IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 13, 2019, accepted February 25, 2019, date of publication March 11, 2019, date of current version March 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2903018

Avoiding Data Corruption in Drop Computing Mobile Networks

RADU-IOAN CIOBANU[1], VLĂDUȚ-CONSTANTIN TĂBUȘCĂ[1], CIPRIAN DOBRE[2], (Member, IEEE), LIDIA BĂJENARU[2], (Member, IEEE), CONSTANDINOS X. MAVROMOUSTAKIS[3], (Senior Member, IEEE), AND GEORGE MASTORAKIS[4], (Member, IEEE)

[1]Faculty of Automatic Control and Computers, University Politehnica of Bucharest, 060042 Bucharest, Romania
[2]National Institute for Research and Development in Informatics, 11456 Bucharest, Romania
[3]Department of Computer Science, University of Nicosia, 2417 Nicosia, Cyprus
[4]Department of Informatics Engineering, Technological Educational Institute of Crete, 714 10 Heraklion, Greece

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

This work was supported in part by projects vINCI: Clinically-validated INtegrated Support for Assistive Care and Lifestyle Improvement: the Human Link under Grant AAL2017-63-vINCI and RO-SmartAgeing: Non-invasive monitoring and health assessment of the elderly in a smart environment under Grant PN 19 37 03 01 and Grant CPN 301 100/2019.

ABSTRACT Drop computing is a network paradigm that aims to address the issues of the mobile cloud computing model, which has started to show limitations especially since the advent of the Internet of Things and the increase in the number of connected devices. In drop computing, nodes are able to offload data and computations to the cloud, to edge devices, or to the social-based opportunistic network composed of other nodes located nearby. In this paper, we focus on the lowest layer of drop computing, where mobile nodes offload tasks and data to and from each other through close-range protocols, based on their social connections. In such a scenario, where the data can circulate in the mobile network on multiple paths (and through multiple other devices), consistency issues may appear due to data corruption or malicious intent. Since there is no central entity that can control the way information is spread and its correctness, alternative methods need to be employed. In this paper, we propose several mechanisms for ensuring data consistency in drop computing, ranging from a rating system to careful analysis of the data received. Through thorough experimentation, we show that our proposed solution is able to maximize the amount of correct (i.e., uncorrupted) data exchanged in the network, with percentages as high as 100%.

INDEX TERMS Mobile, cloud, edge, opportunistic, consistency.

I. INTRODUCTION

Mobile applications nowadays offer a large amount of innovative features for end users. However, although smartphones generally have a high computing power and plenty of resources, these applications generally rely on cloud support in order to offer the best interaction for the user, through mobile cloud computing [1]. Because of the numerous cloud interactions, there are certain limitations and challenges regarding the network load [2], since even two devices located close to one another need to pass through the cloud in order to interact.

Communicating with a cloud system is necessary for devices with limited resources, but costly in terms of infrastructure and delay. In order to reduce these interactions, the amount of data sent to cloud platforms needs to be reduced, by moving some processing at the edge of the network, closer to where data are generated. In a network composed of mobile devices (smartphones, sensors, things, etc.), in order to have access to data or computing power, a node needs to make a request to the cloud. To avoid high latencies in the cloud, as well as the cost of virtual resources, edge computing is employed [3]. This paradigm refers to the existence of routers, switches or set-top-boxes in the same network as the mobile devices, which can cache data received from the cloud or even help process some tasks. Thus, when a node requests data or compute-intensive tasks to be solved, it first contacts the edge devices, which can offer the reply without

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

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

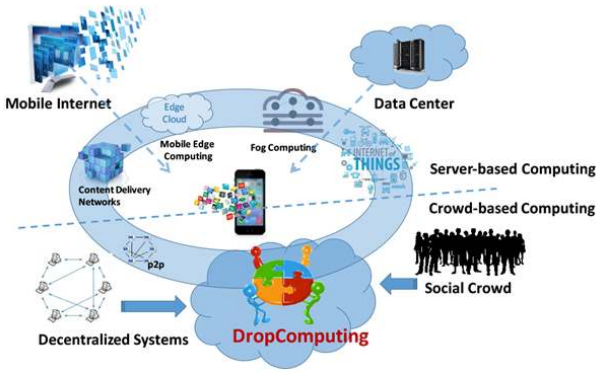


FIGURE 1. The Drop Computing paradigm.

needing to contact the cloud. However, since edge and fog computing have already begun to show some limitations [4], new concepts have become necessary.

By further extending the edge computing model in order to reduce latency and costs even more, we developed the Drop Computing paradigm [5], which adds an extra layer between the mobile devices and the edge nodes, as shown in Figure 1. That layer is composed of neighboring mobile nodes that can be accessed using opportunistic communication in a hop-by-hop probabilistic manner. Thus, if a mobile node needs data or has to compute a task and does not have the necessary resources, it can try to get the results from the ad-hoc cloud of mobile devices existing around it. Using these close-range nodes is cheaper even than contacting the edge, because short-distance protocols such as Bluetooth and Wi-Fi Direct tend to consume less power than mobile broadband protocols, while at the same time having lower latencies due to the short distances. In order to optimize the process of selecting the suitable mobile node that can help with a request (for data or for computations), social and historic metadata about the nodes making up the mobile layer are employed. Thus, only nodes that are considered familiar and trustworthy are selected for serving the requests of a mobile device.

Drop Computing implies that data or tasks are spread into the network composed of mobile devices, for quicker access and a lower consumption of resources. However, in such situations where data belonging to a node pass through other peers, extra care should be taken to ensure data consistency. Since at the lowest layer of Drop Computing (as shown in Figure 1) we are dealing with a decentralized network (composed of mobile devices that only have information about and from the nodes they encounter), the classical data consistency methods cannot be employed. There is no central entity for ensuring consistency, so nodes need to govern themselves and decide together which data are correct. Some nodes might have hardware failures or might be malicious, so they should be avoided. Therefore, in this paper we propose several solutions for ensuring data consistency for task computation in Drop Computing, while striving to have a low effect on the overall processing latency and the number of tasks computed. Through thorough experimental simulations, we show that our solution is even able to achieve 100% correctness in certain situations, while keeping the effect on latency down.

This paper is an extension of the work published at the IEEE International Conference on Computational Science and Engineering [6]. As an addition to that work, we have delved into more detail regarding the Drop Computing paradigm, presenting the way it performs and proposing four scenarios where it proves useful. Furthermore, we extended the experimental validation of our solution by testing thoroughly on an additional mobility trace (collected in conditions that were different than the original trace and the synthetic mobility model), while also analyzing the benefit of our solution on various mobile network-specific metrics.

The remainder of this paper is structured as follows. In Section II, we present several solutions similar to Drop Computing, together with methods for ensuring data consistency in mobile cloud networks. Then, in Section III, we discuss the main components of Drop Computing and the way they work, and then we propose four scenarios (three for mobile applications and one for an IoT-based elderly care facility) that highlight the benefits and usability of our proposed solution, along with the need for data consistency. We propose our solution in Section IV and evaluate it in Section V. Finally, we present our conclusions and future work in Section VI.

II. RELATED WORK

While multiple solutions similar to Drop Computing have been proposed recently, none of them have mechanisms for data consistency, while also not allowing for device-to-device communication along multiple hops.

Huerta-Canepa and Lee [7] present a solution where devices with common goals work together to solve tasks. Each node is able to compute parts of a task, and then all these partial results are merged into the final solution, which is shared by all the contributing nodes. The main issue of this solution is that it does not account for node mobility when offloading tasks, which is an important component of the Drop Computing vision that we are addressing in this paper. Moreover, Drop Computing supports a heterogeneous network, where devices come in all shapes and sizes, whereas Huerta-Canepa and Lee's framework assumes that all nodes are similar. Fernando et al. propose a similar mobile cloud framework [8], but this solution has the drawback of only allowing single-hop device-to-device communication using Bluetooth. Furthermore, users of this framework are incentivized through monetary transactions, which might prove difficult to implement in real-life, especially since no data corruption mechanisms are proposed and communication is performed in a decentralized fashion between mobile devices.

The mCloud platform [9] allows task offloading from mobile node to mobile node, but is also able to employ the cloud as backup. However, device-to-device communication is also performed over a single hop, while incentives for participation are offered by carriers, which might not be

a realistic solution. Moreover, like the previous two solutions, the mCloud platform does not propose mechanisms for ensuring data consistency. Another platform with the same name [10] proposes task offloading from mobile devices towards edge devices or public cloud services. It allows communication through several wireless channels such as Wi-Fi, 3G, Bluetooth or Wi-Fi Direct, employing a multi-criteria optimization solution for the offloading behavior, which takes into consideration battery consumption, computation time, resource availability and network conditions. However, similarly to the other solutions mentioned here, it only allows one-hop device-to-device communication, while not offering data consistency mechanisms to prevent against corruption.

Another solution implies using cloudlets formed of nodes located in the vicinity of a mobile user [11]. The composing nodes of a cloudlet can work together to help each other solve tasks by offloading from one to another. The framework considers two types of cloudlets, elastic (specially built in datacenters for offloading) and ad-hoc (formed on the spot when multiple devices are connected to the same network). The devices in a cloudlet are in the same local network, whereas devices in different cloudlets must be connected to the global Internet in order to communicate. Thus, if a node needs to offload some computations to a different cloudlet, it needs to contact a service that is aware of all the cloudlets. From our standpoint, this represents a disadvantage, because this service is a central point of failure and congestion. Another disadvantage of the cloudlet framework when compared to Drop Computing is that it does not work optimally in scenarios with mobile devices because, in order to use the resources of a cloudlet, a node must be connected to an access point, so nodes that are not connected to a Wi-Fi access point or to a mobile cell tower will not be able to offload their computations, even if some capable devices are in wireless range. Drop Computing is able to solve this issue, because it uses opportunistic communication for offloading tasks.

