FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 7 No. 3, June, 2023, pp 239 - 244
DOI: https://doi.org/10.33003/fjs-2023-0703-1770

A BROADCAST RECEIVE MODEL FOR COMPUTATIONAL OFFLOADING IN MOBILE CLOUD COMPUTING

Fatoba, Oluwaseun Jumoke

Applied Mathematics and Simulation Advanced Research Centre, Sheda Science and Technology Complex, Sheda-Kwali, FCT, Nigeria

*Corresponding authors' email: oluwaseunfatoba5@gmail.com

ABSTRACT

Computational offloading is a vital part of mobile cloud computing which has attracted so much attention in recent times. It is a way of saving energy in mobile devices by sending an intensive task to a remote server for execution. However, in existing offloading systems, the opportunistic moments to offload a task are often short-lived. The social aware hybrid computational offloading framework involves outsourcing a task to any available surrogate either remote cloud, cloudlet or device to device, this comes with a drawback of super peer having to constantly supervise the network to discover peers. This takes a considerable amount of energy and time. This research aims to develop an improved model for peer discovery in computational offloading which relieves the use of the super peer and transfers the discovery process between network peers. We used a device profiler that serves as an information collector in peers on the network. We evaluated our model by developing an application for our client nodes in order to get information from our nodes. We evaluated our model by using ten peers with different processing power and RAM. On an average, discovery time for all peers in the existing model was 2040 milliseconds, while we have 1,213 milliseconds for our new model. Energy level for the existing model was 72.5% while we have 82.3% for our new model**,** evaluating our model with the existing one, it was discovered that we saved more energy and time.

Keywords: Offloading, Peer, Community, Broadcast and receive, Device Profiler

INTRODUCTION

Mobile cloud computing is the hybrid of cloud computing technology, mobile cloud computing and wireless network makes good computational resources available to mobile users (Oladeji & Olubunmi, 2017). Computation offloading combats the shortcomings of Smart Mobile Devices (SMDs) such as limited battery lifetime, limited processing capabilities, and limited storage capacity by offloading the execution and workload to other capable systems with better performance and resources. Mobile Cloud Computing allows the SMDs to offload their workloads on the remote cloud servers and benefit from the Mobile cloud computing extensive resources (Amir, Mokhtar, Adil, Sarkhel H, Mohammed, & Mohammed, 2021). User demands increases quickly with a lot of resource-intensive and power-sapping applications. (Quang-Huy & Falko, 2020) Nowadays, mobile phones are used for more than making phone calls or sending texts, due to the provision of different mobile applications However, these devices' mobility is hindered by battery life and thus their usage is inhibited. (D. Ferreira et al, 2011)

Due to the insubstantial computation resources, network, storage, and energy, mobile devices are inadequate for executing all computational tasks, especially tasks with big volumes and complex data structures (Hoa & Dong-Seong, 2023). This is more critical when a user does not have access to a source for recharging the battery (Ferreira et al, 2015). In such situations, as the battery life approaches its lowest energy levels, the user may experience frustration and anxiety, among others (Hosio et al, 2016).

As the potential of mobile devices gains ground (in terms of CPU power, network connectivity etc), people increasingly use them for other tasks such as emailing, GPS routing, Internet banking, gaming etc. Even though they advance in technology, mobile devices will always be limited in resources, as restrictions on weight; size, battery life, and so on imposes limitations on computational resources and makes mobile devices more resource constrained than their non-mobile peers. Computational offloading systems can be categorized into cloudlets, remote cloud and device-to-device (D2D) (Shi et al, 2014).

A lot of research work has gone into finding a solution to this problem with different researchers coming up with different models. As a mobile device is always linked to at least one source of network infrastructure throughout of the day, by merging cloudlet, device-to-device and remote cloud offloading, the availability of offloading support was increased.

