# DEO: A Smart Dynamic Edge Offloading Scheme using Processing Resources of Nearby Wireless Devices to Form an Edge Computing Engine

Ihsan Alshahib Lami
*Applied Computing*
*School of Computing*
Buckingham, UK
ihsan.lami@buckingham.ac.uk

Ali Al-ameri
*Applied Computing*
*School of Computing*
Buckingham, UK
ali.al-ameri@buckingham.ac.uk

***Abstract*— Edge computing reduces connectivity costs and network traffic congestion over cloud computing, by offering local resources (processing and storage) at one hop closer to the end-users. I.e. it reduces the Round-Trip Time (RTT) for offloading part of the processing workload from end-nodes/devices to servers at the edge. However, edge servers are normally pre-setup as part of the overall computing resource infrastructure, which is tough to predict for mobile/IoT deployments. This paper introduces a smart Dynamic Edge Offloading scheme, (we named it DEO), that forms the "edge computing resource" on-the-go, as needed from nearby available devices in a cooperative sharing environment. This is especially necessary for hosting mobile/IoT applications traffic at crowded/urban situations, and, for example, when executing a processing intensive Mobile Cloud Computing Service (MCCS) on a Smartphone (SP). DEO implementation is achieved by using a short-range wireless connectivity between available cooperative end-devices, that will form the edge computing resource. DEO includes an intelligent cloud-based engine, that will facilitate the engagement of the edge network devices. For example, if the end-device is a SP running an MCCS, DEO will partition the processing of the MCCS into sub-tasks, that will be run in parallel on the newly formed "edge resource network" of other nearby devices. Our experiments prove that DEO reduces the RTT and cost overhead by 62.8% and 75.5%, when compared to offloading to a local edge server or a cloud-based server.**

***Keywords—mobile cloud computing services, edge computing, offloading, parallel processing***

## I. INTRODUCTION

In 2017, 2 billion of SP users are using an estimation of 268 billion MCCS, which represents around 90% of the total mobile traffic [1], the Global Mobile Data Traffic Forecast claimed that there will be around 50 billion connected SPs by 2020 [2]. This exponential growth in SP users has encourage MCCS developers to introduce more processing intensive services. Our proposed DEO experiments are based on SPs to demonstrate the prove of concept of this scheme.

Edge computing solution did emerge to solve such issues by shifting the MCCS computation workload from servers in the cloud to servers near the edge. Thus, reducing both network latency and connectivity costs. That is, a one hop communication between the end-device and the server, to achieve low RTT latency and much reduced internet traffic, especially when adopted at various edge locations/clusters. Having the edge computation infrastructure is of course ideal for the ever-increasing deployment of IoT applications, (e.g. Vehicle or animal tracking, drones-based monitoring, e-health body sensor networks, smart video surveillance and smart homes), that requires intensive computations and activity based on machine/deep learning AI. Actually, recent studies did point out that current edge computing deployments are continually suffering from connectivity issues, due to over subscription to the wireless air interface spectrum, and so expectations that 5G network deployments from 2020 will elevate some of the issues for a while [3]. This means that an on-the-go solution to form an edge-resource when needed, is essential as well as having the pre-planned infrastructure-based edge computing servers that will always be exhausted. The main thinking behind the proposed DEO is to have the capability of establishing a local edge computing resource when and when needed, by forming a local network (via Wi-Fi, and Bluetooth) between available processing devices, (e.g. SPs, tablets, pc's, and servers), in a cooperative sharing environment. That is, DEO overcomes the edge server availability/accessibility limitations, by a scheme that forms an edge computing resource to execute processing intensive MCCS on-the-go from cooperative nearby available devices. DEO offloads the service sub-tasks from the MCCS_host, (a SP running the MCCS), to a local network of nearby devices, using low-cost peer2peer wireless connections between them, so to achieve low RTT latency and minimize the transmitted traffic.

Fig.1 shows DEO end2end scheme. It shows the DEO_Controller, an intelligent engine hosted in the cloud to; (1) recruit cooperative end-devices and authenticate their availability when needed, (2) provide the MCCS_host with decisions on the best scenario to partition and offload the sub-task, so to achieve low execution time and reduce the battery power consumption. As well as to reduce connectivity costs and network traffic congestion.

