Device to Device Collaboration for Mobile Clouds in Drop Computing

Radu-Corneliu Marin, Alexandru Gherghina-Pestrea,
Alexandru Florin Robert Timisica, Radu-Ioan Ciobanu, Ciprian Dobre
Faculty of Automatic Control and Computers
University Politehnica of Bucharest
Emails: {radu.marin, alexandru.gherghina, robert.timisica}@smartrdi.net, {radu.ciobanu, ciprian.dobre}@cs.pub.ro

***Abstract*—The large number of mobile devices existing nowadays has led to the evolution of mobile cloud computing towards bringing data and computations closer to the nodes. This has manifested first in the shape of fog and edge computing, where an additional communication and processing layer is added at the edge of the network. However, the fast adoption of the Internet of Things has shown the limitations of even this model, so the focus now is moving towards another layer that is one level below: the ad hoc network composed of the mobile devices themselves.**

One paradigm based on this model is Drop Computing, where nodes that need to do some computations first attempt to process them through the help of neighbor devices using close-range communication (such as Wi-Fi Direct or Bluetooth), and only then do they attempt to contact the fog/edge nodes or the cloud itself. In this paper, we propose an Android implementation of the device-to-device layer of Drop Computing. On top of this implementation, we present an application that creates a video collage from multiple photos using *ffmpeg* with the help of neighboring nodes through close-range communication using the HYCCUPS and Google Nearby frameworks. Through experiments on four Android devices, we show that our implementation can drastically decrease CPU usage per device, which in turn increases the overall quality of experience for Android users. Furthermore, the total battery consumption is lowered, since nodes have less computations to perform and the CPU cores spend less time in higher frequencies.

***Index Terms*—mobile, cloud, edge, opportunistic, Android**

I. Introduction

Mobile cloud computing (MCC) has appeared as a solution to the challenges faced by mobile devices (especially in terms of resources such as battery life, storage, or bandwidth) due to their limited hardware make-up and high mobility. However, MCC only sees the mobile devices as clients that always have to contact the cloud to obtain services. Because the large-scale adoption of the Internet of Things has drastically increased the number of devices that need to connect to the cloud, alternative means of providing services to mobile users have been proposed, such as fog and edge computing. Their aim is to bring resources at the edge of the network, closer to the mobile devices, but we argue that even this is not enough nowadays. One possible way to improve this situation is to add another layer to the fog/edge architecture, namely the mobile network layer, where devices are able to communicate with each other and collaborate to solve common goals. One paradigm that has this behavior is Drop Computing [1], which uses the social ties between humans to form an ad hoc network of mobile devices as the lowest layer. At this level, devices will opportunistically ask other nodes for help through close-range communication, and only if this help cannot be offered will requests to edge/fog or cloud nodes be made.

Thus, in this paper we aim to demonstrate the feasibility of spreading a computation in the ad hoc network composed of mobile devices through cyber foraging [2]. For this reason, we propose and present an Android application that uses *ffmpeg* to create a video collage from a group of pictures. The app can scatter the necessary computations to a mobile cloud created of other smartphones, so the processing is performed in parallel on multiple devices. Through extensive testing with four Android phones, we show that collaborations even at the lowest level of Drop Computing are able to improve various metrics in the mobile network (such as battery consumption or quality of experience), thus highlighting the benefits of the Drop Computing paradigm.

The rest of this paper is structured as follows. Section II presents related work in the area of mobile cloud computing and computation offloading, while Sect. III describes the Drop Computing paradigm and its functionality. Section IV proposes our Android application built on top of the Drop Computing framework, and Sect. V contains a thorough analysis of its performance on four Android devices. Finally, Sect. VI presents our conclusions and future work.

II. Related Work

The area of MCC (and of distributing computations across the ad hoc mobile network) has gained traction in recent years, thanks to the growth of mobile communication paradigms such as opportunistic computing [3]. Its main advantages include extending the battery lifetime by moving computation away from mobile devices, better usage of data storage capacity and of available processing power, improving the reliability and availability of mobile apps, while also offering cloud computing benefits like dynamic provisioning and scalability [4].

