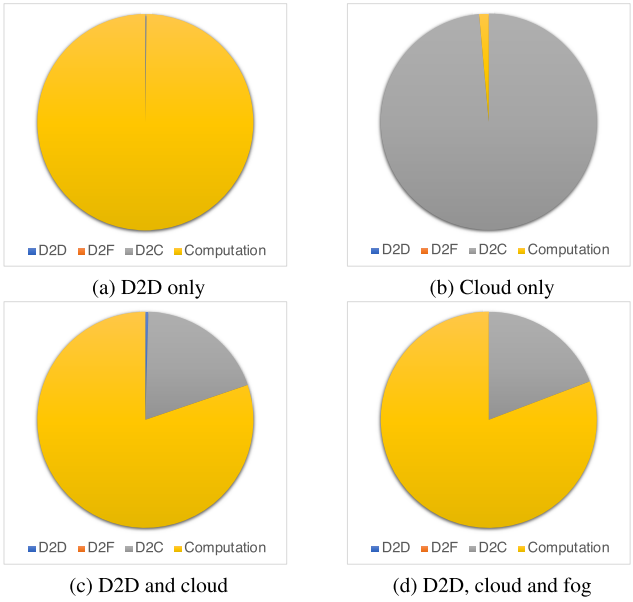


**FIGURE 9.** Distribution of battery consumption per activity for the HCMM scenario with one community (iPhone 5S devices).

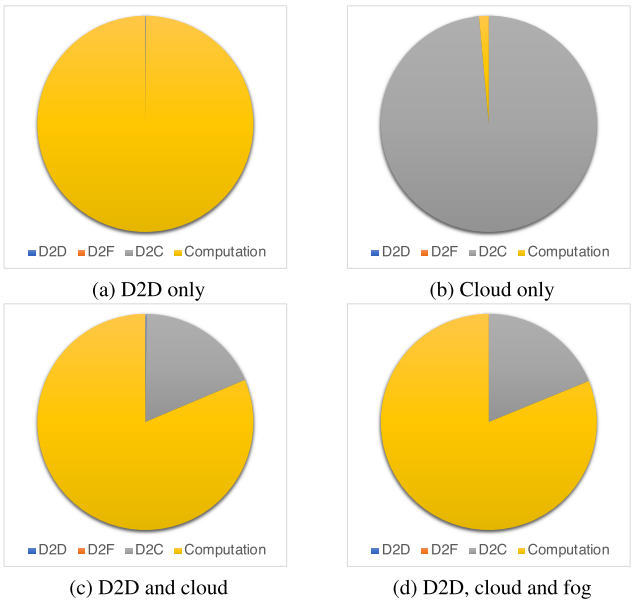


**FIGURE 10.** Distribution of battery consumption per activity for the HCMM scenario with one community (Samsung Galaxy S7 devices).

iPhone 5S devices, the power reduction is 3.6%, whereas for the Samsung Galaxy S7 nodes it is 2.3%).

We also computed the distribution of battery consumption per activity, as shown in Figures 9 and 10. What stands out from these charts at first glance is that, in the scenarios where D2D communication is possible, the battery consumption is barely affected by it. For the D2D-only case, only 0.1% of the energy consumption is caused by close-range communication, for both iPhone and Samsung devices. When also allowing cloud offloading, only 0.4% of the power consumption of iPhone nodes and 0.2% of the energy consumption