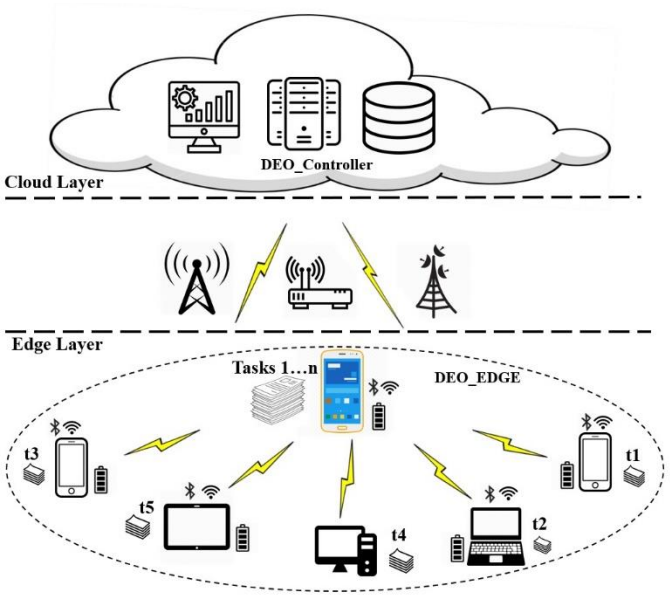


Fig. 1. DEO scheme

Fig.1 also shows the newly formed edge computing resource network (dotted circle in the diagram). When an MCCS_host decides that the MCCS needs offloading, it will ask DEO_Controller for decisions of the nearest available device, (we named it Offloadee), based on the device having, at that time; (1) the lowest load, (2) the highest processing resource (CPU MIPS/Memory/etc), (3) a good battery capacity, and (4) the best network connectivity to use. Then the MCCS_host will; (a) generates VMs (bundle them as APKs and JAR files) of all the partitioned sub-tasks, (b) establishes connectivity with all available devices, as advised by the DEO_Controller, and (c) offloads a portion of the VM's to the Offloadees and retrieves the results.

The novelty contributions of this paper are:

- A unique scheme that forms the edge computing resource, on-the-go, from nearby devices and share the execution of the MCCS in parallel among them via short-range wireless connectivity.
- The offloading between the devices on the newly formed edge network is done intelligently by an engine based in the cloud (no impact on the MCCS_host device). This engine recruits' cooperative devices, (paid back by similar cooperation when needs it), and it monitors (processing capability, battery status, and availability) and authenticates (access, session keys and engagement status).

The rest of this paper includes section II that summarizes the recent literature on edge computing solutions, while Section III presents the development of DEO. Section IV presents the experiments, results and analysis. Conclusions and future work are presented in section V.

## II. LITRATURE REVIEW OF EDGE COMPUTING SOLUTIONS

Review of solutions that perform offloading to a centralised server in the cloud has been published in in our previous paper [4]. This review focuses on solutions that perform offloading the IoT/MCCS sub-tasks to pre-setup infrastructure of edge servers. DEO proposed a scheme that forms an edge computing resource to execute processing intensive MCCS on-the-go, from cooperative nearby available devices using low cost peer2peer connectivity. This is achieved by recruiting a group of available processing resources/devices nearby, in a local network to form a cooperative sharing environment using DEO_Controller engine.

IoT deployments have increased the amount of data generated to the cloud; the amount of data hosted in 2018 is equal to the data gathered in all prior years [5]. This has necessitated that data-handling tasks are shifted to the edge nearer to the IoT sensors network. Running these services on cloud servers can have a negative impact on the offloading process, due to network cost and bandwidth traffic. Therefore, an advantage of edge computing is to provide resources in a close proximity, to end-devices so to reduce long RTT latency and eliminate the network congestion. A solution that facilitates offloading of intensive services from a SP to an edge computing server has introduced a model that provides the use of virtual resources in the edge servers [6]. It achieves this by shifting the service execution from a single SP to the edge servers automatically, by dividing a single task to 5 sub-tasks, using 0-1 integer liner programming method. It marks the sub-tasks with a value of (0,1), where "0" stands for sub-tasks to run locally on the SP, such sub-tasks that access SP local features or input and output sub-tasks, while "1" stands for sub-tasks to run on edge server which has multiple virtual resources to handle the execution of the sub-tasks. This then followed by a "decision solver" engine to decide on which virtual resource to select for the incoming 5 sub-tasks, based on the virtual resource "current queue and completion time". Experiments have affirmed that performing the execution at the edge servers can reduce the network cost and internet traffic. However, this model requires pre-setup infrastructure-based edge servers, which is difficult to predict for MCCS/IoT network type computation, and so we believe a more dynamic model that forms the edge computing resources on-the-go is needed.

