Drop Computing: Ad-Hoc Dynamic Collaborative Computing

Radu-Ioan Ciobanu[a], Catalin Negru[a], Florin Pop[a,1], Ciprian Dobre[a,1,*], Constandinos X. Mavromoustakis[c], George Mastorakis[d]

[a] *Faculty of Automatic Control and Computers, University Politehnica of Bucharest, 313 Splaiul Independentei, Bucharest, Romania*
[b] *National Institute for Research and Development in Informatics (ICI) 8-10 Maresal Averescu, Bucharest, Romania*
[c] *University of Nicosia, Department of Computer Science, 46 Makedonitissas Avenue, 1700, Nicosia, Cyprus*
[d] *Technological Educational, Institute of Crete, Estavromenos 71500, Heraklion, Crete, Greece*

Abstract

Mobile applications nowadays generally consist of a frontend component running on the device, and a backend component running on the cloud that performs the larger computations. However, this usage model leads to high costs for developing the application (since a cloud infrastructure that should scale to the number of users must be maintained), and to a potentially bad user experience (if the latency is high or the users employ mobile broadband they pay for). Thus, we introduce the Drop Computing paradigm, which proposes the concept of decentralized computing over multilayered networks, combining cloud and wireless technologies over a social crowd formed between mobile and edge devices. Mobile devices and people interconnect to form ad-hoc dynamic collaborations to support the equivalent of a crowd-based edge multilayered cloud of clouds, where the capabilities of any mobile device are extended beyond the

[☆] This work was supported by project MobiWay (PN-II-PT-PCCA-2013-4-0321), DataWay (PN-II-RU-TE-2014-4-2731), and Traffic and Data Offloading in Mobile Networks – TTOff, H2020 as part of Measuring Mobile Broadband Networks in Europe.

[*] Principal corresponding author

Email addresses: radu.ciobanu@cs.pub.ro (Radu-Ioan Ciobanu), catalin.negru@cs.pub.ro (Catalin Negru), florin.pop@cs.pub.ro (Florin Pop), ciprian.dobre@cs.pub.ro (Ciprian Dobre), mavromoustakis.c@unic.ac.cy (Constandinos X. Mavromoustakis), gmastorakis@staff.teicrete.gr (George Mastorakis)

local technology barriers, to accommodate external resources available in the crowd of other devices. Thus, instead of every data or computation request going directly to the cloud, Drop Computing employs the mobile crowd formed of devices in close proximity for quicker and more efficient access. Devices in the mobile crowd are leveraged for requesting already downloaded data or performing computations, and the cloud acts as the second (or even third) option. We present a proof-of-concept implementation of Drop Computing and show, through simulations, that it is feasible for real-life usage, since it is able to drastically reduce costs while not affecting or even improving the user experience.
Keywords: edge, cloud, mobile, IoT, fog

1. Introduction

With more functionalities being constantly added for an increased quality of experience (QoE), modern mobile applications rely heavily on the cloud for support. Although mobile devices are equipped with advanced computing capabilities, they have limitations in terms of what they can do. Thus, the cloud can help the devices do the "heavy lifting" when necessary [1]. However, today we witness the phenomenon where mobile cloud computing (i.e., the interconnection between cloud and mobile computing, served by wireless and broadband networks) introduces communication challenges due to increased network overhead.

Current mobile cloud ecosystems rely on centralized, brokered communications. Connections between mobile devices go exclusively through the Internet, even if they happen to be a few feet apart. Thus, this model will not be able to respond to the growing needs of the huge ecosystems of mobile devices of tomorrow [2]. One possibility to cope with this problem is to limit the amount of data sent to global cloud platforms, and move part of the data processing tasks to the edge of the network, close to where data is generated. This is the vision of (Mobile) Edge Computing [3], whereby computing tasks are not exclusively allocated on centralized cloud platforms, but are distributed towards the edge

of the network (as in the Internet of Things and Fog Computing paradigms [4]), and transferred closer to the business thanks to Content Delivery Networks.

The traditional gateway becomes a set-top-box machine, with additional computation and storage capabilities, where micro-tasks can be offloaded first, instead of directly to the cloud. Not only can Mobile Edge Computing offload excessive traffic from wireless access networks, but it can also be a more suitable approach to extract knowledge from privacy-sensitive data, which should not be transferred to third parties for processing [2]. However, most often mobility is not correctly captured, as offloading assumes the user is connected for a long time to the in-house router.

Such computing paradigms were designed with a common scope: to bring data and processing capabilities closer to the devices in need. The current evolution is heading towards an increasingly interconnected, mobile, and ubiquitous Internet of networks, which includes components ranging from small wireless sensor networks to extended local area networks, all of them remotely (and transparently) accessible. However, all these paradigms are still based on the traditional client/server model: a device sends a request for an operation that is being served back by a delegated provider. Today, we predict this model to simply not be enough anymore. We can see a future where decentralization is perceived not as a problem that introduces complexity in the design of applications, but as a complementary solution to today's technology.

We introduce the Drop Computing paradigm, which proposes the concept of decentralized computing over multilayered networks combining cloud and wireless technologies, over a social crowd formed between mobile and edge devices. Mobile devices and people interconnect to form ad-hoc dynamic collaborations to support the equivalent of a crowd-based edge multilayered cloud of clouds, where the capabilities of any mobile device are extended beyond the local technology barriers, to accommodate external resources available in the crowd of other devices. Thus, instead of every data or computation request going directly to the cloud, Drop Computing employs the mobile crowd formed of devices in close proximity for quicker and more efficient access. Devices in

the mobile crowd are leveraged for requesting already downloaded data or performing computations, and the cloud acts as the second (or even third) option. This has the potential to lead to lower costs for developers, as well as improved performance, as we show in Section 4.

The rest of this paper is structured and follows. In Section 2, we present general concepts regarding Fog and Mobile Edge Computing, highlighting the additional benefits that our proposed Drop Computing paradigm brings. In Section 3, we discuss Drop Computing and describe some use cases, while also performing an in-depth comparison between Drop Computing and other similar solutions, showing the improvements that our framework can offer. In Section 4, we present a proof-of-concept implementation of Drop Computing using an opportunistic network emulator, and we describe two test scenarios and analyze their results. Finally, in Section 5, we present our conclusions and future work.

2. Related Work

When the cloud data center model is not enough anymore, a horizontal scaling of communication can complement the vertical scaling of the cloud infrastructure. Technology recently started to move further away from the traditional cloud paradigm of an off-the-premises rented resource market that is served primarily by external data centers, and we witness a return to a more traditional grid-like future, where resources from all over the world are fused together into the "Grid" and commonly used for a greater goal [5]. Instead of externalizing all the business to the cloud, the cloud is brought closer to the business through set-top-box equipment and cloudlet constructs, and we witness the rise of paradigms such as the Internet of Things (IoT) and Fog Computing, where processing of sensed data generated by mobile devices is done on machines running closer-than-cloud, in the same network as the sensing machines themselves. With caching technologies, cloud data is accessed at lower latencies, because it is transferred closer to the actual business, thanks to Content Delivery Networks (CDNs).