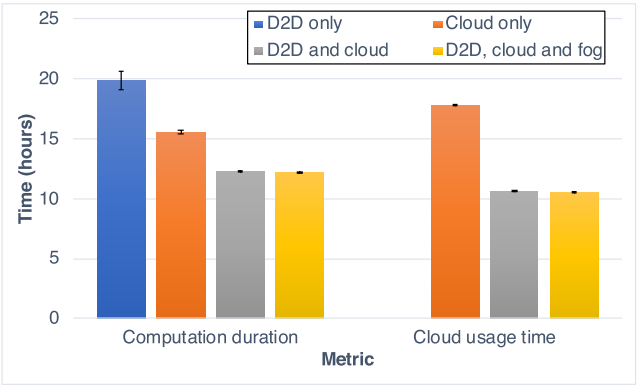


**FIGURE 11.** Computation duration and cloud usage time for the UPB 2015 scenario.

of Samsung devices is prompted by D2D data exchanges. Finally, less than 0.01% of battery drain is given by D2D communication when also adding fog nodes (in this case, data exchanges with the fog nodes also barely affect the battery consumption). This highlights the fact that, if there is a single large community, a node has an equal chance of encountering any other node, which means the results of a task computation offloaded to a given node make their way much easier towards the task's owner. Thus, instead of, for example, a task result going through seven hops between the device that computes the task and the owner, it might only pass through one or two nodes, which drastically decreases the number of D2D transfers in the network and consequently the battery consumption caused by these transfers.

### 3) UPB 2015 RESULTS

For the next set of results, we ran our experiments on the real-life UPB 2015 trace, collected in an office environment from six participants. The results for computation duration and cloud usage are shown in Figure 11, and it can be observed that, similarly to the HCMM scenarios, our proposed solution is able to reduce the computation duration and cloud usage at the same time, when compared to the mobile cloud computing and D2D-only cases. More specifically, the D2D and cloud scenario brings an improvement of 38.29% (i.e., 7.59 hours) in terms of computation time over the D2D-only case, and of 21.35% (i.e., 3.32 hours) over the classic mobile cloud computing scenario. When measured against the latter scenario, our D2D and cloud solution is able to also decrease the total cloud usage time by 7.2 hours, which is the equivalent of a 40% improvement. When adding the two fog nodes, the computation duration and cloud usage time are also slightly improved. Unfortunately, in this scenario we could not control the number of fog devices because we used real data, but the conclusions obtained when analyzing Figure 3 showed that one fog node per twenty mobile devices was sufficient. In the UPB 2012 scenario, there are two fog nodes for six nodes, so we consider this to be an appropriate value, and the results to be valid.

Because the number of nodes and the simulation duration are lower for the UPB 2015 scenario than for the HCMM

**TABLE 5. Total number of tasks computed in the mobile network for the UPB 2015 scenario.**

| Test case | Compute tasks |
|---|---|
| D2D only | 15041 |
| Cloud only | 33315 |
| D2D and cloud | 32098 |
| D2D, cloud and fog | 32170 |

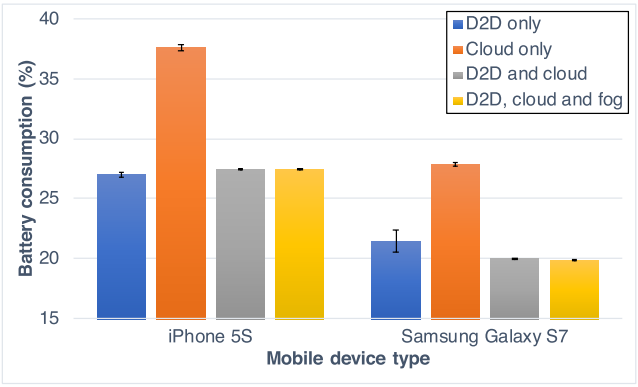


**FIGURE 12. Average battery consumption per mobile device type for the UPB 2015 scenario.**

tests, the maximum number of tasks (set to 60000) cannot be achieved by any test case. Table 5 shows how many tasks can be computed by all the nodes in each scenario. It can be observed that the D2D-only test case has by far the worst behavior, managing to compute less than half of the number of tasks that the other scenarios do. This is caused by the low number of contacts and devices in this trace, which do not offer many chances of offloading tasks to nodes that are free to compute them. The three scenarios that imply using the cloud all obtain similar values, with the cloud-only implementation having the best results. However, the benefit of computing an extra 1216 tasks might not counter-balance the additional 7.2 hours of cloud time, but this is something that should be decided by the developers offering the mobile service, based on its purpose and behavior.

We have seen in Table 5 that the cloud-based scenario sees the most tasks computed during the experiment duration, but Figure 12 shows that this has a very high negative effect on battery consumption, regardless of the device type. For iPhone 5S nodes, the energy consumption is increased by 27% when compared to the case when offloading can also be performed on nodes, while for the Samsung Galaxy S7 nodes 28% more battery is consumed by the nodes. It can also be observed that the battery consumption is approximately the same for all the other three scenarios.