With the deployment of 5G networks, the future networks are expected to be intelligent systems that will have the power to "self-learn", "self-plan" and "self-predict" to make intelligent decisions. Therefore, adaptation of intelligent AI/ML algorithms are needed in such environments to perform intelligent decisions. A solution that facilities the above issue has proposed some ML techniques to deploy and implement to make dynamic and intelligent decisions on-the-go, as well as to predict the available nearby device when offloading [7]. A good example of such techniques is Reinforcement Learning (RL), which learns from its own experience, it is inspired by behavioural state to make optimal decisions in a stochastic and complex environment. It is normally formulated as a Markov Decision Process (MDP), that provides a mathematical model to calculate state, action and reward functions, as to make intelligent decision making. This technique is deployed in [8] to leverages offloading of intensive IoT applications sub-tasks to the edge computing servers. It achieves this by proposing two mechanisms to migrate and balance the load between servers. (1) "Knowledge-passing mechanism" to exchange information

between servers, such as location, current time, data type and where data is stored. (2) "Markov mechanism" to predict the next server destination based on a historic algorithm, to calculate the best server to select, based on server location, probability factor, processing time, and server resources. We used MDP model to estimate the cost overhead of DEO in terms of processing time, battery power, latency, available bandwidth, and efficiency. The cost function has the advantage of being analytically well tractable since the expectation is additive over time and/or when more devices are available to cooperate and share their unused resources. DEO deploys DEO_Controller engine that profile and partition the tasks to sub-tasks and offload such sub-tasks to the corresponding end-devices. We shall deploy RL in DEO engine using Markov state, action and reward functions, to predict the next available nearby device to use, and to make intelligent decisions of what and where to offload, based on observing and learning from the experience of running a certain MCCS, (e.g. here we used Face detection & recognition).

Tracking humans or animals with drones in crowd sensing scenarios like volcanos or disasters, are examples of nowadays IoT applications. These applications require machine learning and AI algorithms engines to analyse streams of audio, video and image data coming from many sensors. Such intelligent algorithms require a significant computational/processing resources that are not typically available at the edge, but rather available in large data centres in the cloud. An offloading solution that balance the computational workload between the available cloud and the edge resources has been proposed in [9]. It achieves this by shifting the training and testing phases of the workload to the cloud. I.e. the end-device uploads data, which are then labelled and tested by multiple ML algorithms, then, based on the chosen decision, the model is retrieved, sterilized and packed in a shared repository. Only the AI inference engine is positioned at the edge as a micro service that can be accessed through the shared repository. We believe that the concept of letting the cloud be responsible of the overall decision making in splitting the computation workload between the edge and the cloud is commendable. We shall deploy a similar concept in DEO, we used AWS services to perform the creation of the DB and recognition using AWS rekognition service [10], only the recognition results of the extracted faces are saved in a local DB shared repository.

Different communication technologies may affect the offloading process and can lead to different trade-off between the computation and communication results. These technologies, (such as Wi-Fi, 3G, BT, etc), have different parameters in terms of bandwidth, latency and RTT. Hence, a proper analysis of the above need to be considered, to select the best communication type to optimize, when making the offloading decisions. A solution that leverages offloading sub-tasks to cloud/nearby servers has proposed an emulation testbed to emulate the network conditions of existing communication technologies, to find the effect of such technologies when offloading [11]. It achieves this by manipulating 4 communication technologies, (namely BT, Wi-Fi direct, Wi-Fi and 3G), that could be used to offload sub-tasks to cloud/nearby servers, using a traffic shaper. The traffic shaper is used to emulate network conditions by using Dummynet emulation tool to add a certain level of RTT, packet loss and bandwidth to reflect a real network condition. These technologies are examined while pushing intensive sub-tasks through the network between the MCCS_host and cloud/nearby servers. The experiments show that the power consumption of sending/receiving data claims that using BT can outperform Wi-Fi and 3G by 25% and 44%. Also using Wi-Fi direct is a better solution compared to Wi-Fi and 3G, albeit Wi-Fi is still promising to use while offloading, if small RTT is introduced, which means offloading to nearby server using Wi-Fi can have better results than using Wi-Fi to offload to the cloud server. Nevertheless, 3G is yet to be the ultimate technology to be used. Being said that, to justify our implementation strategy and to prove the results, this solution as well as our analysis conducted in our previous paper [4], have affirmed us that a low-cost efficient communication technology has to be used, while offloading. DEO addresses this issue by deploying DEO_EDGE to leverage a local low-cost peer2peer connectivity, using (nearby API which combined BT and Wi-Fi p2p), resulting in a low RTT latency and less network congestion as detailed in section IV.