In this context, Mobile Edge Computing (MEC) [6, 7] proposes a model where applications run at the edge of the network, through a novel network architecture concept that enables cloud computing capabilities and an IT service environment at the edge of the cellular network [8]. The network operator faces today the problem of cellular congestion, which in MEC is solved by running applications and performing related processing tasks closer to the cellular customer. Beck et al. [9] propose a taxonomy for Mobile Edge Computing systems, splitting them by application type. Thus, the authors consider that MEC systems can successfully be employed for offloading (of data and computations), edge content delivery (distributed database management systems, geoinformation, etc.), aggregation (Big Data at edge devices for avoiding extra traffic), local connectivity (private networking, edge broadcasting), content scaling (for Web TV and radio, media transformation, etc.), or augmentation.

Similar in scope with MEC, but oriented towards data storage rather than communication, Fog Computing [10, 11, 12] is a paradigm that extends the cloud in support of the IoT vision in the mobile world. The Internet of Things is already generating an unprecedented volume and variety of data. But, by the time the data makes its way to the cloud for analysis, the opportunity to act on it might be gone. Thus, the fog extends the cloud to be closer to the things that produce and act on the IoT data. Such devices presumably can be deployed anywhere with a network connection (since any device with computing, storage, and network connectivity can be a fog node).

What Drop Computing adds to the Mobile Edge and Fog Computing models (as will be shown in the next section) is the social crowd component. It aims to construct the theoretical foundation of a new model of doing computation centered on the device rather than the cloud, providing horizontal computation scaling by considering any device as a potential cloud resource. Drop Computing performs collaborative device-centered computing wherever and whenever the device needs it. Many mobile apps today use computation offloading, but mobile clouds incur delays that negatively affect the end-user perceived application responsiveness. A solution where the mobile device could extend and actively

"borrow" computation cycles from geographically co-located devices offers several advantages: the communication time is lowered compared to sending the data to the cloud; by dividing computation into fine-grained computing tasks that are sent to different devices for processing, the model can solve many of the privacy issues associated with who sees what; finally, the collaborative cloud of devices could better conserve the battery power of devices (i.e., one device in need of executing critical tasks with low battery could export computation tasks with lower priority to another device with more battery, or that is plugged to a power outlet).

3. Drop Computing: Decentralization of Mobile Computing

The main goal of Drop Computing is to offer a new computing model for mobile applications with many constraints, running over crowd-based networks of mobile and edge-cloud devices, and with Key Performance Indicators (KPIs) such as reducing operational costs, minimizing the energy consumption, and increasing the performance of mobile devices, compared to alternative computing models.

3.1. Presentation

The need for Drop Computing arises from the insufficiency of the classic cloud model, which is not suitable anymore for the Internet of Things. When a large number of small devices communicate, they all need to send requests to the cloud, receive them back, and then process them. For example, a smart refrigerator would need to communicate through the cloud with the car to see when the owner arrives home, and then to communicate with the grocery's servers (also through the cloud) to place orders for missing milk. This could be simplified by allowing communication solely between the specified devices [13], reducing latency and congestion in the cloud. Moreover, this would also decrease the risk of privacy and security issues that may arise by sending data to the

Internet. The well-known Shodan search engine[1] allows users to find any IoT device connected to the Internet and even connect to it. Thus, anyone can gain access to webcams or other devices that may offer information about a home owner's whereabouts (e.g., a thief may know whether someone is home or not, which would help them plan a potential break-in). Furthermore, lowering the number of cloud accesses might reduce the costs of running a home IoT network, since less server resources would be required.

Drop Computing elaborates on the feasibility of allowing the resources of the mobile users to be manipulated based on their social relationships and ties. By leveraging the human social tie, the device-to-device (D2D) communication is positively affected by conserving energy on each mobile device. As online social interaction relationships among people (through their devices) are broadened and significantly enhanced [14], the users can passively cooperate through their connections, in order to enable a significant minimization of the computational load and the energy consumption levels. As this communication is subject to connection-oriented constraints, the users could exploit the social ties discovered on their mobile devices (through social interaction), in order to trigger some communication-oriented parameters. The proposed scheme targets offloading efficiency for optimizing the mobile resources utilization (i.e., GPU, CPU, RAM and battery consumption per second), which can result in a significant extension of mobile device lifetime.

From a foundational perspective, Drop Computing proposes a shift towards a decentralized computing model for supporting collaborative computation in clouds formed of mobile/wearable devices, as shown in Figure 1. Unlike P2P systems and volunteer computing, Drop Computing runs over ad-hoc, dynamic, and decentralized networks, formed between mobile and edge devices that interconnect to support the equivalent of a crowd-based edge/mobile cloud. Mobile devices exchange data over available wireless communication protocols, in order to organize themselves. On top, computing/storage services and mobile apps

[1] `https://www.shodan.io`.

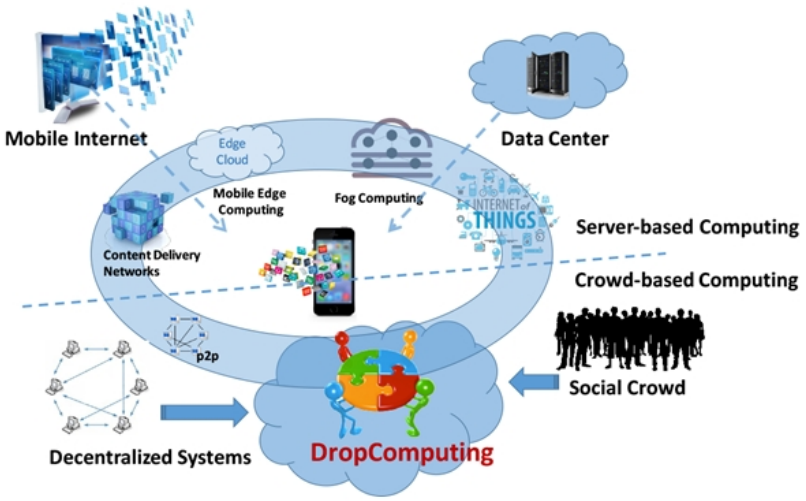


Figure 1: The Drop Computing vision.

(defined as droplets) available in the crowd-based edge cloud make new programming models for decentralized apps possible. The capabilities of any single device are thus extended beyond the technology barriers of the local hardware. Resources that are available in the geographically-dispersed crowd of other devices are accommodated, with hybrid links to further incorporate the edge cloud and even cloud-available resources. This is, in some sense, a decentralized vision of the peer-to-peer model, where tasks are offloaded and data is stored in secure and reliable ways at nodes in the trusted network. The social crowd vision places the people instead of machines at the heart of the computation model.

Drop Computing offers the means to transparently access resources available in the proximity-based network. Content sits closer to the user, and computations can be performed in the mobile crowd. Data is cached or offloaded from the cloud to set-top-box machines at the edge of the cloud, or to mobile devices capable to serve others (closer to where it is needed). Apps are decomposed into tasks that can be executed locally, or offloaded closer to their data or to more powerful/resourceful devices. Offloading and data caching decisions consider predictions of device and network reliability, energy consumption, and

context metrics as captured by sensors. Recently, it has been shown that mobile computation-intensive applications can be offloaded, in the traditional mobile cloud model, using a social model and communication-oriented diversities [15]. This helps conserve energy, by pre-computing available resources, employing social ties for processing, and estimating the power-cost model.