A community is formed by a set of people that are always together during particular hours, in the weekdays users tend to work in the same workplace or study in the same department. Generally, they encounter the same people during a specific period of time. Thus, this leads to the idea of a short-term community among the peers which are regularly present at a specific time period in specific locations. Every node in the system is called a peer, cloudlets and remote servers are naturally super peers that sustain the system.

Computation offloading is the process of sending computation intensive application components to a remote server. Recently, a number of computation offloading frameworks have been proposed with several approaches for applications on mobile devices. These applications are partitioned at different granularity levels and the components are sent to remote servers for remote execution in order to enhance the potential of Smart Mobile Devices.

Computational offloading systems include cloud, cloudlets and device to device.

CLOUD: The cloud infrastructure is equipped for open use whereby the users can offload one or more tasks to the cloud, e.g Amazon, Microsoft Azure and so on.

CLOUDLETS: By relying on these cloud infrastructures computationally intensive tasks can be offloaded to the cloud. Cloudlets was a way to bring it closer since they are far from the mobile users, it serves as a middleware between the cloud and user. It can be said to be group of computers designed to

quickly provide cloud computing services to mobile devices, such as smartphones, tablets and wearable devices within close geographical range-

DEVICE-TO-DEVICE: Offloading to remote clouds and cloudlets were faced with offloading decisions such as mobility, latency, state of the device and so on, making it difficult to know exactly to offload. In order to address this issue, very recent works has relied on other devices that are carried around by other users which produces a low latency environment to offload. Study has shown that social relations tends to provide support for offloading in device-to-device

(Cuervo et al, 2010) focused on using MAUI (Mobile assistant using infrastructure) for the reduction of energy consumption of mobile applications using fine-grained code offload, it decides at runtime which methods should be remotely implemented, driven by an optimization engine that attains the best energy savings possible under the mobile device's current connectivity limitation. There was a need for further enhancement on MAUI for future execution of more than one method /thread at a time.

(Gordon et al, 2012) introduced COMET (Code Offload by Migrating Execution Transparently) a runtime system to allow plain multi-threaded applications to use multiple machines which was a drawback of MAUI, the system allows threads to move freely between machines depending on the workload. COMET uses the underlying memory model of the runtime to implement distributed shared memory (DSM) with as few connections between machines as possible.

(Verbelen et al, 2012) provided middleware architecture between the cloud and the mobile device by introducing Cloudlets, offloading to the cloud is not always a solution, because of the high WAN latencies, especially for applications with real-time limitations such as augmented reality. Therefore the cloud has to be moved closer to the mobile user in the form of cloudlets. Instead of moving a complete virtual machine from the cloud to the cloudlet, they proposed a finer grained cloudlet concept that controls applications on a component level.

(Habak et al, 2015) designed the Femtocloud system which provides a strong, self-configuring and multi-device mobile cloud out of a cluster of mobile devices. The architecture was developed to enable multiple mobile devices to be configured into a coordinated cloud computing service. It was said that there was a need to design a suitable user motivation system in the future due to the fact that Femtocloud relies on social awareness like students in a classroom, or a coffee shop, there was a need to sustain the system from collapsing.

(Flores et al, 2017) tackled the limitation of Femtocloud by introducing a reputation and credit based mechanism scheme to the offloading system. A social-aware hybrid offloading system (HyMobi), which increases the range of offloading means, was designed. As a mobile device is always linked to at least one source of network infrastructure throughout the day, by merging cloudlet, device-to-device and remote cloud offloading, there was an increase the availability of offloading support. HyMobi was designed with an incentive mechanism based on credit and reputation, represented by points. A peer gathers points when it is sharing its computational resources to other peers' requests, e.g., by contributing resources, remaining in a particular place for a long time, a peer loses points when consuming resources of the community pool.

MATERIALS AND METHOD

The proposed model composes of three main parts, namely:

i. The device peer,
ii. The request handler, and
iii. The code-offload processor.

The Device Peer

Every node in the system by default is called peer. This includes Cloudlets, any remote server providing system services or any mobile device. The peer is composed of three main sub-components which are:

i. The network peer status table
ii. The broadcast and receive module, and
iii. The device profiler

The network peer status table represents a small space in memory where the status of individual peer in the network is stored every specified interval which is managed by the broadcast and receive module.