Offloading and sharing the end-user sensitive data to the cloud or edge servers may cause privacy and security issues resulting in malicious activities. A solution that proposes to secure the data before offloading, has introduced a secure computation offloading model based on a social trust factors to select trustable offloadees [12]. It achieves this by: (1) "Social trust inference engine" that infer social trust from social relation between device A and B. I.e. device A initiate a minimum social trust factor, then device B is selected only if it meets the social trust factor. (2) "Nearby service broker" that uses task tracker to authenticate the process and select the trustable offloadees. It is consisting of 2 sub-models as follow: Community Trust Extractor (CTE), to extract factors from social networks such as social distance or number of common neighbours, and Community Trust Inference (CTI), to infer the social trust between 2 users in the social networks, such as Google and FB. Similarly, the work in [13], listed some techniques that can be implemented to secure data before sharing it with cloud/edge servers. These are steganography, trusted computing techniques, hardware-based secure execution and homomorphic encryption. DEO does not claim to propose security solutions while offloading, as being not the focus of this research. However, DEO provides a secure implementation indirectly by introducing the following: (1) a DEO_Controller engine that monitors and approves the nearby end-devices for qualifying as being secure and fit before offloading. (2) Partitions the tasks and distributes the sub-tasks among a variety of nearby edge devices, so the shared data cannot be retrieved or invoked as a package, and so stealing the sub-task will not impact the overall security of the offloading. (3) DEO uses AWS rekognition service, which is a highly secure service that uses access and secret keys to authenticate the nearby devices. (4) DEO uses peer2peer API protocol [14] to communicate the nearby devices, which is a secure middleware that provides fully encrypted P2P data transfer between nearby edge devices.

## III. DEO SCHEME

DEO implementation system consists of two distinct engines, (namely: DEO_EDGE & DEO_Controller), which

interact together to perform efficient and intelligent offloading decisions.

### A. DEO_EDGE

This engine forms the edge computing resource that will execute the MCCS and is led by the SP that is hosting the MCCS (named MCCS_host here). Any participating device in helping to run the sub-tasks is named the Offloadee. The main functions of DEO_EDGE are: (1) the MCCS_host will generate VMs (bundle them as APKs and JAR files) of all the partitioned sub-tasks, based on the instructions provided by DEO_Controller in the pre-processing stage. (2) The MCCS_host will establish connectivity with all available Offloadees as advised by the DEO_Controller engine. Note that the connectivity will be wireless, (Wi-Fi, BT and p2p). (3) The MCCS_host will offload the VM's to the Offloadees and communicate the results from this process appropriately, as well as the MCCS_host will also be executing its own share of the sub-tasks as and when it is not busy with the other sub-tasks. (4) When the MCCS execution is completed, a summary record of this experience is feedback to DEO_Controller engine, to train and update it for future execution if needed by other MCCS_hosts. Each of these steps are detailed as part of the experiments section in IV.C.