Finally, Figures 13 and 14 show that the distribution of battery consumption per activity is similar to the one specific to the one-community HCMM scenario (as seen in Figures 9 and 10). Namely, close-range communication between devices does not consume a lot of battery, while most of the energy consumption is caused by task computations on the device.

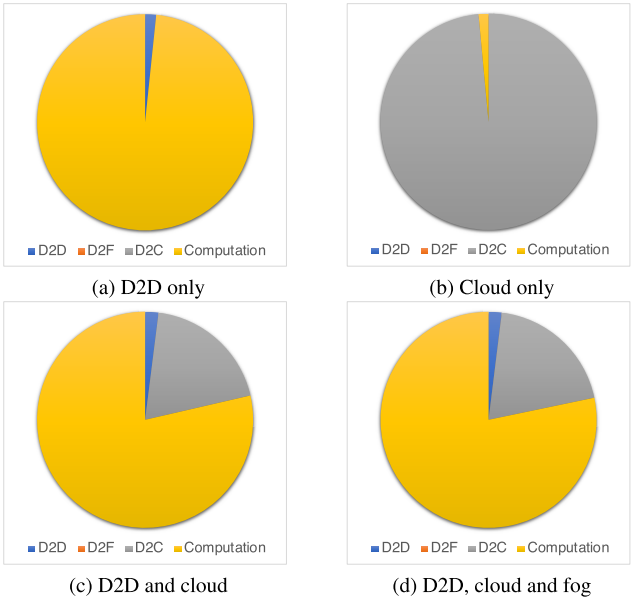


**FIGURE 13. Distribution of battery consumption per activity for the UPB 2015 scenario (iPhone 5S devices).**

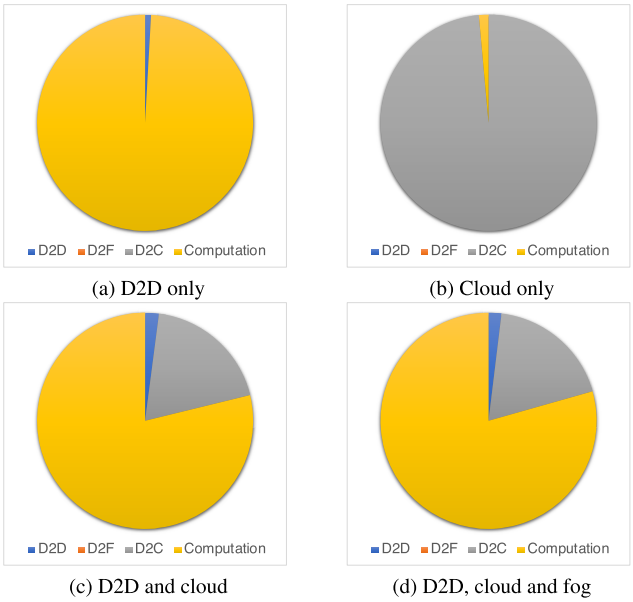


**FIGURE 14. Distribution of battery consumption per activity for the UPB 2015 scenario (Samsung Galaxy S7 devices).**

### 4) COMPARISON

We also compare the proposed Drop Computing offloading solution to mCloud [12] and present and analyze the results in this section. We chose mCloud because it takes into account all the types of offloading supported by Drop Computing and can thus better highlight the advantages brought by our solution. For this reason, we implemented mCloud in MobEmu based on its description in [12]. In short, mCloud uses a cost model in order to be able to decide, at any step, which is the most suitable interface for offloading a task (or if the task should be computed locally). The general cost model is a weighted function of the task execution time and the energy