One example of a mobile cloud computing framework is proposed in [5], where users with similar goals or tasks collaborate with each other. When a task is generated, it is spread to the nodes in a virtual cloud, which compute

This research is supported by NETIO project Tel-MonAer and projects SPERO (PN-III-P2-2.1-SOL-2016-03-0046, 3Sol/2017) and ROBIN (PN-III-P1-1.2-PCCDI-2017-0734).

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

their parts and then their computations are merged into the final results that the nodes share. The main drawback of this solution is that it does not account for node mobility and for device heterogeneity.

Two interesting complementary solutions are Serendipity [6] and COSMOS [7]. The former deals with the case when mobile devices do not have access to a cloud backend or to edge devices at all, so they can only use each other for computation offloading. They use a PNP job model, which defines jobs as having a pre-processing part, a number of parallel tasks, and a post-processing part. The first and the third steps are executed by the job's owner, whereas the parallel tasks can be offloaded to nearby devices. On the other hand, COSMOS offers computation as a service by sharing cloud resources to mobile devices, which offload their tasks based on network connectivity and task characteristics. The Drop Computing framework that we propose offers computation and data offloading on the cloud, on edge devices, or on other mobile devices, combining the benefits of Serendipity and COSMOS. Furthermore, task offloading on other mobile nodes can be done over multiple hops, in an opportunistic fashion.

mCloud [8] is a code offloading framework composed of mobile nodes, nearby cloudlets and public cloud services. The 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.

III. Drop Computing

The idea behind mobile cloud computing is to move computations away from the mobile devices, since they face many challenges in terms of battery life, storage, bandwidth, as well as being affected by the high degree of mobility. These challenges have the potential to significantly affect the quality of service and experience perceived by the users, which is why more powerful nodes are required. Thanks to mobile cloud computing, a rapid provisioning for mobile applications can be obtained with minimal management efforts [4]. However, since the advent of the Internet of Things (IoT) and the high increase in the number of mobile devices that require cloud services, the classic cloud model has become insufficient. This is the reason for the edge and fog computing paradigms, which aim to bring computations even closer to the devices, in the form of services hosted at the edge of networks in servers, set-top-boxes or even access points [9], [10].

Through the Drop Computing paradigm [1], [11], we argue that the data and computations should not be moved too far away from the mobile devices themselves, in some cases not even as far as the edge nodes. We believe this because, when a large number of small devices need to communicate in general, they all need to send requests to the cloud or to the edge, receive replies, and then process them.

Drop Computing is based on the notion that human-carried mobile device interactions are governed by human social

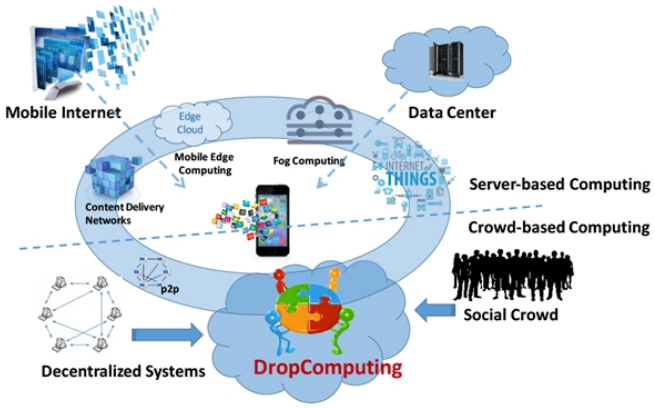


Fig. 1. Drop Computing architecture [1].

ties [12]. These relationships can positively affect communication, by helping users collaborate with the purpose of conserving energy on each mobile device. As online social interaction relationships among people (through their devices) are broadened and enhanced, users can passively cooperate through their connections, leading to the enabling of a significant minimization of the computational load and the energy consumption levels. In order to bypass the constrains of device-to-device communication, users could exploit social ties discovered from mobile device interactions, leveraging other nodes for computation or data offloading [13], [14].