In cloud computing, popular communication protocols generally make the assumption that nodes which make requests are always connected to the sources of information. However, since we assume a scenario with mobile devices, this does not always hold true, so new means of communication have to be devised, that take into consideration critical scenarios where the infrastructures get damaged due to causes such as natural disasters. In these situations, availability-ensuring methods like data replication should be employed, because data spread in multiple geographic locations can reduce latencies and avoid congestion.

Data consistency is an important issue in mobile networks, because the high degree of node mobility can easily lead to partitioning. Hara and Madria
citeHara2009 show that the hit rate can be increased and the congestion reduced if a local consistency mechanism is used instead of taking a global consistency approach. However, this solution is not generally feasible in current mobile networks (and in Drop Computing in particular), because devices might not always be connected to one another, so paths between two or more nodes that want to communicate do not always exist.

In the literature, several data replication methods have been proposed over the years, but they do not necessarily apply to the scenarios that Drop Computing is aimed at. For example, One-to-One Optimization (OTOO) [13] assumes that mobile nodes collaborate with the peers they are in contact with at a given moment and decide what information to store. In this case, each node individually computes a utility value as the frequency of access to a data item, and then stores the item based on the computed value. However, there is a certain limitation to this method, given by the fact that nodes can only communicate with one-hop neighbors, thus restricting their communication range. In Drop Computing, nodes communicate opportunistically, so this problem is averted by allowing peer collaboration across Wi-Fi or Bluetooth range boundaries.

III. DROP COMPUTING

In this section, we present the Drop Computing paradigm that is at the basis of this paper. We begin by describing its evolution from mobile networks where communication is performed through close-range protocols in a probabilistic fashion, and then we propose some use cases where Drop Computing can be employed for mobile applications and in an IoT scenario. Finally, we discuss about data consistency in Drop Computing.

A. THE EVOLUTION OF DROP COMPUTING

Opportunistic networks (ONs) have been proposed as an evolution of mobile ad-hoc networks (MANETs), being part of the delay-tolerant networking (DTN) paradigm [14]. ON nodes are characterized by a high degree of mobility, leading to dynamic interactions between the members of the network. Because of this mobility of nodes, communication between two peers can (and should) occur even if they are not directly connected. Instead, other nodes are probabilistically employed as next hops, based on a series of heuristics that use context information. The main building block of ONs comes from a paradigm entitled store-carry-and-forward [15], where nodes store the data to be sent for a period of time, carry it around the network, and then forward it to a suitable next hop, or to the destination.

The other component of Drop Computing, mobile edge computing, scales communication horizontally [16], assuming that the cloud model is not feasible anymore. Thus, requests for data and computation are not made directly to cloud, but instead they are forwarded to the edge of the network, to specialized small-scale devices that can offer an extremely distributed computing environment that is employed for developing applications and services. Furthermore, the edge devices can store and process data closer to the requesting nodes, reducing the congestion and communication latency. Another advantage brought forth by mobile edge computing is that applications can be split into multiple components which can be spread in the network, so some

requests can be answered by the edge nodes, while others can be resolved directly by the cloud service.

Based on opportunistic and mobile edge networking (with a cloud backbone), Drop Computing offers decentralized computing over multi-level networks, combining cloud and wireless technologies over a social crowd composed of mobile and edge devices [5]. This way, Drop Computing mobile nodes can take advantage of other devices in proximity for a quicker and more efficient access to data and computations. The need for this paradigm comes from the insufficiency of the classic cloud model in the era of the Internet of Things, where tens of thousands of small devices perform cloud requests simultaneously. By employing opportunistic communication before the edge layer, Drop Computing can extend the network and the cloud horizontally.

B. MOBILE USE CASES

Drop Computing can have a variety of use cases in the real world. While initially it was conceived with smartphones as the nodes, we propose extending it for the Internet of Things, which can be easily done, because the working principles are the same. Instead of only having smartphones in the network, they are interspread with sensors and actuators, and the communication patterns remain the same. The smartphones will be the nodes that can aid others with data and computation (for example, pre-processing or aggregating data from some sensors), acting as the extra layer before the edge nodes. Thus, in this section we present three use cases that are oriented more towards smartphones, while in the next section we present in detail another scenario showing how this can be taken further with other types of nodes from the Internet of Things.

As a first use case, we address the situation where, in some crowded areas such as stadiums or concert venues (where the concentration of mobile devices in a small area is large), the mobile broadband or wireless connection will have high latencies and low speeds. This happens because the broadband cell or the Wi-Fi access point are crowded, since many requests are made at the same time. However, based on the assumption that, in such a human-centric scenario, the data requests will be related (for example, participants at a sporting event might be interested in other scores from the matches being played in parallel, or in information about the sport they are watching), Drop Computing might alleviate the connectivity problems. Thus, instead of having all the nodes connect to the cell tower or access point and make requests at the same time, only a (dynamically changing) subset of nodes does this, and then the information is spread in an opportunistic fashion around the venue. The selection of devices that will actually connect to the access point and make the requests is performed using social metrics such as popularity (of the data or of the node) or number of links, since the number of connections needs to be minimized, while the number of nodes that receive the data of interest in a timely fashion should be maximized.

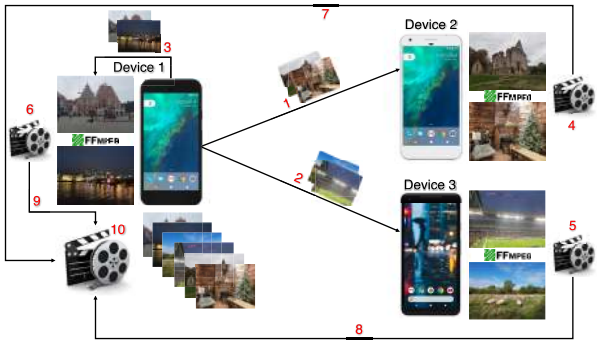


FIGURE 2. Example of a photo collage application over Drop Computing.

Another scenario is represented by CPU-intensive mobile apps, where generally a cloud backend is employed in order to perform the heavy computations. As a first side effect of this, such an application cannot work without an Internet connection, thus adding extra limitations to its functionality. With Drop Computing, the mobile device would be able to leverage the aid of nodes nearby to perform the heavy computations, which, in time, is able to lead to an overall decrease in battery consumption and delay. As a concrete example, we envision an application that allows mobile users to create a video collage from a list of photos, as shown in Figure 2. Generally, similar apps (such as Google Photos, which has this feature), employ the cloud for collating the photos, but we want to address situations where the cloud is not available or the user wants to avoid employing it. Since a device might not always have the capabilities of performing certain computations itself, the application has an offloading component based on Drop Computing. Thus, as shown in the example in Figure 2, the pictures to collate are spread between the nearby devices, then each device creates a collage of its own subset of photos and then sends it back to the originating device, which merges the collages it receives. The main goal of such a scenario (and of the Drop Computing paradigm) is to decrease the load on a single device in a collaborative fashion, in order to achieve fairness and efficiency at the mobile network level. The two scenarios presented so far show that Drop Computing can be used for both data offloading (as in the first use case) and computation offloading (as seen in the collage app scenario).

One of the most important real-life situations where using the opportunistic mobile nodes at the base of a Drop Computing network brings an advantage is in the case of natural disasters. In many such crisis situations (that may lead to large-scale physical damages), the communication between members of the rescue teams and with the victims is crucial. With the help of Drop Computing, nodes would be able to communicate even without an existing infrastructure, since the devices close to each other would be able to exchange information. Thus, if for example a person is caught in a wreckage but has their smartphone on them, then they would be able to notify the rescue teams in a device-to-device fashion (either when in range of other mobile devices, or even

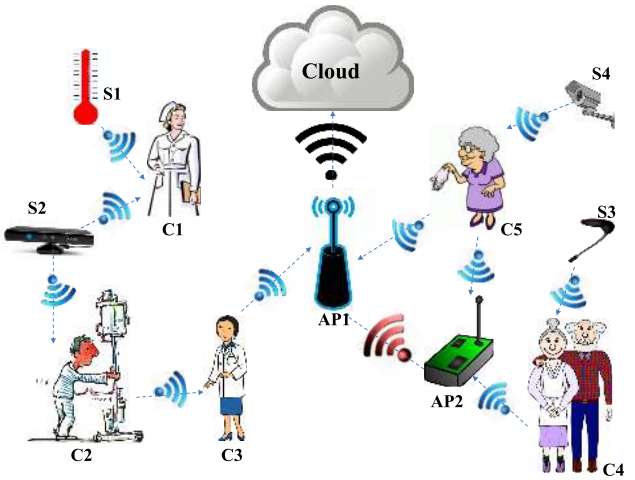


FIGURE 3. AAL IoT scenario for Drop Computing.

with the help of drones that fly around the disaster area looking for devices using close-range protocols such as Wi-Fi Direct or Bluetooth).