The broadcast and receive module is responsible for the management of the network peer status table. The broadcast and receive module can assume different states at different times. The states that the broadcast and receive module can assume are four (4), which are:

i. On,
ii. Off,
iii. Idle, and
iv. Discovery.

The device profiler is responsible for the profiling of devices on the network. The device profiler is trigered when a new device enters the network or when a specified time interval that has been set before hand is expired. Another scenario when the device profiler could be trigered is when a device is about to leave the network.

The Request Handler

The request handler is the gateway where a request is processed based on its type, where the type of the request depends on the role of the peer. When a discovery request is received, the request handler responds to a request either from a peer or a set of peers who are interested in an offloading transaction. If the server is not maxed up already, it accepts the request and the offloading process handled by the code offloading processor is executed, but if it is maxed up already, then the requesting peer or peers will have to wait for the next opportuned time to perform a code offload process

The code offload processor component captures the execution details of a computational task during runtime at the method level, e.g., name of the method, parameters, type of the method, etc. this module is responsible for handling the code offloading task of the network

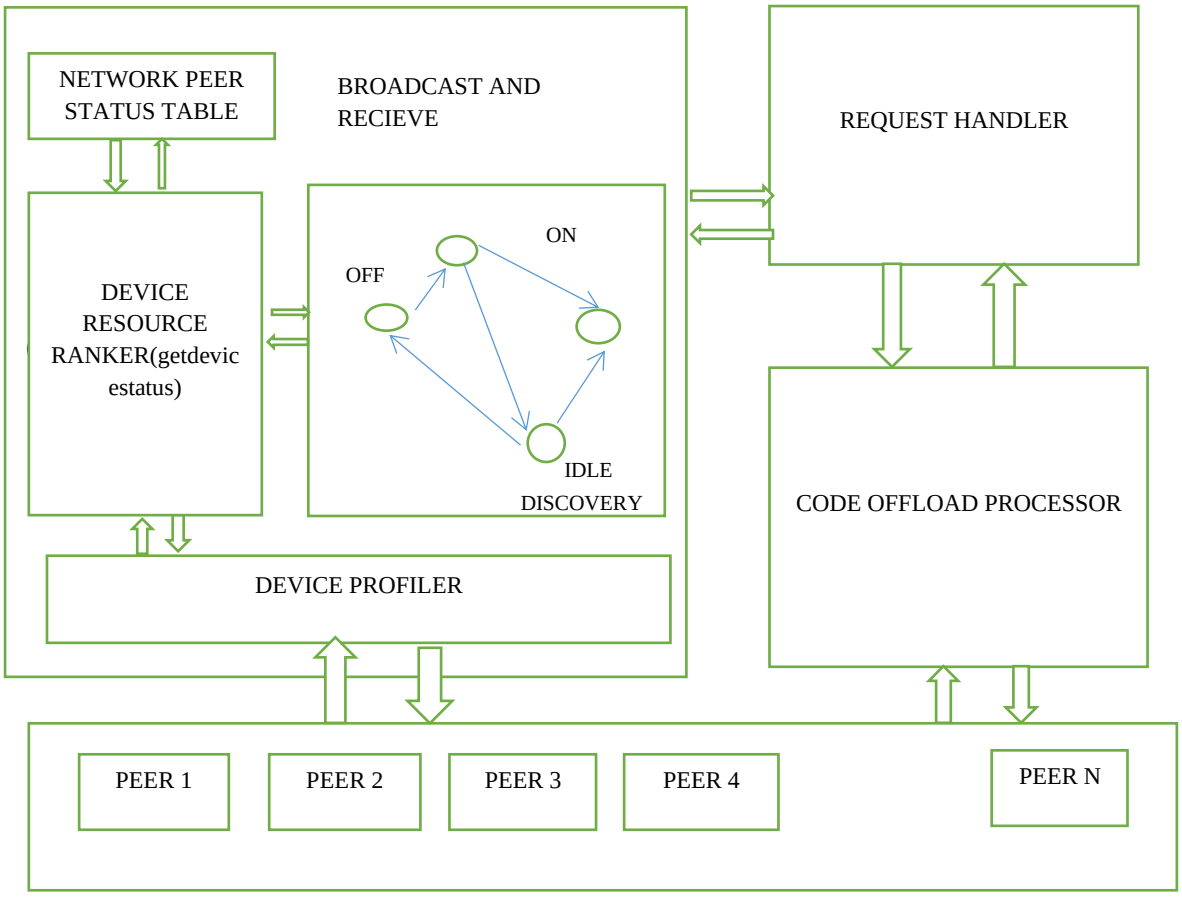

Figure 1: Broadcast and Receive Model

```
Procedure Broadcast_Receive()
        Default_State = OFF_STATE;
        Network_State = OFF;
        If  Network_State == ON then
                Change Default_State to DISCOVERY_STATE
                If network dectected == true then
                        Discover devices on the network , Triger Device_Profiler and change Default_State to
                        ON_STATE;
                        If Default_State == ON_STATE then
                                Broadcast device status and receive device status information from
                                Network_Device_Status_Table;
                        Else if Default_State == IDLE_STATE then
                                Go into sleep mode and wait for STATE_TRIGER_TIMER;