### B. DEO_Controller

This engine is placed in the cloud, to: (1) identify and recruit suitable end-devices that can be used when needed by DEO_EDGE. This process is continuous, and we envisage that such devices, as a principle, are SP's that are willing to contribute to help other SP's when running demanding MCCS. (2) Perform profiling and partitioning of the MCCS, if not already done in a previous request, when requested by the MCCS_host. (3) Provide a list of potential available SPs/devices near the location of the MCCS_host together with their capability, and advice the MCCS_host with the profiling and partitioning process. Note that the choice of having the profiling and partitioning tasks of the MCCS in the cloud was to save battery power of the MCCS_host, and source knowledge of the MCCS provided by the developer is more accessible in the cloud. DEO_Controller engine has the power to "self-learn", "self-plan" and "self-predict" to make intelligent offloading decisions. We used MDP model to estimate the cost overhead of DEO in terms of processing time, battery power, latency, available bandwidth, and efficiency. The cost function has the advantage of being analytically well tractable since the expectation is additive over time and/or when more devices are available to cooperate and share their unused resources. We shall deploy a simple KNN algorithm to find the nearest end-device to select, and a Markov state, action and reward functions to predict the next available nearby device to use that has the lowest load and the highest resources, based on observing and learning from the experience of running FDR. Being said that, we investigated some of the intelligent AI/ML algorithms that could be deployed and implemented in DEO_Controller. (1) Reinforcement Learning (RL), it learns from its own experience, it inspires by behavioural state to make optimal decisions in a complex environment. It is normally formulated as an MDP, and Genetic Algorithms (GA) to decide on what tasks to offload. (2) Supervised/Unsupervised Learning (SL/UR), SL learns from training input data set, such as Support Vector Machine (SVM), and Support Vector Regression (SVR) algorithms. In particular UL, learns from unlabelled data, it tries to find hidden attributes and data structure to achieve prediction and intelligent decisions, such as K-means. It clusters a group of edge servers as possible offloading targets to help with the execution. (3) Deep Learning (DL), it extracts features from massive data to predict and make decisions automatically. These features are learned dynamically from data without manual setup. DL can be deployed in the edge servers in an offline manner to train and test the data, then it provides inference platform to predict and make intelligent decisions.

## IV. EXPIERMENTS, RESULTS & ANALYSIS

The following experiment scenarios is used to prove that DEO can provide an on-the-go (dynamic) edge resource from available nearby devices, and will perform as good as, or better than, a structured pre-setup edge computing server. The details of the implementation of DEO_Controller engine will be documented elsewhere as being not the focus of this paper.

### A. MCCS Choice: Face Detection (FD) & Recognition (FDR)

FDR is chosen to demonstrate the computational complexity and the benefits of offloading, (typically used by police or at an airport mobile search activities). It involves a variety of complex tasks including face detection, feature extraction (we named it here FD) and recognition (we named it here FDR). We developed FDR using Android studio platform and Dlib library, which is an open source library for image detection and recognition. It obtains a face bounding box using (x,y) coordinators of the face in the image, and then it detects and draws 68 (x,y) coordinators of the face, and finally, it extracts the face features. Asysnc class is basically used to run the heavy part of FDR algorithm on another thread, so no pressure on the main thread that is also handling the Graphic user interface. We used the mface.train function to train the algorithm to perform FDR. Then we called the recognizeAsync function to execute the algorithm. Full details about the specifications of the scenarios and experimental devices are illustrated in section C.

As shown in Fig.2, the main GUI of FDR contains three main buttons which are Offloader, Offloadee and server. The Offloader, (which represents MCCS_host), button is to specify whether to run the sub-tasks locally on MCCS_host or remotely on Offloadees. It shows a drop-down list of Offloadees between "(0-3)", (we used up to 4 devices in this experiment, (note that the maximum number of devices to be used is 7 because the BT protocol only allows 7 actual devices to connect to one master node [15])). The "(0)" means the sub-task runs locally on the MCCS_host, while "(1-3)" specify the number of Offloadees. The Offloadee button is to represent the participated Offloadees.