C. IoT SCENARIO

As an application for Drop Computing in the Internet of Things, we propose the scenario presented in Figure 3. We assume that there is an elderly home facility that uses ambient assisted living (AAL) for improving the lives of the residents through careful monitoring and analysis of their behavior, living conditions, and state. The analysis is performed on a local server or in a cloud system, since it requires aggregating information from various sensors and corellating it.

As shown in Figure 3, there are four types of entities present in our scenario: sensors (marked with an *S*), carriers (*C*), access points (*AP*), and the cloud. In the following sections, we present each of these entities in detail, together with their role and behavior.

1) SENSORS

Various types of sensors are employed for monitoring. They can be embedded into the residents' environments, resulting in intelligent living environments capable of enhancing daily life, especially in the case of elderly or individuals suffering from mental or motor deficiencies. In particular, wireless mesh sensor networks (WMSNs) could be used for designing unobtrusive, interconnected, adaptable, dynamic, and intelligent environments where sensors are embedded in everyday objects [17]. The sensors embedded into daily environments are usually called ''ambient sensors'' (as opposed to body sensors). The ambient sensors will collect various types of data to deduce the activities of inhabitants and to anticipate their needs in order to maximize their comfort and quality of life [18].

WMSNs are based on mesh and opportunistic networking, where each node also serves as a relay for other nodes, aside from capturing and disseminating its own data. The main benefit of WMSNs is their capability to dynamically self-organize and self-configure, with the network automatically establishing and maintaining mesh connectivity among sensors [19]. WMSNs do not require centralized APs to mediate the wireless communication, and they are particularly suitable to be used in complex and dynamic environments such as living spaces [20].

In our scenario, we present in Figure 3 (marked with *S*) several external sensors that may be found in the facility, such as temperature sensors (*S1*), motion sensors (*S2*), microphones (*S3*), or cameras (*S4*). These are fixed sensors that are able to collect raw data and send them further. In our scenario, the sensors themselves do not have long-range connection capabilities, so they cannot connect to the cloud or to the server themselves. Instead, they use close-range protocols such as Bluetooth, Wi-Fi Direct or ZigBee to send their data opportunistically to any mobile carriers that come within range.

2) CARRIERS

Since sensors cannot directly connect to the cloud or to the local servers hosting the facility's services, their data are sent opportunistically (through close-range protocols) to mobile nodes which are carried by the residents or by the nurses and doctors, as shown by the entities marked with *C* in Figure 3. The mobile nodes receive the data collected by the sensors when they are in close range, and then, through mobility, move them further towards the access points that have the possibility of connecting to the server or to the cloud.

The communication between the sensors and mobile devices is thus performed through Drop Computing, where nodes, in the form of mobile devices, can collaborate without the need of supervision or coordination from a central entity. Each node can communicate with any other node found in the proximity defined by the Wi-Fi/Bluetooth range. The lack of a central entity forces nodes to form their own opinions about the network, only by gathering information from other nodes (mostly in a gossiping manner). The degree of accuracy of this opinion is most of the time vital to the behavior of the algorithms. Each node has to take complex routing decisions each time it receives a messages, so, for this reason, it should keep itself as informed as possible.

The nodes belonging to the residents do not only have Drop Computing communication capabilities, so their goal is not only to collect data from the sensors and move them towards the cloud; they also have sensors themselves, potentially even being part of a body area network (BAN) [21]. In a BAN, various sensors are attached on clothing or on the body or even implanted under the skin [22]. An important benefit of BANs is their scalability and integration with other network infrastructures. BANs may interface with wireless sensor networks (WSNs), RFID, Bluetooth, Bluetooth Low Energy (BLE), video surveillance systems, wireless personal area networks (WPANs), wireless local area networks (WLANs), the Internet, and cellular networks [23].

3) ACCESS POINTS AND THE CLOUD

In our proposed scenario, aside from static nodes (sensors) and mobile nodes (residents, doctors, nurses), there are also two types of access points, denoted by *AP* in Figure 3. Firstly, there are access points that have an Internet connection (*AP1*), which collect data from sensors and mobile devices, and they are able to upload all the information to the processing server. They also have some computing capabilities of their own, which allow them to pre-process the data before sending them to the server (or even to make their own decisions, so not everything is actually uploaded to the server, thus reducing the transferred data and, implicitly, the power consumption).

Moreover, there are also smaller access points (such as *AP2* in Figure 3) that are used to collect data from mobile devices and send them to the main APs. These smaller access points have a much higher range than the sensors and devices carried by the residents, and can thus move the data towards the main access point easier.

The cloud entity, as shown in Figure 3, is where the actual data processing and decisions are made. The server (which is not necessarily in the cloud, but can be running on a local high-performance machine) receives data from the static sensors and from the BANs carried by the residents, through the main access points. Based on these data, notifications can be sent, or actions can be taken (if some special situations, such as abnormal behavior or danger, are encountered). However, if the server is located in the cloud, the administrators of the assisted living facility may not wish to upload sensitive data regarding their residents. This information can only be stored locally, at the main access points (or at specialized computers), and only non-sensitive data would be uploaded to the cloud. This is an advantage brought to the fore by Drop Computing, since sensor nodes and APs can communicate between themselves, without the need for connecting to the Internet for every interaction (which would require additional security measures for dealing with sensitive data).

4) INTERACTIONS

There are several types of interactions between entities presented in Figure 3, and they are described below.

a: SENSOR TO CARRIER

Sensors collect data from their surroundings and store them. Whenever a carrier comes into range of the protocol used by the sensor (Bluetooth, ZigBee, Wi-Fi Direct, etc.), the collected data are sent to the carrier. This is shown in Figure 3 in the interactions between sensors *S1* and *S2* and carrier *C1*, between *S2* and *C2*, *S3* and *C4*, or *S4* and *C5*.

b: CARRIER TO CARRIER

Not all carriers may end up in the range of an access point, so the devices belonging to the carriers (which also act as BANs for the residents of the facility) are also able to communicate with each other. Thus, whenever two carriers are in range, their devices exchange the data collected so far from other sensors or carriers. This way, the probability that the data collected from various sensors end up at an access point is increased. Such an example is shown in Figure 3 in the interaction between carriers *C2* (a patient) and *C3* (a doctor). *C2* collects some data from sensor *S2*, carries them for a time, and then, upon encountering *C3*, sends them further.

c: CARRIER TO ACCESS POINT

Carriers collect and exchange data among themselves, and also from sensors. These data are meant to be sent to the processing server, operation which can only be performed by the access points. For this reason, they need to collect data from the carriers. This can be seen in Figure 3 in the interactions between carriers *C3* and *C5* and access point *AP1*, or between *C4* and *AP2*.

d: ACCESS POINT TO ACCESS POINT

There are multiple types of access points in our scenario. The first category includes powerful access points that have a connection to the Internet (or to the processing server), such as *A1* in Figure 3. However, since these kinds of APs may be expensive, the scenario we propose also includes some smaller access points, that have a higher range than carrier devices, which can communicate directly to the more powerful APs (but not to the processing server itself). Their goal is to act as intermediaries between the Internet-connected APs and carriers that may not end up in their range. For example, Figure 3 shows a situation where carrier *C4* never ends up in range of *AP1*, but is able to send its data to *AP2*, which in turn delivers it to *AP1*.

e: ACCESS POINT TO THE CLOUD

Finally, the main access points send the data collected from the carriers and sensors to the cloud.

D. DATA CONSISTENCY

Inconsistencies can be caused by various reasons, among which we would like to highlight malicious intent (nodes intentionally tamper with the data in order to spread false information in the network) or hardware failures. At the hardware level, Bairavasundaram et al. mention three classes of data corruption: checksum mismatches, identity discrepancies, and parity inconsistencies [24]. Checksum mismatches can be caused by bit-level corruption and torn or misdirected writes, identity discrepancy is caused by lost or misdirected writes, while parity inconsistency occurs when the memory is corrupted, when writes are lost, or when the parity is miscalculated. When transferring data in a network, corruption may occur due to attenuation or signal loss (or degradation), delay spread, or network congestion. Ways to avoid inconsistencies are generally based on setting up trust and reputation mechanisms (for avoiding malicious nodes), replication, etc. In Section IV, we propose and present our solution for achieving data consistency in Drop Computing.

IV. PROPOSED SOLUTION

In this section, we propose and present several mechanisms for ensuring data consistency in Drop Computing. We implemented our solution assuming that Drop Computing is employed for task computation. Thus, at certain times nodes generate tasks, which they can compute themselves (although they might not always have the required resources) or which can be sent to the Drop Computing network composed of other mobile devices nearby. When the latter occurs, nodes need to ensure that, when they receive the task results back, they are correct and have not been modified. We only focus on the device-to-device component of Drop Computing in this paper because this is where data corruption most commonly occurs. At this level, devices have a much higher chance of malfunctioning and corrupting data, or of maliciously modifying it, whereas in the cloud and at the network edge, consistency mechanisms are usually already in place.