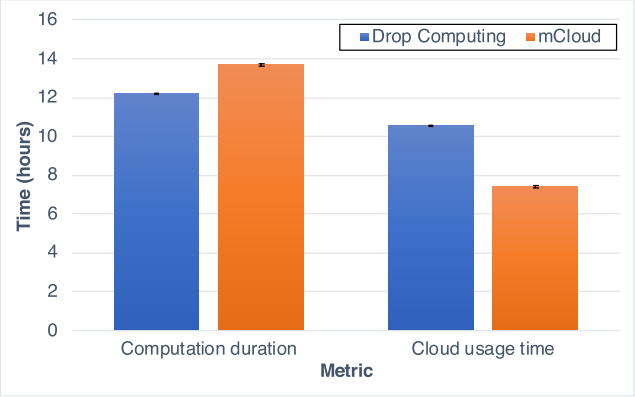


**FIGURE 15.** Computation duration and cloud usage time comparison (UPB 2015).

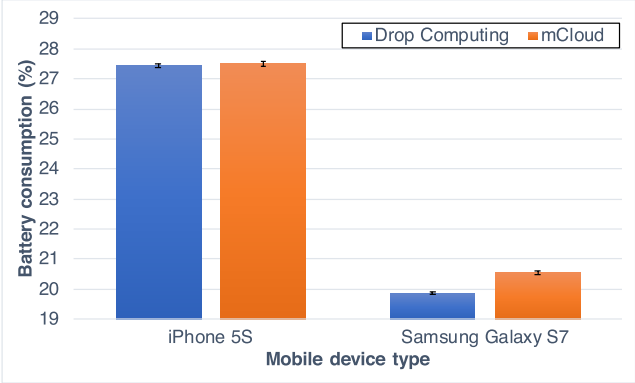


**FIGURE 16.** Average battery consumption per mobile device type comparison (UPB 2015).

consumption, with particular forms of the function depending on the interface being analyzed. When multiple interfaces are available at the same time, TOPSIS [34] (Technique for Order of Preference by Similarity to Ideal Solution) is applied to select the channel that the task will be offloaded on. The weights that we employ for the cost function in our analysis are the ones proposed in [12]. Furthermore, as presented in Section III-B, we assume that nodes communicate with the cloud using 4G, with the fog devices using Wi-Fi, and with each other using Bluetooth.

The computation duration and cloud usage time of Drop Computing and mCloud on the UPB 2015 trace (on the same scenario described in Section IV-A) are presented in Figure 15. It can be observed that the computation duration is 1.5 hours lower when employing Drop Computing, which amounts to an improvement of 10.9%. However, mCloud is able to reduce the overall cloud usage time with more than 3 hours (i.e., a decrease of about 30%). This happens because, based on the cost function proposed in citeZhou2016, mCloud tends to favor offloading to fog or mobile devices, as opposed to the cloud. However, the results that we obtained also show that, while 32170 tasks are solved during a Drop Computing run, mCloud only manages 29929, which is a 7% decrease. Furthermore, we also show the average battery consumption per mobile device type in Figure 16 and it can be observed that less battery is used by Drop Computing on both iPhone and Samsung nodes (with reductions of 0.3% and 3.3%, respectively).

### C. DISCUSSION AND RECOMMENDATIONS

The results presented in this section show that the Drop Computing paradigm can be employed to successfully improve various metrics in a cloud-backed mobile network, ranging from task computation speed to costs or battery consumption. By intelligently offloading to neighboring mobile devices, fog nodes, or the cloud, overall communication parameters can be improved. Depending on the type of mobile network that the multi-layered Drop Computing architecture is deployed in, various metrics can be emphasized as opposed to others. For example, if the application-level requirement is that the task computation duration should be low, then the cloud can be employed more often, leading to lower execution times but to a higher usage of the cloud backend. This in turn leads to higher costs for the application developers or maintainers, which is why these metrics should be balanced properly. On the other hand, if the desire is that battery consumption is reduced as much as possible without particular regard to computation speed, then close-range offloading is preferred due to a lower energy consumption. New technologies such as LoRa, NB-IoT or 5G will only improve Drop Computing, because they can easily slot into one or multiple layers (e.g., LoRa and NB-IoT are useful at the bottom levels, whereas 5G will be able to participate in every layer).

These are all factors that generally need to be balanced by the developers of applications on top of Drop Computing, but in the future we envision a dynamic method of weighting these metrics based on the shape and behavior of the network at each step, through automatic adaptation. However, this is not a trivial task, since there are a lot of types of applications that can benefit from Drop Computing.