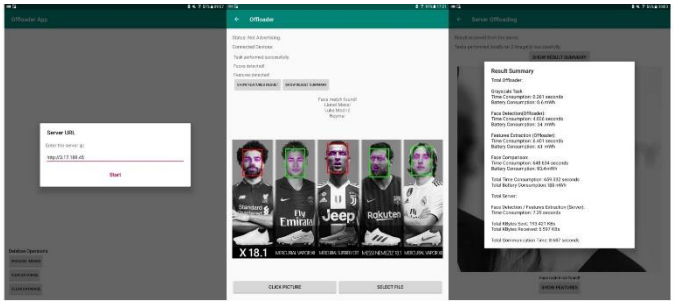

Fig. 2. Screenshots of running FDR

The server button is for running the sub-tasks remotely on the server, (we used 2 servers in this experiment, the first one is a cloud AWS EC2 server, and the second is a local edge WAMP server), it requires a server IP address to start the connection. We developed a simple algorithm to distribute the images among the Offloadees and the servers. Firstly, we divide the number of images (n) equally among the total devices. After that we find the remaining number of images, if the remaining images are equal to "0", then the algorithm starts distributing the images. If the remaining images are > "0" then we distribute the remaining images one by one to the Offloadees. (For example, if the number of connected devices = 4, number of images = 10, then 10/4, so initially each device gets 2 images, then for the remaining 2 images, it assigns one by one to the devices, so MCCS_host = 2, offloadee1 = 3, offloadee2 = 3, offloadee3 = 2 and so on). We used a third-party tool (AWS rekognition service) that uses storage-based API operations to create the DB, which is needed for the recognition sub-task. It gets the images from FDR local repository root, then we call Detectface request, callFaceDetails, and Detectfeatures functions to build a client-side index.

### B. Cost Model

DEO uses a simple cost estimator model based on MDP model, to calculate the cost overhead of our solution [16]. The cost function has the advantage of being analytically well tractable, since the expectation is additive over time and/or when more devices are available to cooperate and share their unused resources. The aim of using MDP model is to normalize the cost equation, and to combine the arbitrary of different unites and scales. Let I denote a set of tasks = $\{i_1,i_2,\ldots.i_n\}$, let J denote a set of devices = $\{j_1,j_2,\ldots\ldots j_m\}$. To define the cost function of task $i_n$ running on devices $j_m$, we calculate the cost function as:

$$Cost(C) = \sum_{j\,\epsilon\,jm}^{i\,\epsilon\,in}\left(P_{i,j} + T_{i,j} + L_{i,j} + D_j + B_j + E_j\right) \quad (1)$$

Where, $P_{(i,j)}$ is the battery power cost of processing task i on device j, $T_{(i,j)}$ is the time for processing task i on device j, $L_{(i,j)}$ is the round trip time latency between 2 devices, $D_{(j)}$ is the available data rate at device j, $B_{(j)}$ is the available bandwidth available at device j, and $E_{(j)}$ is the efficiency of device j based on the CPU load. Since we normalized the cost equation, the units of response time, latency and power consumption will not affect the trade-off. We define a random weight factor (W) between {1,6}, that represents the probabilistic relative significance of power consumption, latency, response time, bandwidth, data rate and efficiency, and it is determined by:

$$W_P = \frac{w_p}{w_p+w_t+w_l+w_d+w_b+w_e} \quad (2)$$

Considering that P is the highest priority in our implementation, followed closely by E and T respectively, they are weighted as 6, 5.5 and 5 scores being near the top of the scale. For D and B, they are regarded as middle priority due to heavily dependent on the available nearby devices and the available connectivity. So, the impact will not strongly influence the P, E and T which are the main objectives for the offloading, and both D and B score is 3. The remaining L scores 2 considering it is the least priority when compared with other weightings. So, we calculate the cost function as:

$$= \sum_{j\,\epsilon\,jm}^{i\,\epsilon\,in}\left(w_p \times P_{i,j} + w_t \times T_{i,j} + w_l \times L_{i,j} + w_d \times D_j + w_b \times B_j + w_e \times E_j\right) \quad (3)$$

### C. Expiermental Scenarios

In this section, the various scenarios for the experiments that has been done to illustrate the overhead of forming the edge resource are described. The aim of these scenarios is to examine the benefit of DEO when offloading, in terms of computation time, battery power consumption and wireless connectivity costs, when FDR sub-tasks are executed by various devices together with the MCCS_host. These scenarios are referred as Edge Server Scenario (ESS), Edge Offloadees Scenario (EOS) and Cloud Server Scenario (CSS) in this paper.