A. TASK EXCHANGE

The scenario we are addressing in this paper assumes that Drop Computing is used by mobile nodes for computing various tasks. A Drop Computing task T is defined as follows:

$$T = <ID, t_g, t_e, c, o, e, p> \tag{1}$$

In the definition above, *ID* is the unique identifier of the task, in order to distinguish it in the network (as an example, this parameter can be generated as the hash of the task data). t_g is the timestamp when the task was generated, whereas t_e is the time when the task expires and needs to be executed in the cloud. This parameter depends on the type of task and the duration it would take to compute, as well as its priority from the standpoint of the user. Parameter c represents the number of cycles of task T, and is useful when estimating the computation duration of a task based on the capabilities of a mobile device. The o parameter specifies the owner of task T (i.e., the ID of the device that generated this task and expects its result), while e is the the current executor of the task. Finally, p is the set of paths taken by this task through the Drop Computing network (a path is basically an array of IDs belonging to the nodes that the task has passed through in the mobile network, whether before being executed or afterwards). Tasks are executed by nodes, which are defined as follows:

$$N = <ID, t, cptu> \tag{2}$$

ID is the identifier of the node, t is the list of tasks this node currently has stored (which can belong to itself or to other nodes, and can be executed or not), and *cptu* is the amount of cycles per time unit that this node can compute. By default, Drop Computing nodes compute their own tasks until they are in range of another mobile device. When a contact occurs, the first step is for the two encountering nodes to verify if they are socially connected, which is performed using the function below (where i and j are the two nodes):

$$conn(i,j) = \begin{cases} 1 & \text{if } contacts(i,j) > t_c \text{ and} \\ & contact\_time(i,j) > t_t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The formula above specifies that two nodes are socially connected if the total number of contacts between them exceeds a threshold t_c and if their total contact duration is larger than a threshold t_t. If the result of the function above is 1, then the nodes in contact attempt to solve the following optimization problem:

$$\begin{aligned} &\text{minimize} \left| \sum_{t \in i.t} t.c \times i.cptu - \sum_{t \in j.t} t.c \times j.cptu \right| \\ &\text{subject to } i.t \cap j.t = \emptyset \end{aligned} \tag{4}$$

The main goal of this bound constrained optimization problem is to balance the computational load on the two encountering nodes, since it attempts to minimize the difference between the total computation durations of the two nodes, calculated as the product between the number of cycles per task and the duration of a cycle per node. 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, we take a greedy approach, where the first node selects the task that takes the longest for it to compute, while the second node chooses one or multiple tasks that take a similar amount of time for it to compute, and so on. We chose this solution over more complex algorithms such as Newton or gradient projection methods due to the mobile device requirements and constraints. Namely, we wanted to avoid using too much of a node's CPU and to choose the tasks quickly, so there is more time during the contact to exchange actual data. In the future, we plan on looking at some more efficient solutions already proposed [25]. When a task is completed by a node that is not its owner, the result needs to return to the owner. That is why, when a node finishes a task, it disseminates its result to all encountered nodes. This process can arguably be improved in order to minimize the load on the network, and that is something that we wish to address in future work.

B. DATA CORRUPTION

In Drop Computing networks, we consider that there are two ways that tasks can be corrupted. The first type of corruption occurs right after the correct execution of a task. In this situation, the task is computed correctly and its result is the expected one, but, when the result is saved from RAM to the main memory, it becomes corrupted. The second way a task can be corrupted is when two nodes meet and exchange information about completed tasks. In this case, a node A initially has task T computed successfully (and correctly), but, when it sends the result of task T to a node B, the information ends up being corrupted. Thus, node A will still have a correct version of task T's result (and will be able to

further spread it correctly in the network upon contact with other devices), whereas node B will end up with a corrupted version. The main difference between these two methods is that, in the former version, the corrupted task spreads to all nodes that the executor encounters after the task is computed, whereas the latter version of corruption only alters the task at the node receiving its result. In other words, the first version corrupts tasks at the executor, while the second one corrupts tasks at the receiver.

C. STORING THE PATH OF A TASK

In our proposal, nodes that corrupt the task results are detected using information regarding the routes a task takes through the network, from its original owner to its executor (or executors), and then back to the owner on multiple paths. Since we are dealing with opportunistic networks, a single task may take multiple paths in the network, be executed by more than one peer, and then end up at its owner by taking different routes. Furthermore, since our data corruption solution requires multiples copies of a task result before deciding which version is correct (as will be shown in Section IV-D), there is a high chance that a task will pass through many different Drop Computing nodes. Aside from this, it should be noted that tasks can be exchanged between nodes before or after being computed (and, in some situations, even when they have been partially computed), so this should also be taken into consideration, especially knowing the ways a task can be corrupted, which have been presented in Section IV-B.

In our implementation, each copy of a task stores the path that it takes through the network. The list of paths taken by a task is represented as follows:

$$T.p = \{p_1, p_2, \ldots, p_n\} \tag{5}$$

p_1 to p_n are the currently stored paths of a task, where a path is defined as follows:

$$p_i = \{< t_{i1}, s_{i1}, d_{i1}, c_{i1}, h_{i1} >, \ldots\} \tag{6}$$

As seen above, a path of index i is a list of 5-tuples, which contains the timestamp of a data exchange (t) between a source node and a destination node (s and d). The c field specifies if the task was executed or not when the exchange took place, while h is a hash of the task, used to differentiate between task versions (i.e., in the case the task has been corrupted).

D. THE EXPECTED VERSIONS OF A TASK

In mobile networks, there is no central entity available at all times that the nodes can query in order to find out if a version of a task is corrupted or not. Thus, we implemented a mechanism where, once the result of a task arrives at the task's owner, it will not be sent directly to the application level. Instead, a certain number of versions of the same task are expected, ideally computed by different nodes or routed through different paths. Once the desired amount of task result versions arrive, the most popular one is selected through a quorum and considered as the correct version.

Algorithm 1 Expected Versions Mechanism

1: L_{data} - list of received versions
2: $Task_{id}$ - received task ID
3: T_c - number of corrupted tasks
4: $N_{versions}$ - number of waiting versions
5: $Task_{data}$ - unmodified task data
6:
7: $Majority =$ percent of most frequent data from L_{data}
8: **if** $Majority$ is equal to 50% **then**
9: increase number of waiting versions for $Task_{id}$ by 1
10: **return**
11: **end if**
12:
13: **for all** data D from list L_{data} **do**
14: **if** detect collisions between D and $Task_{data}$ **then**
15: increase T_c by 1
16: **end if**
17: **end for**
18:
19: **if** $Majority > 50\%$ and $2 \times T_c > N_{versions}$ **then**
20: corrupted version is accepted
21: **else**
22: uncorrupted version is accepted
23: **end if**

However, there might be situations where a task is computed by a node A, which corrupts the result and then sends it through the Drop Computing network on multiple paths, and the owner B of the task only receives versions that have been computed (and corrupted) by A, which would lead to an incorrect task result at note B. In order to avoid such a situation, we propose upgrading the expected versions mechanism by specifying that a given percentage of executors of a task need to be different. Furthermore, we add the restriction that a minimum number of final relay nodes per task should be expected. These restrictions can help increase consistency because of the following:

- by accepting task versions executed by different nodes, the chances that the information is not corrupt increase, regardless of the way data are corrupted
- by accepting task versions from different nodes, we allow the task to have a more diverse path from executor to owner, which is useful in the scenario where tasks are corrupted after they are computed.

The proposed solution for enforcing the number of expected versions is shown in Algorithm 1. When the pre-established number of versions is received, the algorithm applies the quorum method and selects the correct version, which is then sent to the application level. If there are multiple instances where the same number of versions is received, the mobile node will wait for another version of this task result, until a majority is formed (lines 8-11). As shown at lines 19-23 of Algorithm 1, if the majority of versions of a task result are corrupted, then the owner of the task will end

up with the corrupted version. In order to avoid this situation, we propose a rating system in the next section, which has the role of detecting and avoiding the nodes that tend to corrupt data.

E. NODE RATING SYSTEM

As specified above, the role of the rating system is to isolate the nodes that tend to corrupt data, either due to external factors, or because they act maliciously on purpose. It is based on historical information collected when a node waits for a certain number of task versions and whenever two nodes exchange information upon a contact.

Because mobile networks are decentralized, each node has its own local rating for all the other encountered devices in the network, which is re-computed at every interaction on the network. Moreover, information about nodes not yet encountered can also be obtained through gossiping at every contact with other devices. The formula for computing the local rating L_{ij} of a node j from the standpoint of a node i is as follows:

$$L_{ij} = \frac{suc(i,j)}{suc(i,j) + unsuc(i,j)} \tag{7}$$

Thus, L_{ij} is the percentage of successful interactions between the two nodes (i.e., the percentage of times node j has helped node i from the total amount of chances to do this). An interaction means that node i has encountered node j and has asked it to help with the execution of a task or with the delivery of a solved task back towards its owner. If node j was willing to help and thus executed the task or further disseminated it into the Drop Computing network, then the interaction is considered successful. However, whether an interaction was successful cannot be decided on the spot (since the two nodes might not be in range for a sufficient time), so the analysis is performed after a period of time, when node i encounters node j again, or when it comes in contact with another node that has encountered node j recently.

If there have been no direct or indirect interactions between i and j (thus, the rating based on the formula above would be 0), our solution employes pre-trusted nodes, which are considered trustworthy by default based on external factors. In our situation, these factors refer to the connections between the node's owners given by online social networks. Thus, if two nodes have not had any interactions in the network but their owners are socially connected, then their local rating for each other is 1. On the other hand, if they are not socially connected through any online social network, the local trust is set to a predefined value.

However, if a mobile node only calculates other node's task computation willingness based on its local rating, it may end up with a skewed view of the network, while incomplete information might lead to wrong decisions (i.e., non-corrupting nodes can be considered malicious). For this reason, nodes actually employ a global rating mechanism using local values from all the other nodes in the network.