As shown in [35], both mobile and Internet of Things (IoT) applications can easily be deployed on top of Drop Computing. On the mobile side, applications that are used in crowded areas (like stadiums or concert halls) can greatly gain from using Drop Computing, because it can take the load from Wi-Fi access points or broadband cell towers and spread it across nearby mobile devices at the bottom layer of the Drop Computing architecture. Since such an environment has a high concentration of nodes, the obtained latencies are able to remain low. Furthermore, any kind of mobile application that has large computations and can benefit from having them split into smaller tasks is suitable for Drop Computing, since our proposed solution implements the entire offloading logic. Finally, in disaster situations where the fixed infrastructure cannot be utilized, the opportunistic layer of Drop Computing would be able to continue on ferrying data from one node to another. When talking about IoT applications, an interesting use case would be an ambient assisted living (AAL) facility, where the patients and the staff have various kinds of body sensors (even from their smartphones), which can

interact with other sensors located in the facility (such as cameras, motion sensors, thermometers, etc.), thus reducing the need for connecting everything to a single infrastructure and instead using Drop Computing to perform sensor-to-sensor communication towards a server or a gateway.

## V. CONCLUSION AND FUTURE WORK

In this paper, we tackled the problem of mobile data offloading in Drop Computing, by proposing solutions for moving data and computations from a mobile device to the cloud, to fog nodes, or to other mobile devices. Through simulations on real-life traces and synthetic scenarios, we showed that the Drop Computing paradigm, which assumes a crowd computing layer below the fog nodes, is suitable for restricted mobile networks. More specifically, we proved that our solution can decrease total computation time (translated to user QoE), cloud usage (i.e., the cost incurred by the service providers or application developers), and battery consumption (which is a very important metric in mobile networks). We also discussed the potential of Drop Computing in various kinds of scenarios, analyzing its suitability for a variety of use cases.

We are currently working on an implementation of Drop Computing on Android and iPhone, and in the future we plan to perform similar experiments on actual mobile devices. Until then, we also want to improve the behavior of our solutions, especially when adding fog nodes.

## REFERENCES

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

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, Sep. 2015.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, Aug. 2012, pp. 13–16.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[5] P. G. Lopez, A. Montresor, D. H. J. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Oct. 2015.

[6] H. Ma, D. Zhao, and P. Yuan, "Opportunities in mobile crowd sensing," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 29–35, Aug. 2014.

[7] R.-I. Ciobanu, C. Negru, F. Pop, C. Dobre, C. X. Mavromoustakis, and G. Mastorakis, "Drop computing: Ad-hoc dynamic collaborative computing," *Future Gener. Comput. Syst.*, vol. 92, p. 889, Mar. 2019.

[8] A. Aijaz, H. Aghvami, and M. Amani, "A survey on mobile data offloading: Technical and business perspectives," *IEEE Wireless Commun.*, vol. 20, no. 2, pp. 104–112, Apr. 2013.

[9] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile data offloading: How much can wifi deliver," in *Proc. 6th Int. Conf.*, Dec. 2010, p. 26.

[10] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proc. 1st ACM Workshop Mobile Cloud Comput. Services, Social Netw. Beyond (MCS)*, New York, NY, USA, Jun. 2010, p. 6. doi: 10.1145/1810931.1810937.

[11] E. Miluzzo and R. Cáceres, and Y.-F. Chen, "Vision: mClouds—Computing on clouds of mobile devices," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Services*, New York, NY, USA, Jun. 2012, pp. 9–14. doi: 10.1145/2307849.2307854.

[12] B. Zhou, A. V. Dastjerdi, R. Calheiros, S. N. Srirama, and R. Buyya, "mCloud: A context-aware offloading framework for heterogeneous mobile cloud," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 797–810, Sep./Oct. 2017.

[13] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Services*, New York, NY, USA, Jun. 2012, pp. 29–36. doi: 10.1145/2307849.2307858.

[14] R.-I. Ciobanu and C. Dobre, "Mobile interactions and computation offloading in drop computing," in *Proc. Int. Conf. Netw. Based Inf. Syst.* Cham, Switzerland: Springer, 2018, pp. 361–373.

[15] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks," *IEEE Commun. Mag.*, vol. 44, no. 11, pp. 134–141, Nov. 2006.

[16] P. Hui, J. Crowcroft, and E. Yoneki, "BUBBLE Rap: Social-based forwarding in delay-tolerant networks," in *Proc. 9th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.* New York, USA, 2008, pp. 241–250. doi: 10.1145/1374618.1374652.