We already demonstrated the feasibility of phone-to-phone direct wireless communication by simultaneously incorporating modern technologies and protocols such as Wi-Fi, Wi-Fi Direct and the latest Bluetooth variants [16]. With such an approach, some of the communication could be made through direct exchanges, and processing can be (partially) done in the ad-hoc network formed between mobile (i.e., smartphones) and edge (i.e., set-top-box) devices. With such fluid networks, fundamental new algorithms can be developed to process and extract knowledge out of the data collected across a number of physical locations, thus relieving some of the throughput towards the cloud. When combined with an infrastructure-based computation substrate (such as cloud computing), sensor-rich devices leverage human mobility and processing power to enhance the users' ability to communicate, sense and compute in the absence of reliable end-to-end connectivity. The research in [16] explores the feasibility of allowing mobile resources to be manipulated based on their social relationships and ties. As online social interaction relationships among people (through their devices) are broadened and significantly enhanced, users can passively cooperate through their connections, in order to enable a significant minimization of the computational load and the energy consumption levels.

Thus, the Drop Computing vision is to extend the edge cloud model to the extreme, towards the network formed of mobile (and edge) devices. Instead of having a well-known fixed device at the edge of the cloud, a dynamic ad-hoc cluster of co-located devices acts as the edge cloud. The novelty is the usage of mobility as a feature, instead of as an issue that should be circumvented. To our knowledge, such an approach has not been done before, because device mobility has been viewed as a problem. A decentralized vision places people at the heart of the computation-coordination model, the network being formed around

social ties. Social connections between devices would be utilized for deciding the suitability of using resources from other peers. For this, social knowledge, an area which has gained much traction in the field of mobile systems, must be employed. This is because the nodes in such a network are almost exclusively mobile devices carried by humans, whose behavior is governed by social relationships. Their behavior can be modeled according to interests, needs, as well as other socially-connected entities.

3.2. Discussion

There are several real-life scenarios that would surely benefit from mechanisms meant to bring data and resources closer to the end-user. For example, when analyzing mobile computation, we observe that the current trend in mobile cloud computing (MCC) relies on offloading tasks from mobile devices into the cloud [17]. However, mobility is handled through fault-tolerance, which leads to missing opportunities where code could be handled in the local network, even to malfunctioning applications. Furthermore, deciding how to partition an application into remotable code is split into static versus dynamic. Although dynamic partitioning is currently considered better, it implies large overhead that ends up consuming more battery than can be saved, especially in the case of failing to offload. We consider that a proper solution is to employ a programming model that enforces modularization of the application into static and remotable code. Furthermore, it needs to be hidden behind language-specific constructs in order to hide the complexity of offloading [18] and also reduce the training required by application developers in creating scalable applications. We believe that this gap can be filled using Drop Computing as a complementary solution to MCC. Before offloading into the cloud, tasks can be distributed to nearby devices that would be able to execute them in a more feasible fashion.

A real-life situation where a model such as Drop Computing can be extremely helpful is the implementation of CPU-intensive mobile applications. For example, let us assume that some developers want to implement a photo-editing app that allows users to take a photo and then perform a series of complex

operations on it (such as applying filters). Some of these operations may be extremely CPU-intensive, so they can be performed in the cloud for better performance (especially for older devices that are slower). However, this means that the developers need to maintain a cloud backend for the application, which should elastically scale based on the popularity of the app. Naturally, this can lead to increasing costs as the application becomes more popular. In this situation, leveraging neighboring devices for performing the CPU-intensive tasks may benefit both the user and the developers. From the standpoint of the user, the computations might be performed faster because the latency of using the cloud is reduced by communicating with devices in range. On the other hand, the developers see the costs of the cloud backend being reduced, because the number of virtual machine (VM) instances required decreases, especially when the users are in crowded areas with many device-to-device contacts.

3.3. Comparison

In this section, we present several solutions that are similar to Drop Computing, and show why we believe that our proposed framework can bring more benefits.

There are two paradigms at the basis of Drop Computing: cloud computing and opportunistic networks (ONs). The latter have evolved from mobile ad hoc networks (MANETs), being a form of delay-tolerant networks (DTNs). The main difference between MANETs (and similar solutions such as vehicular ad hoc networks - VANETs [19]) and ONs is that opportunistic networks are based on the idea that the composing nodes are extremely mobile, and thus routes between them are very dynamic. In this situation, opportunistic networks do not employ routing tables (as MANETs do), because they would change very often and would thus lead to high overheads at each recomputation. Furthermore, two opportunistic network nodes can communicate even if there is no direct path between them, so routing tables are actually not needed. ONs do not see the disconnections between nodes as an issue, but as the normal behavior, while two nodes that are in wireless range have an opportunity to communicate in a

probabilistic fashion, based on the store-carry-and-forward paradigm.

The second paradigm that Drop Computing is based on, cloud computing, assumes that nodes that do not manage to get the data they need or perform the computations they require from their opportunistic network community, will still get their requests fulfilled. Thus, if a Drop Computing node needs to perform a CPU-intensive task but does not have the necessary resources itself, it first tries to offload this task to other Drop Computing nodes in its vicinity. However, if there are no such nodes that can run the task, or if they do not reply in a timely fashion, the task is sent to the cloud, where the service level agreement (SLA) ensures that the task will eventually be executed (provided the node has an Internet connection, in order to reach the cloud service). Thus, Drop Computing does not require other SLAs than the ones offered by the cloud provider, since it is a service that attempts to improve upon the cloud's performance. In other words, if a node's tasks cannot be computed locally by other nodes, they are computed in the cloud, so there is a guarantee that the user will receive the results.

Several solutions that are somewhat similar to Drop Computing have been proposed in the literature so far. In [20], the authors propose a framework to create virtual mobile cloud computing providers, where users with similar goals or tasks collaborate with each other. They each compute parts of a task, and then merge their parts into the final result that they all share. The framework is composed of five components: application manager (modifies applications by adding features for offloading), resource manager (profiles applications and monitors resources), context manager (gathers contextual information and makes it available for other processes), P2P component (communicates with neighboring devices), and offloading manager (sends and manages jobs from a node to neighbors, receives and processes jobs sent by neighbors). However, there are some drawbacks to this solution that Drop Computing addresses. Firstly, the virtual mobile cloud solution assumes that a user that needs help to compute some tasks is at a stable location, so mobility is considered an issue. Movement patterns are not computed, so a device that moves around a lot will not be able

to benefit from this framework. We consider this a strong downside, especially in this day and age when mobile devices such as smartphones are ubiquitous, and people expect their phones to be able to work in any conditions (not only when they are standing still). Furthermore, the authors assume that all devices in the virtual mobile cloud are similar in terms of technical capabilities, whereas Drop Computing accepts any kind of mobile device with close-range communication capabilities (new and old smartphones, tablets, laptops, mobile sensors, etc.). Another benefit that Drop Computing brings is that it uses multi-hop opportunistic communication for performing tasks opportunistically in the network, so there is a higher chance that a suitable task executor is found (whereas the virtual mobile cloud solution only offloads tasks to devices that are one hop away, so to actual neighbors). Moreover, Drop Computing spreads many multiple copies of the same task on different paths, so the chance of the results coming back in due time are high (especially since social community knowledge is used by Drop Computing to select the collaborators and decrease congestion and increase hit rate). Even if suitable nodes cannot be found through Drop Computing, there is always the fallback of connecting to the cloud and computing the task there, whereas the virtual mobile framework assumes that no cloud infrastructure exists.