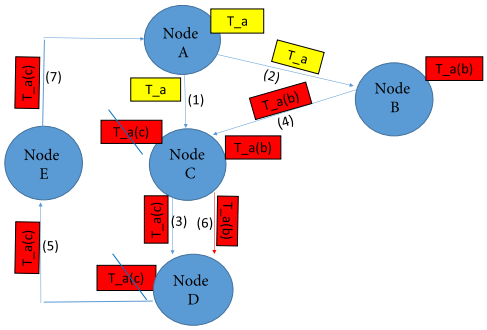


FIGURE 4. Task exchange example.

Thus, for computing the global rating, we calculate a weighted average based on the local rating that each node has for all the other nodes. As such, the global rating value R_{ik} that a node i has for a node k is computed as follows:

$$R_{ik} = \frac{\sum_j s_{ij} L_{jk}}{\sum_j L_{ij}} \tag{8}$$

Once all the expected task versions arrive at the task's owner, it selects the correct version using the quorum mechanism (i.e., the most popular version), and then all the other versions are marked as corrupt. The next step is to analyze all the network exchanges on the paths of the versions that are considered corrupt and see which exchange was responsible for the corrupt version. When this is found, the rating of the sender node at that exchange is decreased, and then the new rating value is gossiped by the task owner at every subsequent contact.

Extra care needs to be taken when deciding which node on the path has corrupted a certain task. For this reason, we add a timestamp to any transaction, in order to create correct paths that can easily be analyzed. Figure 4 shows a node transfer scenario where the timestamp is paramount to making correct decisions. Node A is the owner of task T_a and spreads it in the network for execution, the yellow color of the task showing that it has not been executed yet. At time moment 1, the task is delivered to node C, and at moment 2 to node B. Then, the task is solved by node C (which is what the red color specifies), which then sends it to node D at time moment 3 and deletes it from its own memory. Node B also computes the task and delivers it to node C at time moment 4. Since node C had previously deleted the task at moment 3, it will now receive the version computed by B. This shows that node C sees two versions of the task at two separate moments of time (and, as shown in Figure 4, this is also true for node D). In this situation, if node B corrupts the task and sends a corrupted version to node C, if timestamps are not employed, the owner of the task (node A) will incorrectly assume that node C is the one that corrupted the task, since it had definitely received a correct version from the owner.

The entire node rating mechanism, which is executed whenever a node exchanges tasks with other peers, is presented in Algorithm 2. The algorithm returns *false* if the

Algorithm 2 Node Rating Mechanism

```
1:  dcNodeId - encountered node ID
2:  NMap_R - map between encountered nodes ID and rating values
3:  TMap_R - map between node ID and timer value
4:  T_R - initial timer associated with the rating value
5:  Social - social network of the current node
6:
7:  if NMap_R contains node dcNodeId then
8:      decrease rating timer by 1 and add the new value in TMap_R
9:      R_value = value of the key dcNodeId stored in NMap_R
10:     T_value = timer value from TMap_R
11:
12:     if R_value is between 50 and 100 and T_value <= 0 then
13:         put in NMap_R minimum value between 100 and R_value + 20
14:         put in TMap_R initial timer value T_R
15:     end if
16:
17:     R_value = new value from NMap_R
18:
19:     if R_value >= 75 then
20:         return true
21:     else
22:         return false
23:     end if
24: end if
25:
26: if Social contains dcNodeId then
27:     return true
28: else
29:     return false
30: end if
```

encountered node should not be trusted (i.e., is not socially connected to the current node, or has a rating below 75, as shown at lines 19-23 and 26-30, respectively), and *true* otherwise. Since nodes might corrupt data because of a hardware or software problem and not necessarily out of maliciousness, the proposed algorithm allows such nodes the possibility of increasing their rating through good behavior. This is done using a timer per node, which is decreased every time the rating algorithm is run, as shown on line 8 in Algorithm 2. When the timer expires, if the node has a promising rating (higher than 50), the rating value is increased with 20 and the timer is reset (lines 12-15), thus reincluding the node in the task computation and dissemination process. The node is then considered trustworthy as long as its rating is above 75.

F. HAMMING CODES

Hamming codes are capable of detecting two errors and correcting one and, to implement such a system, parity bits should be added to the data bits. They represent redundant information that contributes to recovering the initial data even when they are corrupted. The overhead of the parity bits reaches small values when the quantity of data grows. If the Hamming codes are not feasible for only 8 bits of data (generating an overhead of 50%), using them on a larger quantity of data will generate a very small overhead. For example, for 2560502 bits of data, only 9 parity bits are added. The number of parity bits is computed based on the following formula:

$$2^r > d + r + 1 \tag{9}$$

In the formula above, r is the number of redundant bits and d is the number of data bits. The new information is built by putting parity bits on the positions specific to powers of 2, while the rest of the positions are completed with data bits. Because of its advantages and its simplicity, we added a Hamming code mechanism to our solution.

V. EVALUATION

This section presents an evaluation of the proposed solution, showing the setup of the experiments and the results obtained.

A. SETUP

We implemented and tested our solution using the MobEmu simulator[1] [26], which is able to run routing and dissemination solutions in mobile opportunistic networks. Drop Computing was already implemented in MobEmu, so we simply had to add our consistency mechanisms on top of the existing implementation.

The first set of experiments was realized using the HCMM model [27], which simulates the behavior and the interactions between multiple mobile nodes. It is based on the caveman model, where users can belong to several base communities (called ''home'' communities), but can also have social relationships outside of their home communities (in ''acquainted'' communities). For this scenario, we simulated a Drop Computing network with the following parameters: 30 mobile nodes, split into 5 different communities, 6 hours of interactions, with 5 of the nodes being travelers that can move between communities. The physical space was simulated as a 1000x1000-meter grid, and the speed of the nodes was set to vary between 1.25 and 1.5 m/s (i.e., the average human speed). Finally, the transmission radius of the nodes was set to 10 meters, which is an approximation of Bluetooth range.

Along with this synthetic model, we also used two real-life mobility traces collected at our faculty (and available in the CRAWDAD archives,[2]) called UPB 2011 [28] and UPB 2012 [29]. They are two traces taken in an academic environment at the University Politehnica of Bucharest, where the participants were students and teachers. UPB 2011 includes

[1] Available at https://github.com/raduciobanu/mobemu.
[2] https://crawdad.org/all-byname.html.

TABLE 1. Testing scenarios.

Version	Mechanisms	Observation
v1	Default Drop Computing (no corruption detection)	data can be corrupted any time
v2.1	v1 + 3 expected versions	-
v2.2	v1 + 4 expected versions	extra version expected if majority is 50%
v3	v1 + 2 expected versions + rating	2 experimentally proven to be the best value
v4	v2.1 + at most 2/3 versions executed by the same node	-
v5.1	v2.1 + Hamming	-
v5.2	v2.2 + Hamming	-
v5.3	v1 + Hamming	-
v5.4	v3 + Hamming	no multiple versions expected

data collected for a period of 25 days by 22 participants, while UPB 2012 had a duration of 64 days and involved 66 participants.

The proposed implementation was evaluated using three metrics, which allowed us to make measurements on multiple data consistency and corruption scenarios. These metrics were:

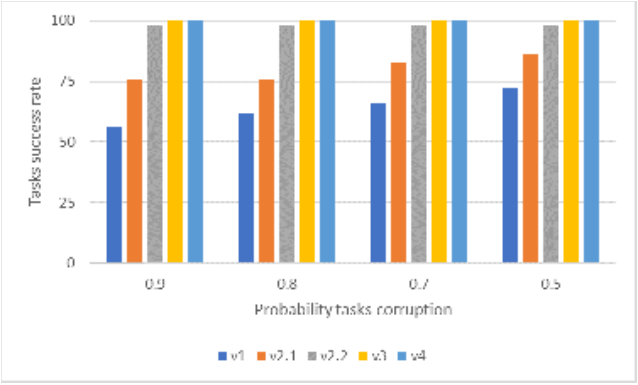
- the percentage of correct tasks received by their owners, which were executed by other nodes in the network
- the total number of the tasks executed by nodes other than their owners (this number varies according to the number of versions of the same task which must be expected before making a decision)
- the average processing latency for every task executed by other nodes (the time elapsed from the generation of the task until the moment when the owner accepts its execution).

The measured values were analyzed both quantitatively and qualitatively in the following situations (also presented in Table 1):

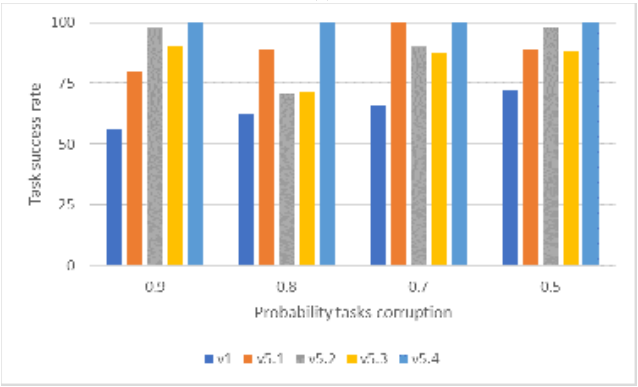
- when data is corrupted, taking into account both scenarios described in Section IV-B (i.e., on task execution or when disseminating an executed task), but no consistency mechanisms are used
- when varying the number of expected versions (3 or 4, with the mention that, if there are two versions with the same frequency, the node waits for another one for that task)
- when the rating mechanism is activated or deactivated (in the affirmative case varying the value of the rating's benchmark)
- when the percentage of different executors varies depending on the number of desired versions
- when using Hamming correction and detection codes (varying the number of permitted corruptions of the same task).