Drop Computing is an attempt to shift mobile communication towards a decentralized model, as shown in Fig. 1. It runs over decentralized mobile networks formed between mobile nodes such as smartphones (or other small devices) and the edge devices that are part of the fog/edge cloud layer. One level above, the cloud servers act as a third option in this multi-layered architecture. Thus, if a user has a mobile application that requires many computations, the current cloud-based behavior would assume that the device sends the computations to the cloud, waits for them to be processed, and then receives the reply. By moving computations closer to the mobile node, the Drop Computing framework allows devices to only go as far as the edge of the network, where specialzed fog/edge devices are able to perform the computations and deliver the response much faster. Furthermore, they are even able to cache their results (or other kinds of useful data), which can increase the performance even more. This is obtained thanks to the lower number of hops required to get to the processing component, the locality of data, as well as faster failover.

Further lower than this, there is the mobile layer, composed of other nodes co-located with the user device. They allow fast communication speeds through close-range protocols such as Bluetooth or Wi-Fi Direct, basically extending the capabilities of a single device beyond the technology barrier of the local hardware. As previously shown and demonstrated [1] (in simulation), Drop Computing allows mobile nodes to transparently access resources available in the mobile opportunistic network, where the content is much closer to the users and computations

can be offloaded. Applications are decomposed into tasks that can be executed locally, or offloaded closer to their data or to more powerful nodes [15].

Drop Computing's decentralized vision places people at the heart of the computation-coordination model [1], where social connections between them are used to select nearby nodes that are able to help a peer through close collaborations. Incentive mechanisms need to be in place, in order to motivate devices to participate in the Drop Computing network and help others who can later help in turn. A solution for incentivizing mobile users to help others with computation offloading is FlopCoin [16], which proposes a distributed incentive scheme that uses a blockchain-based virtual currency. The blockhain is deployed in cloudlets, and each mobile user has a wallet where FlopCoins are added (through transactions) when the device offloads for the benefit of others, and removed when it performs an offload itself after an auction.

In this paper, we attempt to test the feasibility of the Drop Computing paradigm at the lowest level. More specifically, we wish to test whether it makes sense to employ close-range nodes in performing intensive computations with a common goal, in terms of battery consumption, quality of experience (QoE) and computation time.

IV. Proposed Solution

In order to highlight the benefits of Drop Computing (i.e., of adding an extra layer composed of mobile devices below the edge layer), we implemented an offloading framework in Android, based on HYCCUPS [17]. For device-to-device communication, our solution uses AllJoyn[1] and Google Nearby[2] to create an ad-hoc network between Android smartphones that are in close proximity of each other. AllJoyn communication is performed over Wi-Fi when the devices are connected to the same access point, whereas Google Nearby communicates over Bluetooth Low Energy (BLE) and Wi-Fi Direct, depending on the capabilities of the communicating devices and the environment conditions.

On top of the communication framework built over AllJoyn and Google Nearby, we also implemented an Android application that acts as a particular use case for mobile cloud computing in general, and Drop Computing in particular (aside from this scenarios, other Drop Computing use cases may include speech or face recognition, translation, or graphics processing). The purpose of our application is to allow mobile users to create a video collage from a list of photos. A similar feature is present in the Google Photos application, but all the computations are performed in the cloud. In our case, we want to address situations where the cloud is not available (e.g., the device does not have Internet access) or the user wants to avoid employing it. For this reason, the default behavior of the video collage application is to run all computations locally, on the user's mobile devices. For creating the video clip, our application uses the *ffmpeg* library[3].

[1]https://openconnectivity.org/developer/reference-implementation/alljoyn
[2]https://developers.google.com/nearby/
[3]https://www.ffmpeg.org

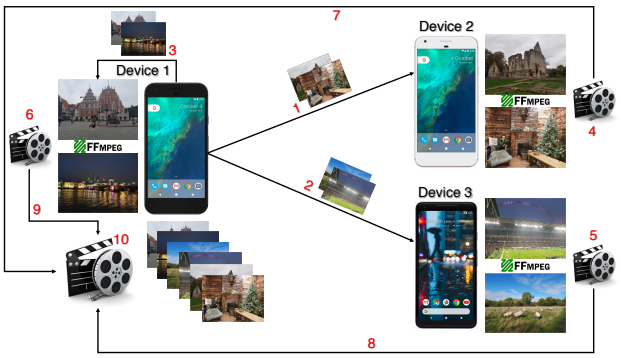


Fig. 2. Functionality of the proposed application.