*a) The MCCS_host Offloads FDR Sub-tasks to ESS:* In this scenario, we created a WAMPSERVER 3.1.0, which acts as a local nearby edge server. Both MCCS-host and the server are connected through an IP address. If the decision is to run the sub-tasks on ESS, the decision engine triggers the distribution algorithm to partition the images between the MCCS_host and the server. The MCCS_host generates a serializable interface and decides on the serialized sub-tasks and the images to be offloaded. Then it invokes the remote manager, to connect to the server using IP address and post API and offload the images in parallel. The edge server waits and listens to any incoming sub-tasks, it runs the requested sub-tasks, records the time using timestamps, converts it to JSON format, and sends the results back to the MCCS_host, as will be stated later in section D.

*b) The MCCS_host Offloads FDR Sub-tasks to EOS:* In this scenario, we performed offloading to cooperative nearby edge-devices on-the-go. We used one MCCS_host and a maximum number of 3 end-Offloadees, full specification of the conducted devices used, are shown in Table I. All the devices are connecting through nearby API which is a peer-to-peer networking API that allows applications to connect, share, and exchange data with each other in order to communicate over a local area network. We used nearby connection type since it offers unlimited payload to be shared, and it supports sensitive data by encrypting the data for secure payload exchange. We defined 5 classes to establish the communication between edge Offloadees, these are Start Discovery (), Start Advertising (), Endpoint Discovery Call back (), Request Connection (), and Payload Call back (). When the device is registered itself as an Offloadee, the MCCS_host starts accepting incoming connections, (the number of incoming connections is equall to the number of Offloadees).

TABLE I. EOS-EXPIERMENTAL SPEC FOR (MCCS_HOST & OFFLOADEES)

| **Devices used** | **Devices specification** | | | |
|---|---|---|---|---|
| | ***CPU*** | ***RAM*** | ***OS*** | ***Battery*** |
| Samsung Sm-T710 | 1.3 GHz | 3 GB | Android 7.0 | 4000 mAh |
| Lenovo TB-7304F | 1.3 GHz | 1 GB | Android 7.0 | 3500 mAh |
| LG Nexus 4 | 1.5 GHz | 2 GB | Android 5.1.1 | 2100 mAh |
| LG Nexus 4 | 1.5 GHz | 2 GB | Android 5.1.1 | 2100 mAh |

When we select more than "0" in the drop-down list, the MCCS_host starts advertising itself to accept incoming connections from nearby Offloadees. The Offloadees then discover the MCCS_host and send a request to connect. The MCCS_host accepts the connection and adds the incoming Offloadee to the connected devices list. Then the connection is established, and devices are ready to exchange images between them. We developed a simple algorithm to distribute the images among Offloadees explained in section A. I.e. if we have 20 images to run, then each device executes 5 images in parallel and performs the required sub-tasks for the images, then sends the results back to the MCCS_host. The Offloadees wait and listen for any incoming sub-tasks, when the devices receive the images, they run the grayscale, face detection and feature extraction sub-tasks, record the time using timestamps and send the results back to the MCCS_host. A total of 100 images to perform offloading between a variety of edge end-devices are used. The images are set to have the same resolution (700X700), and have a maximum size of 300 KB, and tests are repeated 5 times to examine stable and unstable network channels when offloading. The results are calculated (an average of 5 runs) in terms of computation time, battery power consumption, communication saving, and offloading gain, as illustrated in section D.

*c) The MCCS_host Offloads FDR Sub-tasks to CSS:* In this scenario, we created a server in the cloud using Amazon AWS services, namely t2.micro Amazon Linux 2 AMI EC2 server. We created the credentials (secret, access, and IAM keys) to authenticate the server with FDR, so it connects and pushes images to the cloud server. We used FileZilla and Putty tools to install and migrate the necessary PHP files to the server. We created an S3 bucket to save the offloaded images if needed for future execution and/or to train DEO_Controller engine. If the decision is to run the sub-tasks on the server, the MCCS_host connects to the server and starts to offload the images through an IP address and POST API. The server waits and listens to any incoming sub-tasks, it runs the requested sub-tasks when receives the images, records the time, converts it to JSON format, and sends the results back to the MCCS_host, as will be stated later in section D.