                End if
        End if
End Procedure

Procedure deviceProf()
        Set Device_Profiler == SLEEP;
        If New_Device_Detection == TRUE then
                Set Device_Profiler == AWAKE;
                Call getDeviceStatus();
                Set Device_Profiler == SLEEP;
        Else If TIME_INT == EXPIRED then
                Set Device_Profiler == AWAKE;
                Call getDeviceStatus();
                Set Device_Profiler == SLEEP;
        Else If DEVICE_EXIT == TRUE then
                Set Device_Profiler == AWAKE;
                Call getDeviceStatus();
                Set Device_Profiler == SLEEP;
        Else
                Set Device_Profiler ==SLEEP;
```

```
        End if
End Procedure
```

The **getDeviceStatus()** module serves as the module for performing device status rating. This calculations help reduce the work of selecting the network device peer server. The calculations is based on three (3) parameters, which are:

1. The computing resource of the device,
2. The number of nodes connected to the device, and
3. The energy level of the device.

The equation (1) below shows the the equation used for the device ranking.

Score(Nc,Cc,Bc) = (Nc/N) * (Cc/100) * (Bc/100)

Where:

Nc = the number active connections to the device,

Cc = the computing resource processing frequency,

Bc = the battery level of the device.

```
Procedure request_handler()
        Set request_handler == OFF;
        If New_Device_Detection == TRUE then
                Set request_handler == AWAKE;
                Call device_prof();
                Update the Network_Device_Status_Table;
                Set request_handler == SLEEP;
        Else If DEVICE_EXIT == TRUE then
                Set request_handler == AWAKE;
                Call device_prof ();
                Update the Network_Device_Status_Table;
                Set request_handler == SLEEP;
        Else
                Set request_handler ==SLEEP;
        End if
End Procedure
```

RESULTS AND DISCUSSION

We evaluate and analyze two different aspects of the framework,

i. peer discovery performance and
ii. energy consumption ,

Huber Flores et al used ten mobile devices to carry out their experiment. We replicated that by also using ten devices, the result of the experiment is presented in table below.