Another mobile cloud framework is proposed in [21], where mobile devices owned by different individuals help each other perform computational tasks. The framework consists of a resource handler (which deals with resource discovery, establishing connections, exchanging meta data with clients), a context manager (which gathers context data about clients), a cost handler (which estimates costs and selects client devices based on requirements), a transaction module (for performing micropayments between devices), and a job handler (which dynamically partitions an application). The main drawback of this framework is that communication is only performed between neighbors (i.e., single-hop) using Bluetooth. This severely limits the applicability of this solution, because nodes that are multiple hops away cannot be reached. Furthermore, similarly to the virtual mobile cloud computing solution presented above, this framework

also assumes that there is no cloud infrastructure to be used in case device-to-device offloading cannot be performed. Another particularity of this solution that can be viewed as an issue is that the incentive mechanism it uses is based on monetary transactions. This would be difficult to put into place in real-life, especially in an environment where no connection to a global Internet infrastructure is available. In such a case, asymmetric encryption would be hard to implement, since nodes would not have access to a certificate authority.

A mobile cloud computing architecture that runs resource-intensive applications on collections of cooperating mobile devices, while also using cloud services as a backup, is the mCloud platform [22]. It is composed of *mDevs* (which are the nodes in the network) that execute *mTasks*. Tasks can be split into multiple smaller pieces, which can be spread in the network towards several *mDevs*. However, similarly to the two other platforms presented so far in this section, device-to-device communication only occurs in a single-hop manner, so only devices that are in range can offload *mTasks*. The mClouds platform assumes that incentives for participation are being offered by the carriers (since it is in their best interest to reduce congestion) or by the nodes that request the distributed computations. The first solution is somewhat unrealistic, because mobile carriers would be more interested in making their clients pay for using mobile data, rather than use other devices through close-range communication.

A somewhat different solution implies using cloudlets [23], which are composed of trusted, resource-rich nodes which are located in the near vicinity of a mobile user. Thus, in a heterogeneous network composed of all kinds of devices (fixed or mobile), all nodes that are connected to the same LAN network can cooperate by forming a cloudlet. Inside a cloudlet, the composing nodes can work together to help each other solve tasks by offloading from one to another. The cloudlet infrastructure is not fixed, and devices can join and leave the cloudlet at any moment. This framework considers two types of cloudlets, namely elastic (built for this particular purpose in datacenters) and ad hoc (formed on the spot when multiple devices are connected to the same network and at least one of them needs help with some tasks). The devices in a cloudlet are in the same

LAN, whereas devices in different cloudlets must be connected to the global Internet in order to communicate. Therefore, if a node needs to offload some tasks to a different cloudlet than the one it is located in, it needs to contact a service (i.e., a catalog or a similar construct) that knows of all the cloudlets. This is a disadvantage, because this service represents a single point of failure. Furthermore, if the network is large and has many cloudlets, the access to the catalog can become problematic, since many requests may be performed simultaneously, and the catalog files could be extremely large. One main caveat of the cloudlet framework is that it does not work optimally in scenarios with mobile devices. This is because, in order to use the resources of a cloudlet, a node must be connected to an access point (i.e., it must be in the same local network as the cloudlet, or it needs to be connected to the global Internet network to access other cloudlets). Thus, 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 its wireless range. Drop Computing is able to solve this issue, because it uses opportunistic communication for offloading tasks in a node's vicinity.

Finally, the most complex and functional platform that we present here is (coincidentally) also named mCloud [24]. It is a code offloading framework consisting of mobile devices, nearby cloudlets (not in the sense of the platform presented before, but as the services that run at the edge of the network) and public cloud services. The mCloud platform considers offloading through various wireless channels, such as Wi-Fi, 3G, Bluetooth, or Wi-Fi Direct. It employs a multi-criteria offloading decision-making algorithm, which takes into account energy consumption, execution time reduction, resource availability, network conditions, and user preferences. Similarly to Drop Computing, mCloud attempts to offload computations to nearby devices, and then uses the cloudlets at the edge of the network, as well as the actual cloud services (if no other options are available). However, device-to-device communication is again limited to only one hop.

3.4. Scope and Limitations

The main conclusion that can be drawn from the analysis in this section is that Drop Computing manages to address the limitations of existing solutions by employing device-to-device communication for the first layer of task offloading. Thus, not only devices that are in range of the current node can be employed, but also other nodes in the network that may be more resource-capable. Through context-based decisions, Drop Computing can choose the most suitable nodes for helping with the computations. Furthermore, edge devices are viewed as regular nodes that can help with computations, so, when they are accessible, they can be employed by the devices in the network.

The main limitation that Drop Computing can exhibit is that a node may be located in a sparse network, thus rarely meeting other devices that it can offload its computations to. However, when no other peers are present when a node wants to perform a computation, it can go directly to the cloud. If no connection to the cloud exists, then the node must either perform the computation itself, or it can wait until the task can be offloaded. However, this is a limitation that appears in any cloud-based computing system, not necessarily something that is specific to Drop Computing. Furthermore, sending a task to other peers might lead to delays in execution while the current node waits for the results. If the task executors are not chosen well (i.e., they move away from the current node, or cannot or will not perform the task), then a delay is added, until the current node sees that it does not receive the computation results and thus ends up connecting to the cloud. This is an app developer or user-specific decision, because they must decide between cloud costs, battery consumption, and computation duration. These metrics are generally counter-balanced, since low costs and reduced battery consumption imply the usage of neighbor nodes, which might lead to higher computation durations. However, as we show in Section 4, there are situations where using other nodes for computations not only reduces the cloud usage (and costs), but also the duration of the computations.

One other challenge that needs to be addressed in future work is data consistency, which is needed when computations are split in multiple tasks that

are offloaded to various other devices or infrastructures. Moreover, privacy and security still need to be addressed, since we are dealing with (partly) decentralized networks. For incentives, Drop Computing currently uses a tit-for-tat mechanism, where nodes are helped by peers if they help others in turn. Furthermore, nodes that have similar tasks to compute can split the computations among themselves. Thus, each would perform a subtask and then the results would be merged together (which would be in the benefit of all nodes). However, in the future we want Drop Computing to use micro-payment mechanisms for providing utility and motivating additional contributions.

4. Experimental Results

In this section, we present a proof-of-concept implementation of Drop Computing using an opportunistic network emulator, and we describe two test scenarios and analyze their results.

4.1. Drop Computing Implementation

In order to prove that the Drop Computing solution presented in this paper is feasible for real-life use, we have implemented and tested it in MobEmu[2], an opportunistic network emulator that can run a user-created routing or dissemination algorithm on a desired mobility trace or synthetic model. We have added extra functionality to MobEmu which allows us to simulate a cloud system that opportunistic nodes can upload tasks to. Aside from emulating opportunistic contacts, MobEmu can also simulate data transfers between nodes, as well as between mobile devices and the cloud.