B. RESULTS

In this section, we present the results obtained following the analysis of the proposed solution for the HCMM mobility model and for the two traces (UPB 2011 and UPB 2012).



(a)



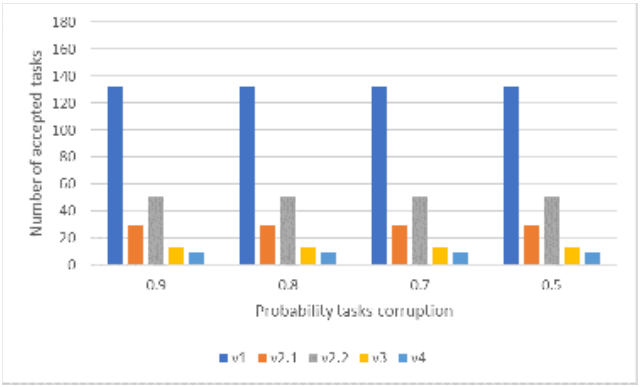
(b)

FIGURE 5. Task success rate for HCMM (for more information about the testing scenarios, please see Table 1).

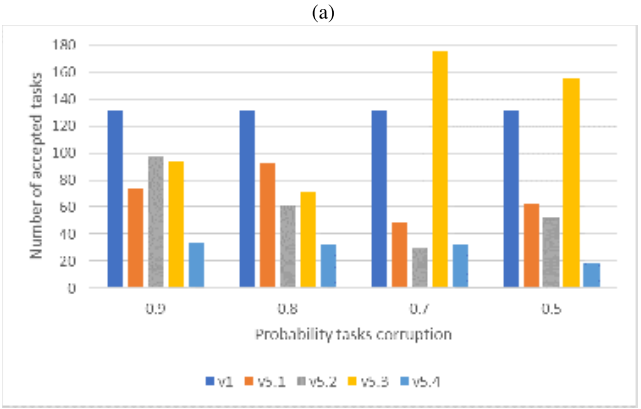
1) HCMM MOBILITY MODEL

Figure 5 shows the percentage of tasks correctly executed and transferred by the network nodes back to their owners. In Figure 5a, it can be observed that the standard version *v1* (where no corruption detection mechanisms are used) has a correctness percentage noticeably lower than all versions implemented in this paper (for a corruption probability of 90%, the task success rate for default Drop Computing is barely above 50%). On the other hand, some of our proposed techniques provide a fairness percentage of accepted tasks as high as 100%. Along with this perfect percentage, we also observe a considerable reduction in corrupted versions in the network by using the rating mechanism. Thus, from 857 corrupted tasks in the standard version, there were 15 corruptions in the *v3* scenario, suggesting that an efficient filtering of the corrupted nodes was produced, the data no longer being predisposed to corruption due to the avoidance at critical moments of nodes with low ratings. Furthermore, it can be observed that the highest success rate is obtained when using the rating mechanisms, or when imposing task execution at different nodes in the network.

Figure 5b shows all scenarios that use Hamming correction and detection codes, and it is important to observe how the percentage of tasks correctly received does not increase proportionally with the decrease of the probability of corrupted

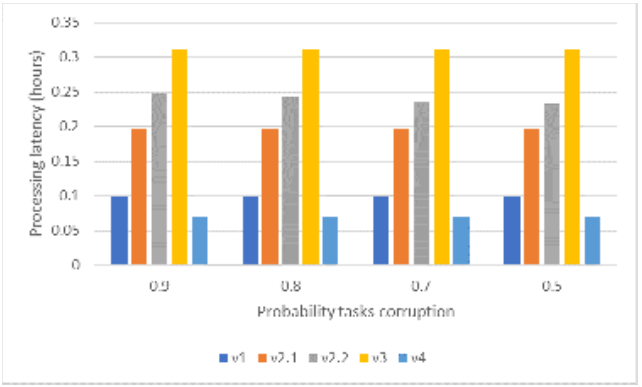


(a)

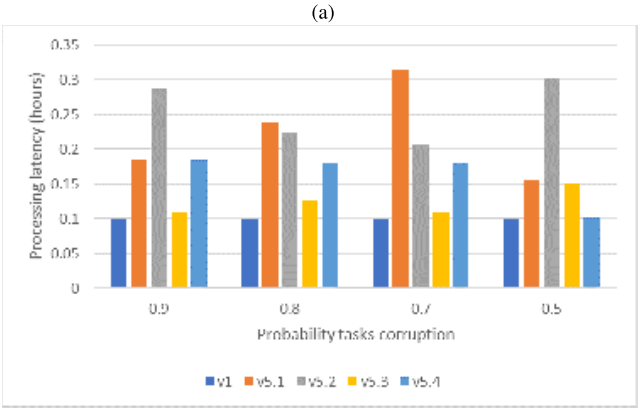


(b)

FIGURE 6. The number of executed tasks in the network for HCMM.



(a)



(b)

FIGURE 7. Processing latency for HCMM.

nodes. This behavior is due to the fact that, at the time of corruption, a random number between 1 to 3 is chosen, with the possibility that the proportion of corruption with more than one bit is higher even if fewer corruptions are recorded. However, it can be seen that in all scenarios shown in Figure 5, a better share of correct results is recorded than in the standard version, *v1*, which confirms the improvements brought about by our solution. The methods that seem to work best in this scenario are when we are expecting more versions of a task and when we are employing the rating mechanism.

The lower value of the number of executed tasks in the network is confirmed by Figure 6. The charts show that our solution provides a degree of reliability regarding the correctness of information at the cost of a lower number of tasks solved. This is due both to node filtering and to the fact that these simulations were made with the same amount of waiting time of a node for any task. Thus, with the increase of the waiting versions, chances for a node to execute its own tasks using only its own resources are also increased.

Figure 7 shows how the average processing latency for each given task executed in the network increases, but there are cases where its value is similar or even lower than the one in the scenario with no data consistency mechanisms (e.g., *v4* or *v5.3*). *v4* is the most incontestable example demonstrating the improvements made, because it produces both a lower latency and a percentage of 100% of correct tasks accepted, as highlighted in Figure 5a. The existence of these situations shows that the entire network was efficiently covered, tasks being shared through nodes that actually had more chances to meet the tasks' owners, despite the decrease of transfer possibilities. Increased processing latency is somewhat inevitable when multiple versions are expected, but it can be seen that the percentage of duration increase is not proportional to the percentage of increase in the number of expected versions.

In Figure 8, we analyze the three metrics described in Section V-A (*m1-m3*) and an extra metric (*m4*), which is the total number of corrupted task versions existing in the network. We show the latency (*m2*) in minutes, in order to be able to represent all metrics (each with its own measurement unit) on the same chart. We vary the number of expected versions of a task, attempting to see what parameters to use to maximize all four metrics. It can be observed that the best values (lowest processing latency, most tasks executed in the network, and a maximum percentage of correct tasks) are obtained when the number of expected versions of a message is 2. In this case, if two different versions arrive, a third one will also be expected, which acts as a tie-breaker. The conclusion drawn here is not necessarily true for any mobile network scenario, so a more suitable way would be to dynamically detect the behavior of the corruption detection and prevention mechanism at runtime. Similarly to the work

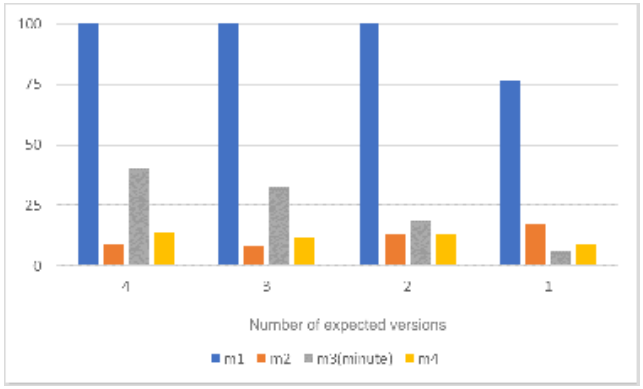


FIGURE 8. Rating mechanism analysis for HCMM (*m1* is the percentage of correct tasks that return to their owner after being executed by other nodes in the network, *m2* is the number of the tasks executed by other nodes, *m3* is the processing latency, and *m4* is the total number of corrupted task versions existing in the network.

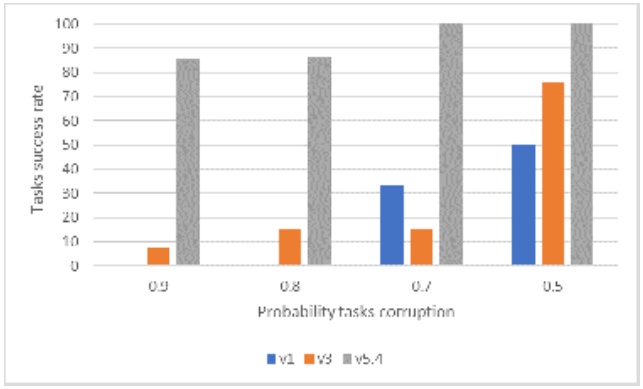


FIGURE 9. Task success rate for UPB 2011.