Table 1: Results of replication of Huber Flores et al. experiment

PEER	CPU (Hz)	RAM (GB)	DISCOVERY TIME (Milliseconds)	BATTERY LIFE (Percentage)
P0	Quad-core1.4	1	1800	82
P1	Quad-core2.7	3	500	69
P2	Dual-core1.2	1	1870	76
P3	Dual-core1.2	1	500	72
P4	Quad-core2.5	3	1490	85
P5	Quad-core1.5	4	550	58
P6	Quad-core1.4	2	1820	80
P7	Quad-core1.2	1	1880	76
P8	Quad-core1.8	2	1700	70
P9	Dual-core1.5	4	1770	79

The simulator uses a network peer status log file which represents a small space in memory where thestatus of individual peer in the network is stored every specified interval of time. We developed an application that was installed on our mobile clients used by the simulator to handle updates and statistics of nodes on the network. We analysed peer discovery performance and energy consumption of the model, the average discovery time for all peers in the exisiting model was 2040 milliseconds, average discovery time was 1,213 milliseconds while for the new model, the average energy level for the existing model was 72.5% while the average energy level for the new model was 82.3%.

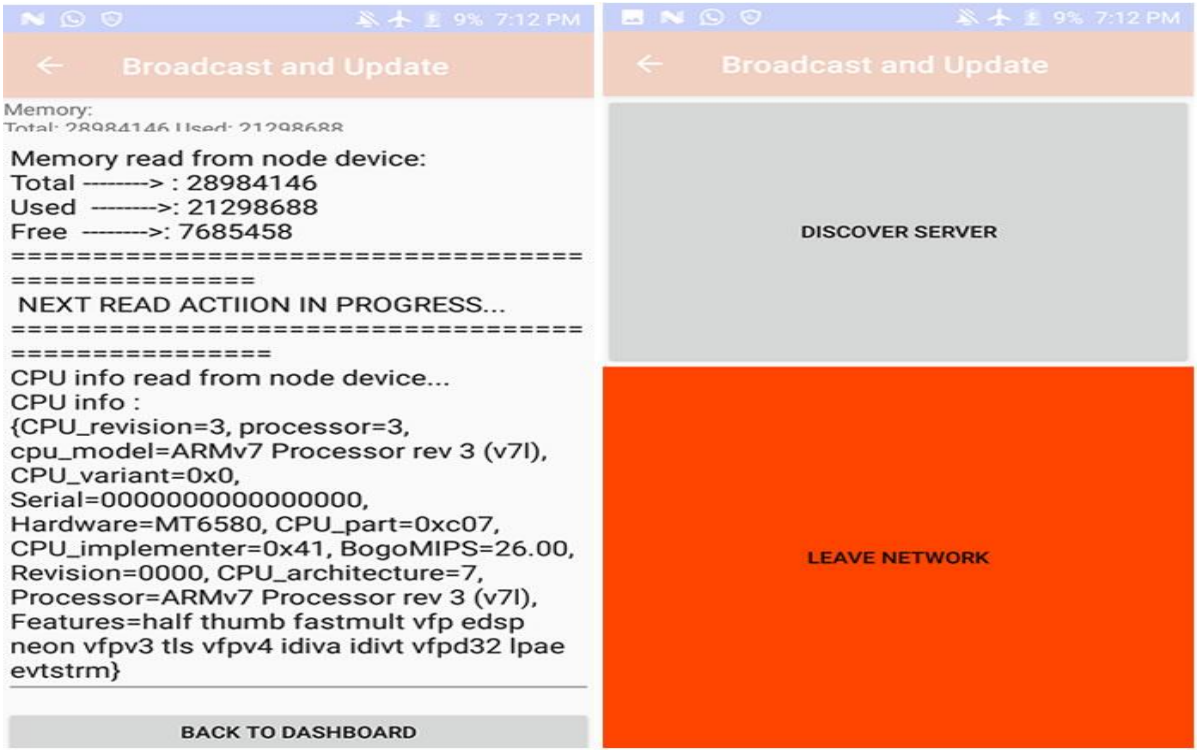

Figure 2: The mobile client interface

The figure above shows the broadcast and receive interface, the request handler handles a request by any peer to discover the server or leave the current network. When such a request is received, it checks to see if the peer has any on going code offloading process, and therefore triggers the device profiler which in turn collects necessary device status information at that particular time as seen in the figure above and updates the network device status table accordingly.