For our tests, we considered two situations, as previously described: nodes exclusively use the cloud for performing computations for a mobile application, or nodes also employ other nodes from the crowd in order to avoid having to transfer data to and from the cloud, and to thus decrease the costs by utilizing

[2] `https://github.com/raduciobanu/mobemu`.

less of its resources. Our implementation of Drop Computing has three main components, which we describe below.

At each clock tick, a node performs the following steps:

- if the node currently has no tasks that it needs to execute, it generates some tasks
- if we are dealing with the cloud-only scenario, the node uploads the new tasks to the cloud as soon as they are generated
- if other devices may be employed for task computation:
  - the node picks the first task from its task list, and starts computing it (the task list is sorted by priority, where tasks belonging to the current node and expiring soon are at the beginning of the list)
  - when the task is completed:
    - if it belonged to the node itself, it is marked as finished
    - if it belonged to a different node, it is placed into a list of finished tasks
  - the node uploads its expired tasks (i.e., tasks that the node or other peers could not compute in time) to the cloud
- the node downloads the tasks previously sent to the cloud if they have been completed.

For the opportunistic case, the following actions are performed when two nodes encounter each other (i.e., they are in wireless range):

- they exchange data regarding each other's completed tasks
- if the two nodes are familiar (i.e., are socially connected or they have encountered each other for at least a given number of times), they also exchange information about the completed tasks of other nodes (this way, they can help to opportunistically deliver the computation results to other nodes they may encounter)

- the two nodes check if the tasks each has to execute are balanced (i.e., if the total estimated durations of their computations are roughly equal)
- if the nodes are not balanced, an attempt to balance them is made, taking into account the processing power of each node and the owner of each task
- if the two nodes are familiar and they need to balance their tasks, they simply exchange tasks (i.e., a node that offloads a task will no longer have that task in its computation list)
- if the two nodes are not familiar but they need to balance their tasks, replicas of the tasks are made (so each node keeps its previous tasks and potentially adds some more).

Finally, the cloud behavior is described below:

- the cloud coordinator waits for tasks from nodes and, when it receives a task, it assigns it to a free VM instance
- if all VMs are busy, the coordinator stores the task in a waiting queue, and assigns it to a VM once one is available.

4.2. Test Scenarios

We have tested our proof-of-concept implementation on two scenarios, as shown in Table 1. The first scenario uses the HCMM mobility model [25] to simulate node behavior and interactions. HCMM assumes that nodes in a mobile network are not driven only by the social relationships between them, but also by the attraction of physical locations, so each node is attracted to a home cell according to the social attraction exerted on that node by all nodes that are part of its community. Moreover, the attraction of an external cell is computed based on the relationships with nodes that have their home in that cell. For our tests, we created a scenario with 40 nodes grouped in four communities, with 10 travelers (i.e., nodes that move between communities). We considered a simulation area of 400x400 meters, with nodes moving with speeds between

Configuration	Scenario 1	Scenario 2
Mobility model/trace	HCMM mobility model	UPB 2015 mobility trace (office)
Duration (hours)	24	7
Number of nodes	40	6
Contacts	134034	324
Bluetooth transfer speed	3 MB/s	
Wi-Fi transfer speed	5 MB/s	
Node types	Samsung Galaxy S5, iPhone 5s (equal number)	
Cloud VM instance types	Amazon EC2 t2.small (Intel Xeon, 1 core, up to 3.3 GHz)	
Minimum battery threshold	20%	
Types of tasks	small (1 Mcycle), medium (1000 Mcycles), large (10000 Mcycles)	
Task size	5 MB	
Task generation	1-10 small, 0-10 medium, 0-10 large	

Table 1: Testing parameters.

1.25 and 1.5 m/s. We set the transmission radius of the devices to 10 meters, since we assume that data exchanges between nodes are done on Bluetooth.

For the second testing scenario, we used a real-life mobility trace collected in an office environment in Bucharest between 2014 and 2016. The data collection was performed using an Android application entitled HYCCUPS Tracer[3], which was installed and run by the employees of an IT company. We selected a sample from this trace which contains contacts collected from 6 nodes during a 7-hour window in a workday.

We considered that a node generates workloads that are composed of three types of tasks (small, medium, and large), as shown in Table 1. A workload contains at least one small task, and no more than 10 tasks of each type. When the mobile nodes compute the tasks, they only use a single core. We added this restriction in order to avoid straining a user's device too much in the mobile application that would benefit from the Drop Computing framework, both in terms of QoE, as well as battery consumption. The dimension of a task was set to 5 MB, which we believe to be a reasonable size for a photo, if we are to simulate

[3] `http://hyccups.hpc.pub.ro`.

the scenario we previously presented in Section 3.2. In all our simulations, tasks are sent to the cloud using Wi-Fi (so we assume a node to be connected to an access point) and are exchanged between devices using Bluetooth.

We performed our tests by simulating two types of devices: Samsung Galaxy S5 and iPhone 5s. For each scenario, half of the nodes are Samsung devices, while the other half are iPhones. Based on each of these phones' specifications (as obtained from GSM Arena[4]), we set the duration for computing a megacycle on a core as 0.7 ms for the iPhone 5s devices (since they have 1.3 GHz cores) and as 0.4 ms for the Samsung Galaxy S5 devices (which have 2.5 GHz cores).

For the cloud, we chose and simulated Amazon t2.small instances[5], which have clock speeds of up to 3.3 GHz, so we approximated the duration of a megacycle on a cloud VM to 0.3 ms. Thus, it can be observed that the best way to compute a task, if we do not consider the transfer speed or the cost, would be the cloud. However, through our simulations, we attempt to show that it is cheaper and more efficient to leverage other devices in the crowd for at least part of the computations.

For both scenarios, we vary two sets of simulation parameters. Firstly, for the Drop Computing measurements, we vary the expiration time of the tasks. The lower it is, the quicker the tasks will expire and will be uploaded to the cloud. This might lead to faster completion times, but the cloud utilization (and thus, the costs) will be higher. On the other hand, if the expiration time is higher, there will be more time for the other devices in the network to help compute some tasks, so the cloud will not be used as often, even if the completion duration might increase.

The second simulation parameter we vary for our tests is the maximum number of virtual machine instances that can be obtained in the cloud. Although we assume on-demand pricing for the VMs (i.e., instances are rented when needed and the payment is done per hour), we place a hard cap on the number

[4] `http://www.gsmarena.com`.

[5] `https://aws.amazon.com/ec2/instance-types/`.

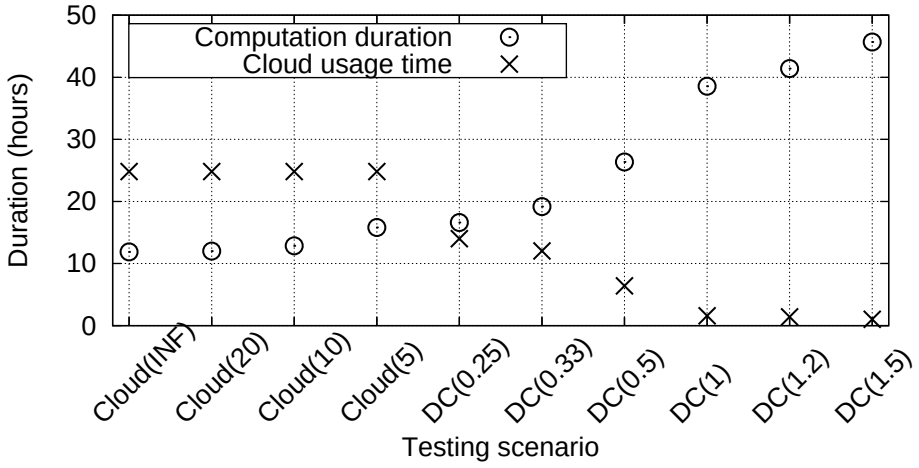


Figure 2: Experimental results for scenario 1.

of VMs that can be rented, and analyze the behavior of cloud computing in this situation.