[17] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 7, no. 3, pp. 19–20, Jul. 2003. doi: 10.1145/961268.961272.

[18] R.-I. Ciobanu, R.-C. Marin, C. Dobre, V. Cristea, and C. X. Mavromoustakis, "ONSIDE: Socially-aware and interest-based dissemination in opportunistic networks," in *Proc. Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–6.

[19] A. Socievole, E. Yoneki, F. De Rango, and J. Crowcroft, "Opportunistic message routing using multi-layer social networks," in *Proc. 2nd ACM Workshop High Perform. Mobile Opportunistic Syst. (HP-MOSys)*, New York, NY, USA, Nov. 2013, pp. 39–46. doi: 10.1145/2507908.2507923.

[20] P. Hui and J. Crowcroft, "Predictability of human mobility and its impact on forwarding," in *Proc. 3rd Int. Conf. Commun. Netw. China*, Hangzhou, China, Aug. 2008, pp. 543–547.

[21] W. Moreira, P. Mendes, and S. Sargento, "Opportunistic routing based on daily routines," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimediaa Netw. (WoWMoM)*, Jun. 2012, pp. 1–6.

[22] R. I. Ciobanu, C. Dobre, and V. Cristea, "SPRINT: Social prediction-based opportunistic routing," in *Proc. IEEE 14th Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2013, pp. 1–7.

[23] C. Boldrini and A. Passarella, "HCMM: Modelling spatial and temporal properties of human mobility driven by users' social relationships," *Comput. Commun.*, vol. 33, no. 9, pp. 1056–1074, Jun. 2010.

[24] C. Boldrini, M. Conti, and A. Passarella, "Design and performance evaluation of contentplace, a social-aware data dissemination system for opportunistic networks," *Comput. Netw.*, vol. 54, no. 4, pp. 589–604, Mar. 2010.

[25] A. Mei, G. Morabito, P. Santi, and J. Stefa, "Social-aware stateless forwarding in pocket switched networks," in *Proc. INFOCOM*, Apr. 2011, pp. 251–255.

[26] R.-I. Ciobanu, R.-C. Marin, and C. Dobre, *MobEmu: A Framework to Support Decentralized Ad-Hoc Networking*. Cham, Switzerland: Springer, 2018, pp. 87–119. doi: 10.1007/978-3-319-73767-6_6.

[27] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," Dept. Comput. Sci., Duke Univ., Durham, NC, USA, Tech. Rep. CS-200006, Apr. 2000. [Online]. Available: http://issg.cs.duke.edu/epidemic/epidemic.pdf

[28] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: An efficient routing scheme for intermittently connected mobile networks," in *Proc. ACM SIGCOMM Workshop Delay-Tolerant Netw.*, New York, NY, USA, Aug. 2005, pp. 252–259. [Online]. doi: 10.1145/1080139.1080143.

[29] R. Ciobanu, R.-C. Marin, C. Dobre, V. Cristea, C. X. Mavromoustakis, and G. Mastorakis, "Opportunistic dissemination using context-based data aggregation over interest spaces," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 1219–1225. doi: 10.1109/ICC.2015.7248489.

[30] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE simulator for DTN protocol evaluation," in *Proc. 2nd Int. Conf. Simulation Tools Techn.*, Brussels, Belgium, Mar. 2009, p. 55. doi: 10.4108/ICST.SIMUTOOLS2009.5674.

[31] R.-C. Marin, C. Dobre, and F. Xhafa, "Exploring predictability in mobile interaction," in *Proc. 3rd Int. Conf. Emerg. Intell. Data Web Technol.*, Sep. 2012, pp. 133–139.

[32] D. J. Watts, *Small Worlds The Dynamics of Networks Between Order and Randomness*, vol. 9. Princeton, NJ, USA: Princeton Univ. Press, 2004.