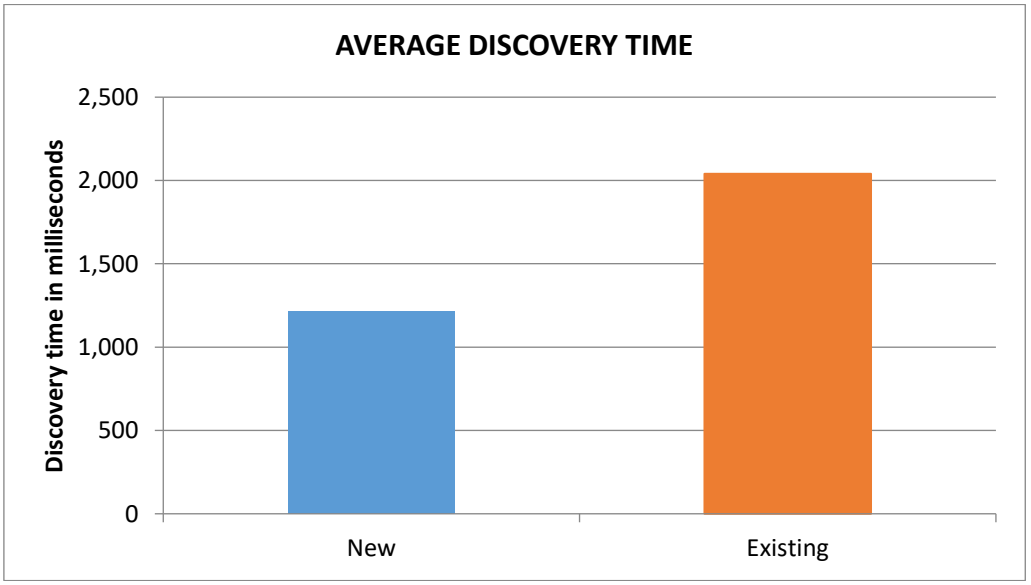

Figure 3: Average discovery time comparison graph for all peers

We used ten devices just as in the existing model done by Flores et al, the devices has different rams and processors for easy comparison, the result above shows the average time it took to discover the peers used in our model, the found the average of 10 peers and compared it with the average discovery time of the existing model.

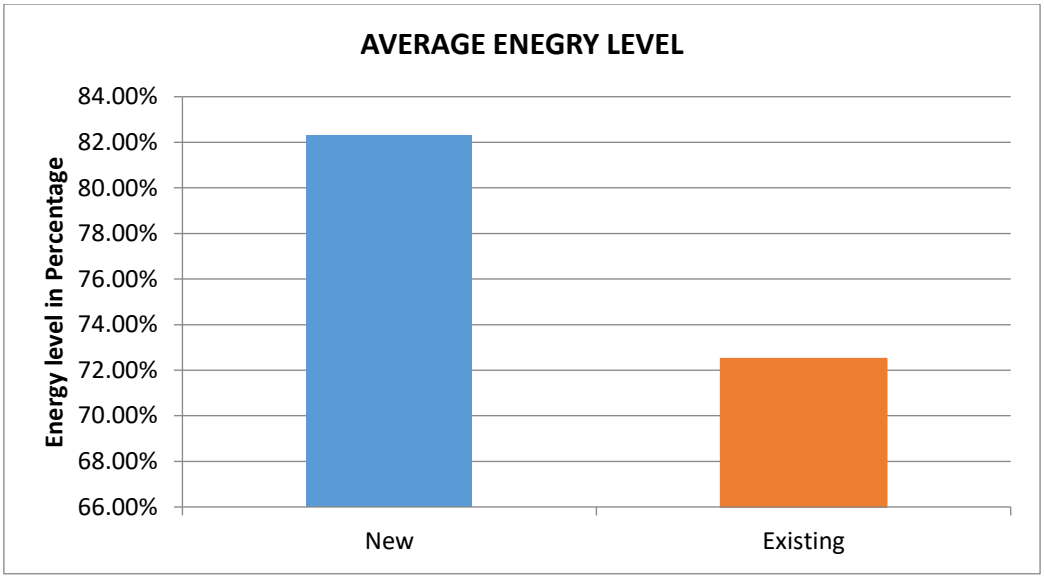

Figure 4: Average energy level comparison graph for all peers

The figure above showed the average energy level of our peers in percentage, we found the average energy level of 10 devices and compared it with the result of the already existing model done by Flores et al.