4.3. Results and Analysis

Figure 2 shows the total computation duration of all the tasks and the total cloud usage time for the first scenario. For the cloud-only experiments (first four data points in Figure 2), we tested with unlimited VM instances, and then with a maximum of 20, 10, or 5 allowed VMs. It can be observed that, as we decrease the number of maximum allowed VM instances, the computation duration increases, whereas the cloud usage time stays the same. For the unlimited VM instances test, the maximum number of VMs used at once was 48.

For Drop Computing, we kept the maximum number of VM instances fixed to 40 (i.e., equal to the number of nodes in the scenario) and we varied the task expiration time. The default timeout (*DC(1)* in Figure 2) was computed as the sum of all tasks generated at once (e.g., if we generate two small tasks, one medium task, and three large tasks, the expiration time is 31002 ms, computed as 2 + 1 * 1000 + 3 * 10000), while for the other Drop Computing tests we multiplied it by the value shown in Figure 2 (e.g., *DC(0.25)* means that the task expiration time for that particular test was a quarter of the default timeout). It

can be seen that, when we increase the task expiration time, the cloud usage time naturally drops (since tasks expire later and thus get the chance to be computed in the crowd) and the computation duration increases. For the *DC(0.25)* case, Figure 2 shows that, when using Drop Computing, the total cloud usage time drops by 44%, which is a decrease of almost 11 hours for a 24-hour scenario. This means that, for a cloud with Linux-based Amazon t2.small instances rented on-demand from the US East (N. Virginia) region, we would have a decrease in cost of 28 cents per day[6]. This might seem like a small value, but if we assume a scenario with hundreds of thousands of users daily, a 44% cost reduction would be extremely significant.

Figure 2 also shows that, for a 44% cost reduction, the total computation duration only increases by 5% when comparing to the scenario where the number of VM instances is capped to 5. This would be an unnoticeable performance degradation for the end-user, and a high cost decrease for the application provider. Furthermore, users could benefit themselves from this drop in costs, because they would be forced to see fewer ads, since the application developers would no longer need to earn as much money from advertising. If we assume that the cloud can scale up in an unlimited fashion, the total computation duration is increased by 39% for the Drop Computing case. This is still a good value, since the percentage of cost reduction is higher than the percentage of performance degradation. If the mobile application accepts higher delays (i.e., it does not need its processing to be necessarily performed instantly), the cloud usage time can be decreased even further. Figure 2 also shows that the cloud usage time can be reduced by 75% if we accept a doubling of the computation time (for the *DC(0.5)* case).

The results obtained for the second scenario (which emulate a real-life situation) are even better, as shown in Figure 3(a). There are situations where Drop Computing not only decreases the costs of the cloud infrastructure by using it less, but the computations are even performed faster due to the help of the

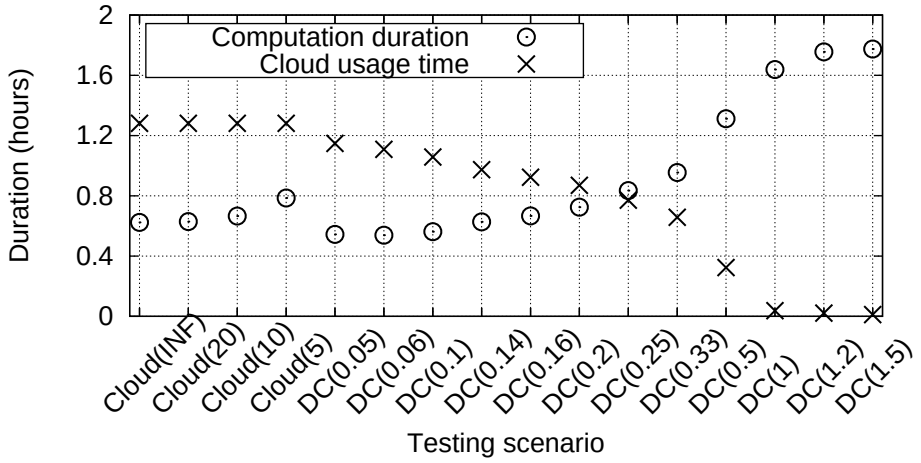
[6] `https://aws.amazon.com/ec2/pricing/on-demand/`.

other nodes in the crowd. If we compare the *Cloud(INF)* scenario (where the number of VM instances in the cloud is unlimited) to the *DC(0.1)* scenario, we can observe that not only is the cloud usage reduced by 18%, but the computation duration is also decreased by 10%. Furthermore, the cloud usage can be reduced by 28% with only a 6% degradation in computation time (as seen for the *DC(0.16)* case), or a reduction of 75% in cloud usage time with a doubling of the computing time (for any applications that allow this).

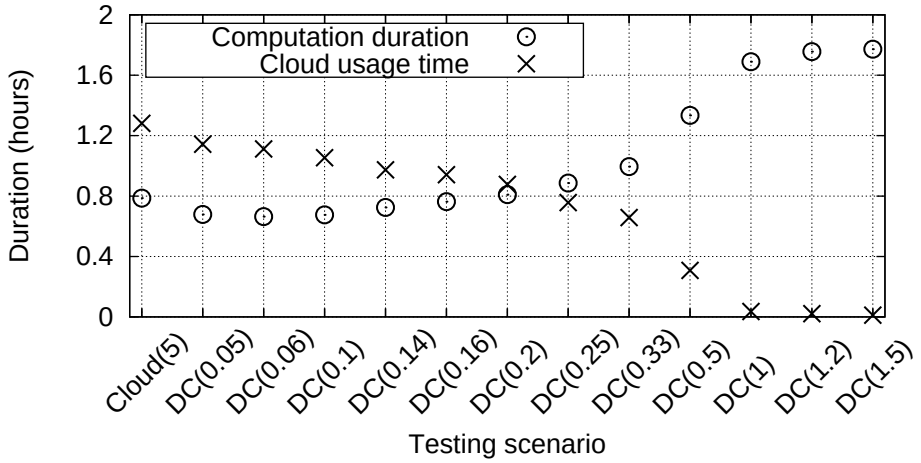
We went one step further and assumed that the maximum number of VM instances allowed is 5 for all situations, and the results can be seen in Figure 3(b). For the *DC(0.16)* case, the results are still better than when using only cloud, both from the standpoint of cloud usage, as well as computation duration. The cloud usage is reduced by 27%, while the computation time is reduced by 3%. These results show that, even if we limit the number of VM instances for the cloud-only case (in order to reduce the costs), employing the crowd for opportunistic communication still brings clear benefits, both in terms of costs for the app developer, as well as in terms of QoE for the end-user.