However, as discussed in Sect. I, a device might not always have the capabilities of performing certain computations itself, or might want to offload some of them. Thus, our proposed application also has an offloading component implemented over the HYCCUPS and Google Nearby-based communication framework. Our app works as shown in Fig. 2. Let us assume that the user of device 1 has six pictures that need to be collated into a video clip. At the first step, the app performs a scan, looking for devices in close range (using Wi-Fi on AllJoyn, or BLE and Wi-Fi Direct on Google Nearby). In the example shown in Fig. 2, two other devices (2 and 3) are found. Thus, the six pictures are scattered into groups of two (in order), and are sent to devices 2 and 3 (shown at points 1 and 2 in Fig. 2), while the master (device 0) also keeps a group of two photos to process itself (point 3 in Fig. 2).

After the photos have been spread evenly, each device uses *ffmpeg* to process its own group, as shown at points 4, 5 and 6 in Fig. 2. Once this is done, the resulting collages are sent back to the master node (points 7, 8 and 9), which unites the three videos into a single file. The main goal of this application (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.

A similar solution was proposed by Chatzopoulos et al. [18], who implemented an Android application that is able to perform video compression with *ffmpeg* by splitting the computations on multiple mobile devices, and then tested it on two devices. In this paper, we extend the experiments to four (newer) Android devices, while presenting another potential use case for computation offloading in mobile scenarios. Furthermore, we plan on augmenting our solution with edge and cloud support.

V. Evaluation

For evaluation of the Drop Computing offloading platform, we deployed the video collage application presented in Sect. IV on four mobile devices, as shown in Table I. We tested the app with 10 photos of 10 megapixels each, first by collating the photos on a single device (for the baseline run), then on two, three and four devices, while alternating the master (i.e., the node that scatters the photos and then gathers and merges

TABLE I
TESTING DEVICES.

ID	Model	Android	CPU
1	Google Pixel XL	9	Quad-core (2x2.15 GHz Kryo & 2x1.6 GHz Kryo)
2	Google Pixel XL	9	Quad-core (2x2.15 GHz Kryo & 2x1.6 GHz Kryo)
3	Google Pixel 2	8.1	Octa-core (4x2.35 GHz Kryo & 4x1.9 GHz Kryo)
4	Google Pixel 2 XL	8.1	Octa-core (4x2.35 GHz Kryo & 4x1.9 GHz Kryo)

TABLE II
EXPERIMENTAL RESULTS OBTAINED WHEN RUNNING THE VIDEO COLLAGE APP ON A SINGLE DEVICE. THE DATA FROM THIS TABLE WILL BE USED AS BASELINE FOR ALL THE OTHER EXPERIMENTS.

ID	Load (%)	Mcycles	CPUs	Mem (MB)	Time (s)	Bat (%)
1	88.29	6081	0.13	152.24	26.08	0.1
2	87.03	5580	0.11	131.44	25.67	0.09
3	48.06	2110	0.06	91.76	16.12	0.08
4	52.59	2458	0.07	126.96	15.51	0.07

the results). Using Android Battery Historian[4], we measured CPU load, number of megacycles consumed, number of CPUs used for processing, RAM, duration and battery consumption.

Table II shows the baseline run for each of the four Android devices deployed. It can be observed that, for the two Google Pixel XL devices (i.e., with IDs 1 and 2), the processing of the collage takes about 26 seconds, keeping the CPU in 88% load and consuming approximately 10% of the device's battery. On the other hand, since the two Pixel 2 devices (normal and XL) employ newer technology and have eight cores each with higher clock speeds, the computation duration is considerably lower than for the Pixel XL phones (16 seconds as opposed to 26), while also keeping the CPU load hovering around 50%. Since the load is lower, the battery consumption is naturally smaller, the collage processing taking 8% for the Pixel 2 and 7% for the Pixel 2 XL (which has a bigger battery).