[33] V. Rao, G. Singhal, A. Kumar, and N. Navet, "Battery model for embedded systems," in *Proc. 18th Int. Conf. VLSI Design Held Jointly 4th Int. Conf. Embedded Syst. Design*, Jan. 2005, pp. 105–110.
[34] Y.-J. Lai, T.-Y. Liu, and C.-L. Hwang, "TOPSIS for MODM," *Eur. J. Oper. Res.*, vol. 76, no. 3, pp. 486–500, Aug. 1994.
[35] R.-I. Ciobanu, V.-C. Tăbuşcă, C. Dobre, L. Băjenaru, C. X. Mavromoustakis, and G. Mastorakis, "Avoiding data corruption in drop computing mobile networks," *IEEE Access*, vol. 7, pp. 31170–31185, 2019.

**RADU-IOAN CIOBANU** received the B.S., M.S., and Ph.D. *(summa cum laude)* degrees from the Faculty of Automatic Control and Computers, University Politehnica of Bucharest, in 2010, 2012, and 2016, respectively.

He has worked in the area of mobile devices for more than eight years, having experience in both startups (VirtualMetrix) as well as corporations (Luxoft). He is currently a Lecturer and a Researcher with the Computer Science Department, Faculty of Automatic Control and Computers, University Politehnica of Bucharest. His research interests include pervasive and mobile networks, DTNs, opportunistic networks, and cloud computing. His research has led to the publishing of numerous papers and articles at important scientific journals, such as *Pervasive and Mobile Computing*, the *Journal of Network and Computer Applications*, *Transactions on Emerging Telecommunications Technologies*, and *Ad Hoc Networks* and conferences, such as the IEEE GLOBECOM, ICC, IM, and WoWMoM. During his still young career, he has been involved in several national and international research projects in the area of mobile and cloud computing, and the IoT and ambient-assisted living.

**CIPRIAN DOBRE** received the Ph.D. and Habilitation degrees. He currently leads the MobyLab Laboratory on Pervasive Products and Services, University Politehnica of Bucharest. He is also a Professor. He has scientific and scholarly contributions on data science, mobile and ubiquitous computing, mobile and urban smart technologies, the Internet of Things, monitoring, wireless networks, and modeling/simulation. He received a Ph.D. Scholarship from the California Institute of Technology (Caltech), USA, and another one from Oracle. He received the Award Gheorghe Cartianu of the Romanian Science Academy and the IBM Faculty Award. His results received two CENIC Awards and five Best Paper Awards and were published in articles in major international peer-reviewed journal and well-established international conferences and workshops. He currently coordinates the project Clinically validated INtegrated Support for Assistive Care and Lifestyle Improvement: the Human Link (vINCI, AAL2017-63-vINCI).

**MIHAELA BĂLĂNESCU** received the M.D. degree in applied statistics and optimization method from the University of Bucharest, Romania, and the Ph.D. degree in engineering from the University Politehnica of Bucharest, Romania, in 2005. She is currently a Senior Researcher with Beia Consult International over 20 years of experience in the field of data analysis, environmental impact assessment, technological process modeling, climate change, and environmental risk management. She is a Mathematician. She has authored three books, has published over 25 articles in ISI and BDI journals, 30 in proceedings and conference volumes, and over ten technical review reports for United Nation Framework Convention on Climate Change (UNFCCC), and holds one patent. Also, she was the Project Manager and a Team Member of nine national grants and five international projects within the last five years. Besides the research activities, she provides support for national authorities (Ministry of Environment and National Environmental Protection Agency) for the reporting under various EU legal requirements and international agreements and offers technical consultancy for industrial units.

**GEORGE SUCIU, JR.** received the M.Sc. degree and Ph.D. degrees in cloud communications from the Faculty of Electronics, Telecommunications and Information Technology, University Politehnica of Bucharest, and the M.B.A. degree in informatics project management and IPR from the Faculty of Cybernetics, Statistics and Economic Informatics, Academy of Economic Studies Bucharest. He has been an ICT Solutions Manager, since 1998. He has also been an Research and Development and Innovation Manager with BEIA Consult International, since 2008. He is currently a Senior Scientific Researcher Third Degree, with more than 15 years of experience in Research and Development (R&D) projects. His postdoctoral research work is focused on the field of cloud communications, blockchain, big data, and the IoT/M2M. He has experience as a Coordinator and a WP Leader for over 30 Research and Development projects, such as FP7, H2020, and Eureka/Eurostars and is involved currently in over ten international and five national projects. He has authored or coauthored over 150 journal articles and scientific papers presented at various international conferences and holds over five patents.