In order to highlight the cost reduction achieved by employing Drop Computing, we show in Figure 4 the cost for several key cases from scenario 2. We computed the total cost by analyzing the maximum number of VM instances employed simultaneously for each of the 7 hours of the simulation, and multiplying this number by \$0.023, which is the price for a t2.small VM instance[7]. As expected, when only the cloud is used and there is no cap for the number of VM instances allowed, the cost is the highest. For the use case where only the cloud is used and the number of VMs is limited to 5, the cost is reduced by 77%, but (as shown in Figure 3(a)) the computation duration also increases. We also showed the cost for the best use case in terms of computation duration and cloud usage time, *DC(0.05)*. It can be observed that the cost is reduced by 20% when compared to the default case. Finally, we also highlighted the minimum cost that we obtained through our simulations, for *DC(1.5)*. In this

[7] `https://aws.amazon.com/ec2/pricing/on-demand/`.



(a) Default



(b) Limited number of cloud VMs

Figure 3: Experimental results for scenario 2.

case, there is a cost reduction of 85%, but the total computation duration also grows accordingly.

Since Drop Computing employs other peers located nearby for helping with computations, it is very important that the load is balanced equally between nodes. In other words, there should not be some nodes that perform most of the work for others, since fairness is required. This is also important because

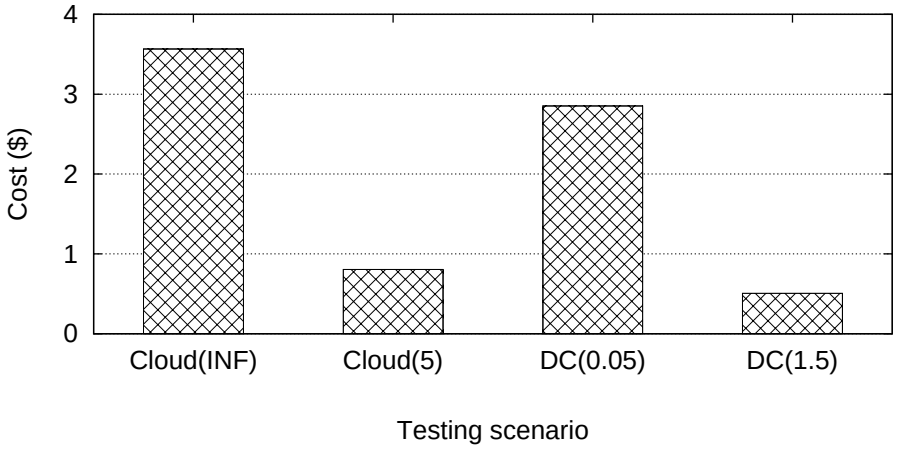
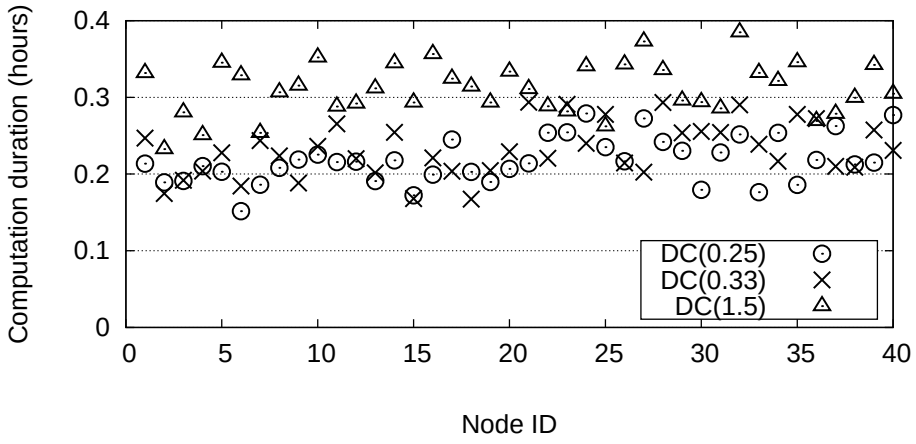


Figure 4: Cloud usage costs for scenario 2.

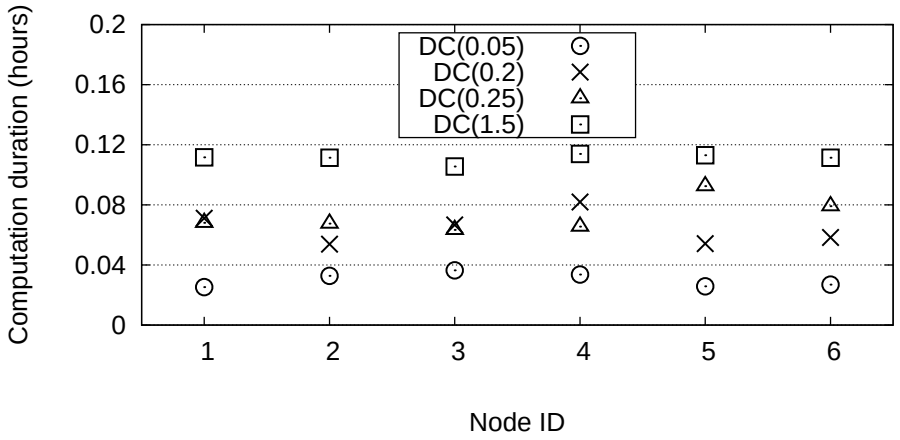
Scenario	Scenario 1			Scenario 2			
Test case	DC(0.25)	DC(0.33)	DC(1.5)	DC(0.05)	DC(0.2)	DC(0.25)	DC(1.5)
Minimum	0.152	0.167	0.233	0.025	0.054	0.063	0.106
Maximum	0.279	0.294	0.385	0.036	0.082	0.093	0.114
Average	0.218	0.231	0.311	0.030	0.064	0.073	0.111
Standard deviation	0.030	0.035	0.034	0.005	0.011	0.011	0.003

Table 2: Load balancing (data is in hours).

people might not be willing to participate in opportunistic computations if their device is used for the benefit of others, while not getting anything in return. For this reason, in Figure 5 we show the way load is balanced for the two scenarios presented in Table 1. For each scenario, we selected the test cases that we considered the most relevant (i.e., the ones with the fastest computation duration, the lowest cloud usage time, and several in-between). Furthermore, Table 2 presents the minimum and maximum values for each test case, along with the average and the standard deviation. It can be observed that, for all the situations analyzed, the load is balanced well: the standard deviation is low, never amounting to more than 18% of the mean value.



(a) Scenario 1



(b) Scenario 2

Figure 5: Load balancing between nodes.

5. Conclusions and Future Work

In this paper, we have proposed a shift towards a decentralized computing model for supporting collaborative computation and data storage in clouds composed of mobile/wearable devices, in the shape of the Drop Computing paradigm. It moves some of the necessity of using clouds towards the crowd of

mobile devices, by leveraging human mobility and using contacts as opportunities. We have shown that Drop Computing has applicability in real-life, by simulating a scenario where a mobile app uses not only cloud-based services for data computations, but also devices located nearby with available resources. The results showed that the Drop Computing paradigm has a clear applicability in real-life situations. We saw that it can drastically reduce the costs of employing a cloud platform without affecting the user experience much, in some cases even decreasing the total computation duration. We believe that Drop Computing can be very useful for application developers, because it helps them move some of the costs from the cloud to the crowd, without having an effect on the end-product.