Next, Table III shows the results of applying the solution presented in Sect. IV to a mobile cloud composed of two devices. As previously said, one device acts as the master that performs a scatter-gather of the 10 photos to other devices in range and to itself. In this case, each of the two devices gets 5 photos, and we chose to perform two tests, one with two devices of the same kind (the two Pixel XLs) and one with two devices with different specs (the Pixel 2 and the Pixel 2 XL), while alternating the master inside the mobile cloud. The aggregated results for total computation time, battery consumption and CPU load can also be observed in Fig. 3.

Table III shows several interesting things. Firstly, it can be observed that the CPU load per device decreases considerably for all test cases, when compared to the single-device case. For the Pixel XL mobile cloud, the per-device load has a maximum of 70%, whereas a single Pixel XL collating the 10 photos can have the CPU as much as 88% loaded. One positive effect of this is that the quality of experience (QoE) for people using the mobile device while it is running the collage app will

[4]https://developer.android.com/studio/profile/battery-historian

TABLE III
EXPERIMENTAL RESULTS OBTAINED WHEN RUNNING THE VIDEO COLLAGE APP ON TWO DEVICES. AN ASTERISK NEXT TO A DEVICE'S ID SPECIFIES THAT IT WAS THE MASTER FOR THAT PARTICULAR USE CASE.

ID	Load (%)	Mcycles	CPUs	Mem (MB)	Time (s)	Bat (%)
*1	50.8	12109	0.19	144.72	32.78	0.03
2	70.96	62939	1.17	121.2	28.8	0.04
1	66.95	61098	1.31	153.71	25.16	0.05
*2	55.66	11685	0.21	167.14	29.84	0.04
*3	23.5	6871	0.14	85.74	26.39	0.03
4	28.78	55823	1.14	103.14	22.61	0.03
3	27.81	51595	1.12	84.57	22.74	0.03
*4	25.28	7053	0.15	125.56	25.81	0.04

be better, since Android behavior will be affected much less by the processing performed in the background. Furthermore, another positive side effect of lower CPU load is that the overall battery consumption of the mobile cloud is lower than the battery consumption of a single device performing the computations all by itself. For example, a single Pixel XL consumes 10% of its battery collating the photos (as shown in Table II), while two Pixel XLs only used up a maximum of 9% in total (5% on the worker and 4% on the master, as seen in the second test from Table III). This shows that, if we have a Drop Computing mobile cloud where devices can collaborate and help each other with their computations (in a "tit for tat" fashion), then the overall power consumption of the mobile network will be lower, even if devices only collaborate in groups of two. Similar conclusions can be drawn from Table III for the experiments with devices 3 and 4, which are different from each other.

One other interesting observation that can be made when analyzing Table III is that the number of megacycles for the non-master device in the mobile cloud is much higher than for the master device or than for the baseline run, although other parameters do not seem to be affected by this. However, those extra cycles are actually NOP cycles that occur when performing the close-range communication between the devices in the mobile cloud. The only metric that is not improved by employing Drop Computing is the overall duration of performing the collage, because of the extra time required by communication and synchronization between devices (our solution increases the computation time by a maximum of 27% when dealing with two identical devices, and 70% for the mobile cloud composed of the Pixel 2 and the Pixel 2 XL). However, this is something that becomes less and less significant as the single-device computation duration increases, and we would argue that the idea of mobile cloud applies mostly to delay-tolerant computations, where a slightly higher latency is not a dealbreaker. Nonetheless, we are working on improving the communication overhead as much as possible.