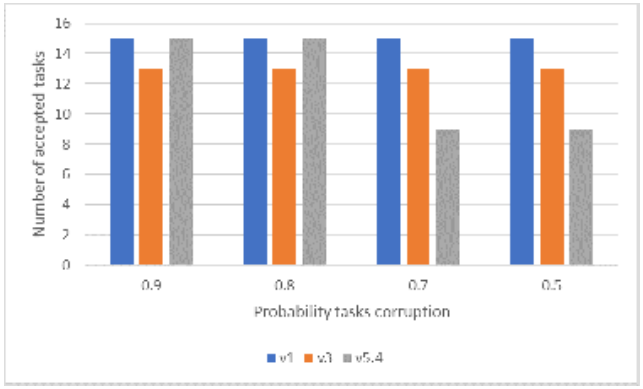


FIGURE 10. The number of executed tasks in the network for UPB 2011.

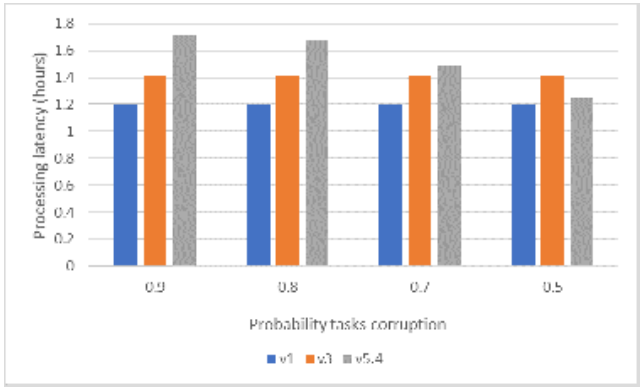


FIGURE 11. Processing latency for UPB 2011.

in [30], this would assume solving a multi-objective optimization problem offline based on various network metrics, and then using the information learned to set the suitable algorithm version.

2) UPB 2011 TRACE

The results obtained for the UPB2011 trace in terms of task success rate are highlighted in Figure 9. Due to the fact that only 20 mobile nodes were used in this simulation, the percentage of nodes that are predisposed to corruption (30%) is higher than in the HCMM simulation (26%), all simulations being made with the assumption that 1 in 3 nodes can corrupt information. Thus, for a high probability of corruption (more than 90%), there are no correctly received tasks if no data consistency mechanisms are employed (*v1* in Figure 9). Taking into account that this scenario has a small number of node connections, implementations of our solution with multiple waiting versions or different executors could not be used. However, using the rating mechanism, as well as Hamming detection and correction codes, an improvement of up to 80% of correctly received tasks is obtained. In Figure 9, at probability 0.7, it can be seen that the percentage of corrupt accepted tasks after applying the rating mechanism is lower than that obtained in the standard version. This is an isolated case due to the decreased rating of a node that has not yet been corrupted. Therefore, due to the lower number of connections between nodes, the current node fails to increase its rating above the reference value, not having enough opportunities to transfer the correct information.

Regarding the number of accepted tasks, Figure 10 shows that, for a high probability of task corruption, our proposed solution manages to execute the same number of tasks as the default version (*v1*), but instead of the tasks being corrupted, they are correct (so this means that our solution is able to correct all corrupted tasks in certain situations). From the results depicted in Figure 9 and Figure 10, we can compute the total number of correct tasks that reach the user (i.e., are accepted by the mobile node and are also correct). For the default Drop Computing scenario, where no corruption detection and prevention mechanisms are employed, the total number of correct tasks is 8 when the corruption rate is 50%. On the other hand, for the *v3* version of our proposed solution, there are 10 correct tasks after a simulation run, while for *v5.4* there are 9 correct tasks. The latter version also has the advantage that all tasks that are accepted are correct, so the user has the guarantee that task results cannot be corrupted.

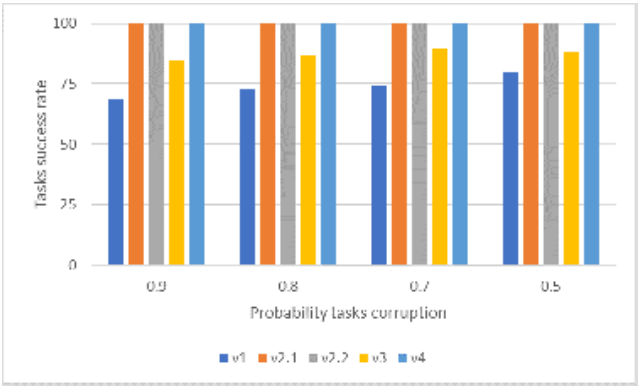


FIGURE 12. Task success rate for UPB 2012.

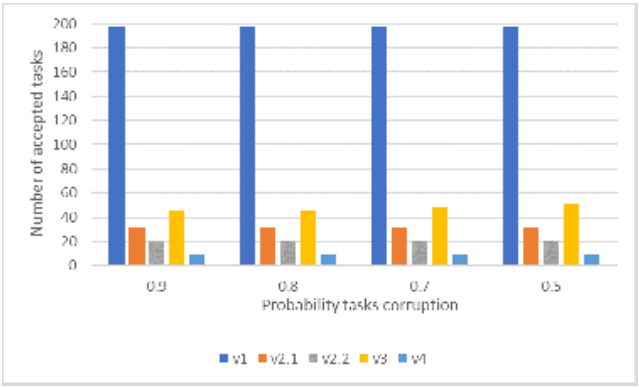


FIGURE 13. The number of executed tasks in the network for UPB 2012.

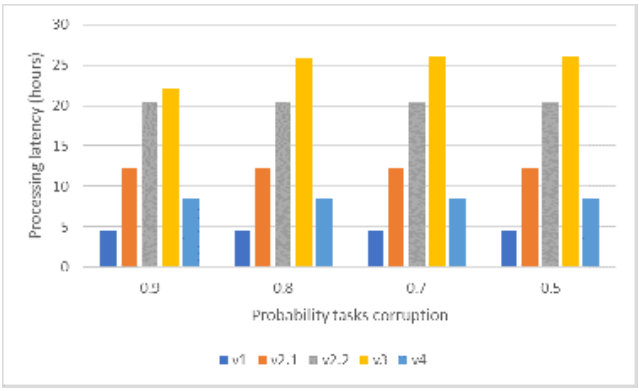


FIGURE 14. Processing latency for UPB 2012.

Figure 11 shows the latency of our solution compared to the default case (*v1*). It can be observed that, as expected, the latency is higher when corruption detection mechanisms are employed, but the differences are small, and we believe that they are negligible and acceptable if one wants to obtain correct data.

3) UPB 2012 TRACE

Regarding the UPB 2012 trace, Figure 12 shows that versions *v2.1*, *v2.2* and *v4.1* manage to achieve a 100% value for the number of correct accepted tasks. When employing the rating mechanism, the maximum percentage of correct tasks is not achieved, but, because of the lower number of expected versions, a higher number of tasks are executed and returned to their owner, as seen in Figure 13. The UPB 2012 trace contains data collected for 1507 hours and is much sparser than UPB 2011 and HCMM, so the processing latency values shown in Figure 14 are naturally higher. Nodes meet much rarer than in previous simulations, so the duration between the time a task is sent to the Drop Computing network and the time it arrives back to its original owner is somewhat high.

In conclusion, the results presented in this section show that our implementation can lead to a high percentage of correct tasks received by their owner after being computed in the Drop Computing network. This suggests that corrupt nodes are correctly filtered out and thus avoided, through optimized exchanges of useful messages. There is still plenty of work to be done to improve the results, and, in the near future, we would like to focus on decreasing the processing latency and increasing the number of executed tasks in the network. We believe that we can achieve this through an optimal detection of nodes that tend to corrupt data (either because they belong to malicious individuals, or because they suffer hardware or software malfunctions), in order to avoid them and learn when the corruptions appear.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the Drop Computing paradigm, which combines edge and fog computing with mobile network and social information to decrease latency and power consumption. We presented several use cases for Drop Computing, including an AAL scenario where we show that it can be employed in an elderly care facility to reduce costs.

Then, we proposed data consistency mechanisms for Drop Computing, assuming a scenario where mobile nodes want to solve some computation tasks and thus offload them to devices in proximity in an opportunistic fashion. The thorough experimental testing showed that, through setting appropriate restrictions, our consistency solution can satisfy the requirements of a network with regard to a desired trust level, since the proposed rating mechanism can lead to a task correctness as high as 100%. Furthermore, this happens without the latency and the number of tasks executed in the network being affected too much.

For future work, our aim is to improve the rating mechanism in order to obtain higher hit rates, as well as lower latencies and overhead. Moreover, we wish to come up with methods to increase the altruism of Drop Computing nodes, incentivizing them to participate in the collaborative network. This would be done by integrating reward mechanisms for nodes that execute and disseminate computing tasks. Finally, we also wish to implement a Reed-Solomon code [31] as an improvement over the Hamming mechanism, since it would be able to detect corrupted bits based on the number of parity bits added in the payload.