CONCLUSION

Computational offloading has been proposed as a solution for saving the battery life of the mobile devices therefore, in order to maximize computational offloading process overall, we improved the discovery process of these peers, developed a broadcast and receive model that can effectively gather devices information and update requesting peer. We improved the discovery time it takes to see a peer on the network and also reduced the energy it takes to search for a peer. A peer can offload a task to a more capable device on the network if it doesn't have enough resource to process its task.

REFERENCES

Amir, M. R., Mokhtar, M., Adil, H. M., Sarkhel H, T. K., Mohammed, K. M., & Mohammed, M. (2021). Towards Data and Computation Offloading in Mobile Cloud Computing: Taxonomy, Overview, and Future Directions. *Wireless Personal Communications*.

Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). MAUI: Making Smartphones Last Longer with Code Offload. *MobiSys'10*, *17*, 49–62. DOI: 10.1145/1814433.1814441

Ferreira. D, A.K. Dey, V. Kostakos, (2011) Understanding human-smartphone concerns: a study of battery life, in: International Conference on Pervasive Computing, Pervasive, Berlin, Heidelberg,. DOI:10.1007/978-3-642-21726-5_2

Ferreira P, M.McGregor, A.Lampinen, (2015) Caring for batteries:Maintaining infrastructures and mobile social contexts in:17th InternationalConference on Human-Computer Interaction with Mobile Devices and Services, ACM, MobileHCI'15,.

Flores, H., Sharma, R., Ferreira, D., Kostakos, V., Manner, J., Tarkoma, S., Li, Y. (2017). Social-aware hybrid mobile offloading. *Pervasive and Mobile Computing*, *36*, 25–40 https://doi.org/10.1016/j.pmcj.2016.09.014

Gordon, M., Jamshidi, D., Mahlke, S. & Morley Mao. Z(2012). COMET: code offload by migrating execution transparently. *Proceedings of the 10th …*, 93–106.

Hosio. S, D. Ferreira, J. Goncalves, N. van Berkel, C. Luo, M. Ahmed, H. Flores, V. Kostakos, (2016) Monetary assessment of battery life on smartphones, in:Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI,

Habak, K., Ammar, M., Harras, K. A., & Zegura, E. (2015). FemtoClouds : Leveraging Mobile Devices to Provide Cloud Service at the Edge. DOI**:** 10.1109/CLOUD.2015.12

Hoa, T.-D., & Dong-Seong, K. (2023). DISCO: Distributed Computation Offloading Framework for Fog Computing Networks. *JOURNAL OF COMMUNICATIONS AND NETWORKS*.

Oladeji, Akomolafe Patrick & Olubunmi, Ajayi (2017). Data Offloading Security Framework in MCLOUD. *Journal of Computer Sciences and Applications*, *2017*, Vol. 5, No. 1, 25-28 DOI: 10.12691/jcsa-5-1-4

Quang-Huy, N., & Falko, D. (2020). A smartphone perspective on computation offloading—A survey. Computer Communications, 133-154.

Shi. C, K. Habak, P. Pandurangan, M. Ammar, M. Naik, E. Zegura,(2014) Cosmos: Computational offloading as a service for mobile devices, MobiHo,14. Pages 287–296 https://doi.org/10.1145/2632951.2632958

Verbelen, T., Simoens, P., Turck, F. De, & Dhoedt, B. (2012). Cloudlets : Bringing the Cloud to the Mobile User, 29–35. DOI: 10.1145/2307849.2307858