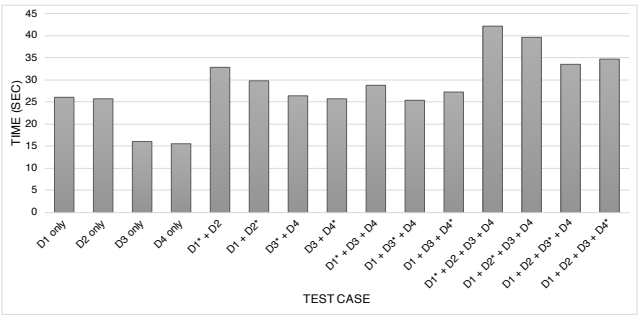
In order to highlight the impact of the device-to-device communication, we have also measured statistics only for the raw *ffmpeg* processing when running the proposed application on a two-node cloud with devices 3 and 4. Thus, Table IV shows that the actual video creation process only takes about

TABLE IV
EXPERIMENTAL RESULTS OBTAINED WHEN RUNNING THE VIDEO COLLAGE APP ON TWO DEVICES, WHILE ONLY LOOKING AT THE RAW FFMPEG PROCESSING. AN ASTERISK NEXT TO A DEVICE'S ID SPECIFIES THAT IT WAS THE MASTER FOR THAT PARTICULAR USE CASE.

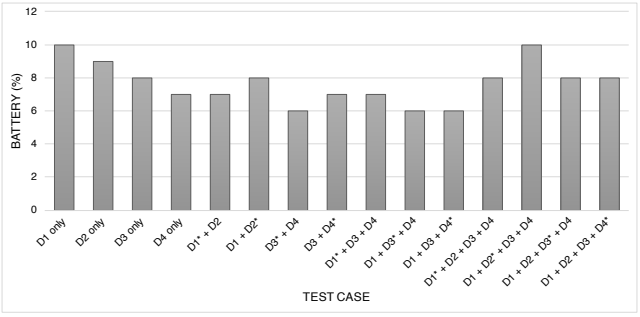
ID	Load (%)	Mcycles	CPUs	Mem (MB)	Time (s)	Bat (%)
*3	39.22	1087	0.08	100.41	8.47	0.04
4	37.09	1237	0.07	131.95	8.93	0.03
3	34.81	1387	0.09	85.92	8.81	0.03
*4	36.33	1404	0.09	128.97	9.63	0.03

TABLE V
EXPERIMENTAL RESULTS OBTAINED WHEN RUNNING THE VIDEO COLLAGE APP ON THREE DEVICES. AN ASTERISK NEXT TO A DEVICE'S ID SPECIFIES THAT IT WAS THE MASTER FOR THAT PARTICULAR USE CASE.

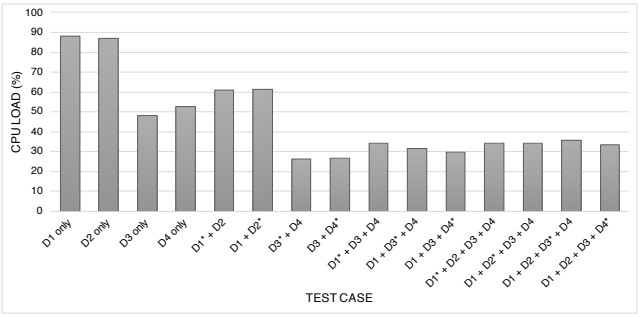
ID	Load (%)	Mcycles	CPUs	Mem (MB)	Time (s)	Bat (%)
*1	55.77	14698	0.26	178.18	28.72	0.04
3	26.51	29451	0.98	84.44	13.95	0.02
4	20.73	29852	0.58	93.36	23.99	0.01
1	47.78	29406	0.78	145.97	23.01	0.01
*3	24.2	7613	0.16	83.36	25.46	0.03
4	22.98	31789	0.83	93.38	18.51	0.02
1	42.8	31570	0.69	168.09	24.61	0.02
3	22.95	28816	0.75	84.88	18.31	0.02
*4	23.56	7796	0.15	127.37	27.33	0.02



(a) Computation time



(b) Battery consumption



(c) CPU load

Fig. 3. Computation offloading experimental results. An asterisk next to a device's ID specifies that it was the master for that particular use case. Computation time was calculated as the maximum time between the devices in the mobile cloud (since the master always takes longer, because it has to gather the partial video files back and merge them). The battery consumption was calculated as the sum between the percentage of battery consumed on all devices in the mobile cloud, whereas the load was computed as the average of CPU loads for the collaborating nodes.

36% of the total processing time, the rest being taken up by the communication. This further highlights our opinion that mobile cloud computing is useful when there is a lot of data to process, but without the need of much communication between nodes. Furthermore, the CPU load is also considerably lower when we account only for the video creation, and so is the number of cycles.