ACKNOWLDEGMENT

The authors would like to thank the networking support by the COST Action CA16226, ''SHELD-ON: Indoor Living Space Improvement: Smart Habitat for the Elderly.'' They would also like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

REFERENCES

[1] K. Akherfi, M. Gerndt, and H. Harroud, ''Mobile cloud computing for computation offloading: Issues and challenges,'' *Appl. Comput. Informat.*, vol. 14, no. 1, pp. 1–16, 2018.
[2] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, and S. U. Khan, ''Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges,'' *J. Netw. Comput. Appl.*, vol. 52, pp. 154–172, Jun. 2015.
[3] 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, 2015.
[4] R. Roman, J. Lopez, and M. Mambo, ''Mobile edge computing, Fog *et al.*: A survey and analysis of security threats and challenges,'' *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, Jan. 2018.
[5] 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, pp. 889–899, Mar. 2017.
[6] V.-C. Tabusca, R.-I. Ciobanu, and C. Dobre, ''Data consistency in mobile collaborative networks based on the drop computing paradigm,'' in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE)*, Oct. 2018, pp. 29–35.
[7] 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: ACM, 2010, pp. 6:1–6:5. doi: 10.1145/1810931.1810937.
[8] N. Fernando, S. W. Loke, and W. Rahayu, ''Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing,'' in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput. (UCC)*, Washington, DC, USA: IEEE Comput. Soc., Dec. 2011, pp. 281–286. doi: 10.1109/UCC.2011.45.
[9] 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 (MCS)*. New York, NY, USA: ACM, 2012, pp. 9–14. doi: 10.1145/2307849.2307854.
[10] B. Zhou, A. V. Dastjerdi, R. N. 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.
[11] 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 (MCS)*. New York, NY, USA: ACM, 2012, pp. 29–36. doi: 10.1145/2307849.2307858.
[12] T. Hara and S. K. Madria, ''Consistency management strategies for data replication in mobile ad hoc networks,'' *IEEE Trans. Mobile Comput.*, vol. 8, no. 7, pp. 950–967, Jul. 2009.
[13] P. Nithiyalakshmi and V. U. Kumar, ''Data consistency for cooperative caching in mobile environments,'' *Int. J. Sci. Res.*, vol. 3, no. 1, p. 1, 2014.
[14] R.-I. 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.
[15] R. Ciobanu, C. Dobre, and V. Cristea, ''Reducing congestion for routing algorithms in opportunistic networks with socially-aware node behavior prediction,'' in *Proc. IEEE 27th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Barcelona, Spain, Mar. 2013, pp. 554–561. doi: 10.1109/AINA.2013.63.
[16] Y. Yu, ''Mobile edge computing towards 5G: Vision, recent progress, and open challenges,'' *China Commun.*, vol. 13, no. Supplement2, pp. 89–99, 2016.
[17] D. He, C. Chen, S. Chan, J. Bu, and A. V. Vasilakos, ''ReTrust: Attack-resistant and lightweight trust management for medical sensor networks,'' *IEEE Trans. Inf. Technol. Biomed.*, vol. 16, no. 4, pp. 623–632, Jul. 2012.
[18] E. J. Pauwels, A. A. Salah, and R. Tavenard, ''Sensor networks for ambient intelligence,'' in *Proc. IEEE 9th Workshop Multimedia Signal Process. (MMSP)*, Oct. 2007, pp. 13–16.
[19] I. F. Akyildiz, X. Wang, and W. Wang, ''Wireless mesh networks: A survey,'' *Comput. Netw.*, vol. 47, no. 4, pp. 445–487, Mar. 2005.
[20] G. U. O. W. Wendy, W. M. Healy, and Z. Mengchu, ''Wireless mesh networks in intelligent building automation control: A survey,'' *Int. J. Intell. Control Syst.*, vol. 16, no. 1, pp. 28–36, 2011.
[21] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. C. M. Leung, ''Body area networks: A survey,'' *Mobile Netw. Appl.*, vol. 16, no. 2, pp. 171–193, 2011.
[22] B. Latré, B. Braem, I. Moerman, C. Blondia, and P. Demeester, ''A survey on wireless body area networks,'' *Wirel. Netw.*, vol. 17, no. 1, pp. 1–18, Jan. 2011. doi: 10.1007/s11276-010-0252-4.
[23] G. Acampora, D. J. Cook, P. Rashidi, and A. V. Vasilakos, ''A survey on ambient intelligence in healthcare,'' *Proc. IEEE*, vol. 101, no. 12, pp. 2470–2494, Dec. 2013.
[24] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, and B. Schroeder, ''An analysis of data corruption in the storage stack,'' *Trans. Storage*, vol. 4, no. 3, pp. 8:1–8:28, Nov. 2008. doi: 10.1145/1416944.1416947.
[25] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, ''A limited memory algorithm for bound constrained optimization,'' *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, 1995.
[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] 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, 2010.
[28] R. I. Ciobanu, C. Dobre, and V. Cristea, ''Social aspects to support opportunistic networks in an academic environment,'' in *Proc. Int. Conf. Ad-Hoc Netw. Wireless (ADHOC-NOW)*, Belgrade, Serbia, Springer, Jul. 2012, pp. 69–82.
[29] 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.
[30] R.-I. Ciobanu, C. Dobre, D. G. Reina, and S. L. Toral, ''A dynamic data routing solution for opportunistic networks,'' in *Proc. 14th Int. Conf. Telecommun. (ConTEL)*, Jun. 2017, pp. 83–90.
[31] A. R. de Araujo Zanella and L. C. P. Albini, ''A reed-solomon based method to improve message delivery in delay tolerant networks,'' *Int. J. Wireless Inf. Netw.*, vol. 24, no. 4, pp. 444–453, 2017.

RADU-IOAN CIOBANU received the B.S., M.S., and Ph.D. degrees *(summa cum laude)* from the Faculty of Automatic Control and Computers, University Politehnica of Bucharest, in 2010, 2012, and 2016, respectively. He has worked in mobile devices for more than eight years, having experience in both startups (VirtualMetrix) and 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.

During his still young career, he has been involved in several national and international research projects in mobile and cloud computing, the IoT, and ambient assisted living. 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 (IEEE GLOBECOM, ICC, IM, and WoWMoM.)

VLĂDUȚ-CONSTANTIN TĂBUȘCĂ received the B.S. degree with a thesis about drop computing, in 2018. He is currently pursuing the master's degree in advanced software systems with the Faculty of Automatic Control and Computers, University Politehnica of Bucharest. His research interests include mobile opportunistic networks, cloud computing, the Internet of Things, and ambient assisted living systems.

CIPRIAN DOBRE received the Ph.D. and Habilitation degrees. He received the Ph.D. Scholarship from the California Institute of Technology, USA, and another one from Oracle. He is currently a Professor and leads the MobyLab Laboratory on Pervasive Products and Services, University Politehnica of Bucharest, and the National Institute for Research and Development in Informatics (ICI), Bucharest. 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 the Gheorghe Cartianu Award 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 journals 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).

LIDIA BĂJENARU was born in Barlad, Romania, in 1962. She received the B.S. and M.S. degrees in computer engineering from the Technical University ''Gheorghe Asachi'', Iasi–Faculty of Electrotechnics, Automation and Computer Science, in 1985, and the Ph.D. degree *(magna cum laude)* in economics informatics from the Bucharest University of Economical Studies, in 2017.

From 1995 to 2013, she was a Training Expert in computer science with the Computer Training Centre S.A. Bucharest, conducting didactic activities in information and communication technology using modern online training solutions, in parallel with ICT-applied research activities. Since 2013, she has been a Principal Senior Analyst with the Department of Systems and Applications for Society, National Institute for Research and Development in Informatics, Bucharest. She authored and co-authored more than ten books and more than 50 scientific papers in journals and volumes of conferences in national and international publications. She coordinated and has been member in research teams in more than 40 national and international projects. Her research interests include education, e-learning, e-health, mobile computing, artificial intelligence, computer ontologies, e-services, e-government, social networks, and cloud computing.

She is a member of professional associations, national, and international scientific committees.

CONSTANDINOS X. MAVROMOUSTAKIS (SM'03) received a five-year Diploma of Engineering (B.S., B.Eng., M.Eng./KISATS approved/accredited) degree in electronic and computer engineering from the Technical University of Crete, Greece, the M.S. degree in telecommunications from the University College of London, U.K., and the Ph.D. degree from the Department of Informatics, Aristotle University of Thessaloniki, Greece.

He is currently a Professor with the Department of Computer Science, University of Nicosia, Cyprus. He is leading the Mobile Systems Laboratory, Department of Computer Science, University of Nicosia. He has participated in several FP7/H2020/Eureka and National projects.

He has a dense research work outcome in mobile and wearable computing systems and the Internet of Things, consisting of numerous refereed publications including several Books (IDEA/IGI, Springer, and Elsevier). He has served as a Consultant to many industrial bodies (including Intel Corporation LLC). He is a Management Member of the IEEE Communications Society Radio Communications Committee and a Board Member of the IEEE-SA Standards IEEE SCC42 WG2040. He has been an Active Member (Vice Chair) of IEEE/ R8 regional Cyprus Section, since 2016. Since 2009, he has been serving as the Chair of C16 Computer Society Chapter of the Cyprus IEEE Section. He is also a Co-Founder of the IEEE Technical Committee on IEEE SIG on Big Data Intelligent Networking (IEEE TC BDIN SIG). He also serves as a Vice Chair.

GEORGE MASTORAKIS received the B.S. degree in electronic engineering from UMIST, in 2000, the M.Sc. degree in telecommunications from UCL, in 2001, and the Ph.D. degree in telecommunications from University of the Aegean, in 2008. He is serving as an Associate Professor with the Department of Business Administration and as a Research Associate in research and development with the Telecommunications Systems Laboratory, Technological Educational Institute of Crete, Greece.

He has more than 250 publications in various international conferences proceedings, workshops, scientific journals, and book chapters. His research interests include cognitive radio networks, the Internet of Things, energy-efficient networks, big data analytics, and mobile computing.