Currently, Drop Computing is at the stage of proposal and proof-of-concept implementation, so there are still challenges that need to be addressed in the future. One of them is data consistency, which can arise when a task or data item is split into smaller pieces and spread in the network. So, novel groundbreaking approaches that have been proven to work (e.g., BitTorrent) must be adapted to current needs. Furthermore, privacy and security are handled through the decentralization of data and other resources offered by devices. We envision Drop Computing to be served by micro-payment mechanisms designed to provide utility to participants to motivate additional contributions (monetary and information incentives). New methods and techniques must be developed to better motivate people to collaborate. Also, financial models for supporting these types of applications must be developed and evaluated. "Human factors" need to be used to help humans better understand their contributions.

References

[1] T.-C. Huang, C.-K. Shieh, N. Chilamkurti, M.-F. Tsai, S. Rho, Architecture for speeding up program execution with cloud technology, Journal of Supercomputing 72 (9) (2016) 3601–3618. doi:10.1007/

s11227-016-1715-x.
URL http://dx.doi.org/10.1007/s11227-016-1715-x

[2] L. Valerio, A. Passarella, M. Conti, Hypothesis transfer learning for efficient data computing in smart cities environments, in: Smart Computing (SMARTCOMP), 2016 IEEE International Conference on, IEEE, 2016, pp. 1–8.

[3] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: Vision and challenges, SIGCOMM Comput. Commun. Rev. 45 (5) (2015) 37–42. doi:10.1145/2831347.2831354.
URL http://doi.acm.org/10.1145/2831347.2831354

[4] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12, ACM, New York, NY, USA, 2012, pp. 13–16. doi:10.1145/2342509.2342513.
URL http://doi.acm.org/10.1145/2342509.2342513

[5] N. Kumar, R. Iqbal, N. Chilamkurti, Capacity and load-aware service discovery with service selection in peer-to-peer grids, Future Generation Computer Systems 28 (7) (2012) 1090–1099. doi:10.1016/j.future.2011.11.008.
URL http://dx.doi.org/10.1016/j.future.2011.11.008

[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing - a key technology towards 5g, ETSI White Paper 11.

[7] A. Ahmed, E. Ahmed, A survey on mobile edge computing, in: Intelligent Systems and Control (ISCO), 2016 10th International Conference on, IEEE, 2016, pp. 1–8.

[8] B. Garvelink, Mobile edge computing: a building block for 5g, Telecompaper (subscription).

URL https://www.telecompaper.com/background/mobile-edge-computing-a-building-block-for-5g--1092281

[9] M. T. Beck, M. Werner, S. Feld, S. Schimper, Mobile edge computing: A taxonomy, in: Proc. of the Sixth International Conference on Advances in Future Internet, Citeseer, 2014.

[10] I. Stojmenovic, S. Wen, The fog computing paradigm: Scenarios and security issues, in: Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on, IEEE, 2014, pp. 1–8.

[11] L. M. Vaquero, L. Rodero-Merino, Finding your way in the fog: Towards a comprehensive definition of fog computing, ACM SIGCOMM Computer Communication Review 44 (5) (2014) 27–32.

[12] S. Yi, C. Li, Q. Li, A survey of fog computing: concepts, applications and issues, in: Proceedings of the 2015 Workshop on Mobile Big Data, ACM, 2015, pp. 37–42.

[13] J. M. Batalla, P. Krawiec, C. X. Mavromoustakis, G. Mastorakis, N. Chilamkurti, D. Negru, J. Bruneau-Queyreix, E. Borcoci, Efficient media streaming with collaborative terminals for the smart city environment, IEEE Communications Magazine 55 (1) (2017) 98–104. doi:10.1109/MCOM.2017.1600225CM.

[14] N. Kayastha, D. Niyato, P. Wang, E. Hossain, Applications, architectures, and protocol design issues for mobile social networks: A survey, Proceedings of the IEEE 99 (12) (2011) 2130–2158.

[15] J. F. Pérez, G. Casale, S. Pacheco-Sanchez, Estimating computational requirements in multi-threaded applications, IEEE Transactions on Software Engineering 41 (3) (2015) 264–278.

[16] R.-C. Marin, C. Dobre, F. Xhafa, Exploring predictability in mobile interaction., in: EIDWT, 2012, pp. 133–139.

URL http://dblp.uni-trier.de/db/conf/eidwt/eidwt2012.html#MarinDX12

[17] Y. Wang, I.-R. Chen, D.-C. Wang, A survey of mobile cloud computing applications: Perspectives and challenges, Wirel. Pers. Commun. 80 (4) (2015) 1607–1623. doi:10.1007/s11277-014-2102-7.
URL http://dx.doi.org/10.1007/s11277-014-2102-7

[18] R. Kemp, N. Palmer, T. Kielmann, H. Bal, Cuckoo: a computation offloading framework for smartphones, in: International Conference on Mobile Computing, Applications, and Services, Springer, 2010, pp. 59–79.

[19] N. Kumar, R. S. Bali, R. Iqbal, N. Chilamkurti, S. Rho, Optimized clustering for data dissemination using stochastic coalition game in vehicular cyber-physical systems, Journal of Supercomputing 71 (9) (2015) 3258–3287. doi:10.1007/s11227-015-1436-6.
URL http://dx.doi.org/10.1007/s11227-015-1436-6

[20] G. Huerta-Canepa, D. Lee, A virtual cloud computing provider for mobile devices, in: Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, MCS '10, ACM, New York, NY, USA, 2010, pp. 6:1–6:5. doi:10.1145/1810931.1810937.
URL http://doi.acm.org/10.1145/1810931.1810937

[21] N. Fernando, S. W. Loke, W. Rahayu, Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing, in: Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing, UCC '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 281–286. doi:10.1109/UCC.2011.45.
URL http://dx.doi.org/10.1109/UCC.2011.45

[22] E. Miluzzo, R. Cáceres, Y.-F. Chen, Vision: Mclouds - computing on clouds of mobile devices, in: Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services, MCS '12, ACM, New York, NY, USA,

2012, pp. 9–14. doi:10.1145/2307849.2307854.
URL http://doi.acm.org/10.1145/2307849.2307854

[23] T. Verbelen, P. Simoens, F. De Turck, B. Dhoedt, Cloudlets: Bringing the cloud to the mobile user, in: Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services, MCS '12, ACM, New York, NY, USA, 2012, pp. 29–36. doi:10.1145/2307849.2307858.
URL http://doi.acm.org/10.1145/2307849.2307858

[24] B. Zhou, A. V. Dastjerdi, R. Calheiros, S. Srirama, R. Buyya, mcloud: A context-aware offloading framework for heterogeneous mobile cloud, IEEE Transactions on Services Computing PP (99) (2016) 1–1. doi:10.1109/TSC.2015.2511002.

[25] C. Boldrini, A. Passarella, Hcmm: Modelling spatial and temporal properties of human mobility driven by users' social relationships, Comput. Commun. 33 (9) (2010) 1056–1074. doi:10.1016/j.comcom.2010.01.013.
URL http://dx.doi.org/10.1016/j.comcom.2010.01.013