Table V shows the results obtained when running the Drop Computing-based collage application on three different devices (i.e., we only used one of the Pixel XLs). When compared to the baseline runs in Table II, the results (also presented in a graphical and more intuitive fashion in Fig. 3), are similar to ones obtained with the two-device cloud. Namely, the per-device CPU load decreases drastically, allowing for a much better user experience on the Android phones, while also leading to a lower overall battery consumption (since the CPUs spend less time in high frequencies and do not overheat so quickly). An interesting observation that can be made for the three-device scenario is that, because there are some more powerful devices in the mobile cloud (i.e., the Pixel 2 and the Pixel 2 XL), lower-range devices will benefit from a higher computation power. For example, when device 3 is the master, the overall computation time is lower than when device 1 performs the computations by itself.

Finally, Table VI shows the results for a four-node Drop Computing cloud. Again, it can be observed that the per-device CPU load and the overall battery consumption are decreased, leading to a better QoE and longer device lifetime. The disadvantage here (as seen better in Fig. 3) is that the overall computation time increases, because the communication overhead outweighs the computation duration. Thus, the conclusion that we draw here is that, when deciding how many devices should a computation be scattered on, care must be taken in analyzing its characteristics and striking a balance between communication overhead and computation duration. Furthermore, we have taken a simplified approach where the master device first sends all the photos to be collated to all the other devices, and then they each start their computations. However, we plan to improve this by allowing the mobile nodes to start collating photos even when not all of them have been received, so we have a higher degree of parallelism. In conclusion, the main takeaway from this section is that grouping multiple devices into a mobile cloud has significant benefits that encourage us to move further with the Drop Computing framework.

TABLE VI
EXPERIMENTAL RESULTS OBTAINED WHEN RUNNING THE VIDEO COLLAGE APP ON FOUR DEVICES. AN ASTERISK NEXT TO A DEVICE'S ID SPECIFIES THAT IT WAS THE MASTER FOR THAT PARTICULAR USE CASE.

ID	Load (%)	Mcycles	CPUs	Mem (MB)	Time (s)	Bat (%)
*1	48.93	16068	0.21	168.99	42.27	0.04
2	56.6	34316	0.8	111.82	22.34	0.02
3	14.59	19495	0.24	73.83	38.19	0.01
4	16.1	18063	0.3	127.03	28.62	0.01
1	53.85	35151	0.82	107.11	22.45	0.03
*2	51.47	16625	0.24	168.45	39.63	0.04
3	15.05	19479	0.27	72.4	35.67	0.02
4	16.03	17969	0.33	99.64	26.08	0.01
1	48.54	19562	0.34	116.99	30.13	0.02
2	55.69	31968	1.03	141.8	17.17	0.03
*3	20.42	9918	0.16	90.95	33.62	0.02
4	17.91	18371	0.42	127.11	20.93	0.01
1	39.53	21197	0.41	109.54	26.68	0.02
2	57.46	34361	0.79	116.45	23.2	0.03
3	14.12	16961	0.26	74.28	31.65	0.01
*4	21.98	9953	0.16	126.79	34.81	0.02

VI. CONCLUSIONS AND FUTURE WORK

We presented a mobile cloud computing framework entitled Drop Computing, where mobile devices attempt to offload computations to neighboring nodes through close-range protocols. If this is not possible, nodes at the edge of the network are employed, while the cloud acts as the third option. We implemented the first layer of this framework on Android, and we built an application on top of it to highlight its capabilities. The app receives a set of photos and collates them into a video with *ffmpeg*. Since collating the photos requires a lot of computation resources, the process can be spread to nearby devices through our Drop Computing framework.

We tested our application using four Android devices, and showed that our framework can drastically decrease CPU usage per device, which increases the overall QoE for Android users. Furthermore, the total battery consumption is lowered, since nodes have less computations to perform and the CPU cores spend less time in higher frequencies. The overall computation time is slightly increased by our implementation, but this is something that we are working to improve in the future. However, this also depends on the size of the computations and the amount of data to be exchanged between devices, so care must be taken when deciding whether to offload or not. We plan on performing experiments while varying the dimension of the tasks (i.e., the amount and size of the photos to be collated), in order to find the point where the size of the task makes the communication overhead irrelevant.

For this paper, we tested the first layer of the Drop Computing paradigm, namely the device-to-device opportunistic layer. However, in the future we wish to also perform some experiments where devices can either perform the computations themselves, offload them to neighbor nodes, or send them to the cloud. We would like to show that using device-to-device collaboration still remains a viable option for intensive computations in certain situations, even when the cloud is available. We also want to address the issue of security [19], [20], which is currently a sensitive topic in MCC.

REFERENCES

[1] R.-I. Ciobanu, C. Negru, F. Pop, C. Dobre, C. X. Mavromoustakis, and G. Mastorakis, "Drop computing: Ad-hoc dynamic collaborative computing," *Future Generation Computer Systems*, 2017.

[2] R. K. Balan and J. Flinn, "Cyber foraging: Fifteen years later," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 24–30, 2017.

[3] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: data forwarding in disconnected mobile ad hoc networks," *IEEE communications Magazine*, vol. 44, no. 11, 2006.

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

[5] G. Huerta-Canepa and 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*, ser. MCS '10. New York, NY, USA: ACM, 2010, pp. 6:1–6:5.

[6] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '12. New York, NY, USA: ACM, 2012, pp. 145–154. [Online]. Available: http://doi.acm.org/10.1145/2248371.2248394

[7] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: Computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '14. New York, NY, USA: ACM, 2014, pp. 287–296. [Online]. Available: http://doi.acm.org/10.1145/2632951.2632958

[8] B. Zhou, A. V. Dastjerdi, R. Calheiros, S. Srirama, and R. Buyya, "mcloud: A context-aware offloading framework for heterogeneous mobile cloud," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[9] F. Bonomi, R. Milito, J. Zhu, and 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*. ACM, 2012, pp. 13–16.

[10] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[11] R.-I. Ciobanu and C. Dobre, "Mobile interactions and computation offloading in drop computing," in *Advances in Network-Based Information Systems*, L. Barolli, N. Kryvinska, T. Enokido, and M. Takizawa, Eds. Cham: Springer International Publishing, 2019, pp. 361–373.

[12] R.-I. Ciobanu, R.-C. Marin, C. Dobre, V. Cristea, and C. X. Mavromoustakis, "Onside: Socially-aware and interest-based dissemination in opportunistic networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–6.

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

[14] F. Rebecchi, M. D. De Amorim, V. Conan, A. Passarella, R. Bruno, and M. Conti, "Data offloading techniques in cellular networks: A survey." *IEEE Communications Surveys and Tutorials*, vol. 17, no. 2, pp. 580–603, 2015.

[15] J. F. Pérez, G. Casale, and S. Pacheco-Sanchez, "Estimating computational requirements in multi-threaded applications," *IEEE Transactions on Software Engineering*, vol. 41, no. 3, pp. 264–278, 2015.

[16] D. Chatzopoulos, M. Ahmadi, S. Kosta, and P. Hui, "Flopcoin: A cryptocurrency for computation offloading," *IEEE Transactions on Mobile Computing*, vol. 17, no. 5, pp. 1062–1075, May 2018.

[17] R.-C. Marin, "Hybrid contextual cloud in ubiquitous platforms comprising of smartphones," *International Journal of Intelligent Systems Technologies and Applications*, vol. 12, no. 1, pp. 4–17, 2013.

[18] D. Chatzopoulos, K. Sucipto, S. Kosta, and P. Hui, "Video compression in the neighborhood: An opportunistic approach," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[19] M. Haus, M. Waqas, A. Y. Ding, Y. Li, S. Tarkoma, and J. Ott, "Security and privacy in device-to-device (d2d) communication: A review," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 1054–1079, 2017.

[20] K. Sucipto, D. Chatzopoulos, S. Kosta, and P. Hui, "Keep your nice friends close, but your rich friends closer - computation offloading using nfc," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.