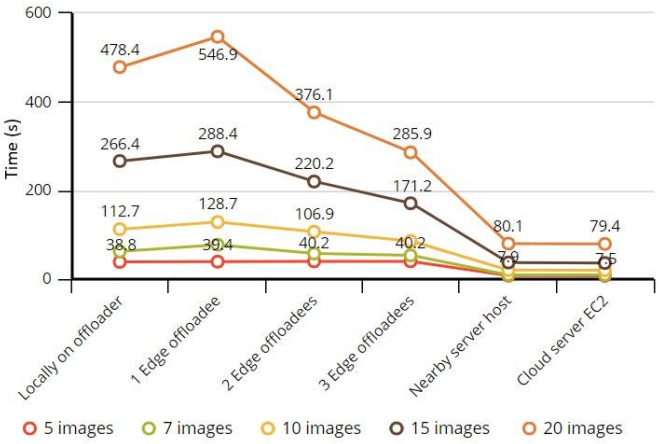


Fig. 3. Processing time of FD

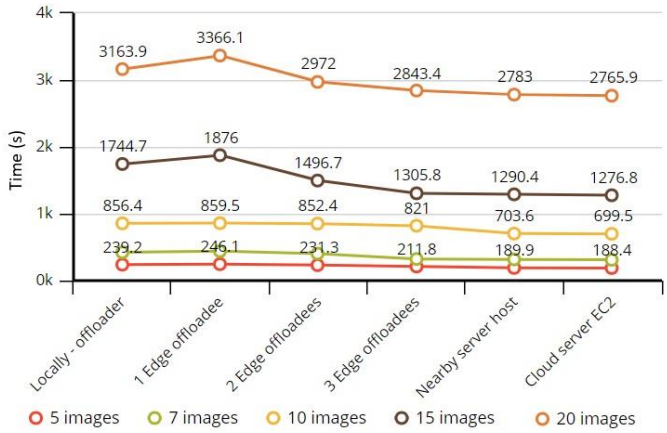


Fig. 4. Processing time of FDR

## D. *Results & Discussion*

This section presents all the results achieved from the conducted various experiments, for the scenarios we designed to illustrate the concept of DEO solution using MDP model.

Fig.3 shows the processing time of executing FD for ESS, EOS and CSS we described in section 4.3. Offloading to ESS and CSS has reduced the burden on the MCCS_host by 83.4% due to their unlimited resource capability. Note that the results are testimony that having an edge server is the correct decision, since it will be less overhead when communication traffic is taken into consideration. It also clear that offloading to a single Offloadee is too costly with an increase of the task by 14.3% due to the overhead not meeting the crossover point of being advantageous. However, offloading to >1 Offloadee has significantly improved the MCCS_host resource capability, (21.3% & 40.2% for 2 & 3 Offloadees respectively).

Fig.4 shows the processing time when running FDR for ESS, EOS and CSS. It shows an increase of the complexity of the FD, by adding the recognition sub-task with the DB. This highlights the importance of DEO, where the processing time became liner for all ESS, EOS and CSS. This means that the overall cost of DEO is much less than having the offloading done to the cloud, without the network traffic caused by transporting the data to the cloud.

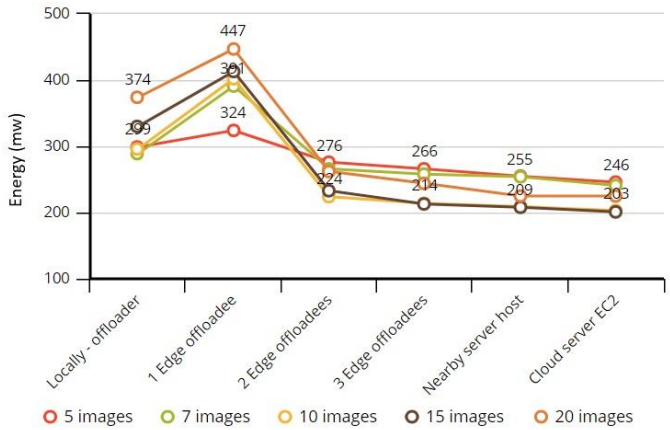


Fig. 5. Battery power consumption of FDR


Fig. 6. RTT latency

For 20 images with 4 edge end-devices, we achieved 10.13% in comparison to running the sub-tasks locally on the MCCS_host, while 12.1% for the CSS scenario, which indicates the DEO will outperform offloading to the cloud solution when more intensive sub-tasks are executed on more participated edge end-devices.

The battery power consumption measured when executing FDR for ESS, EOS, and CSS are shown in Fig.5, it clearly shows that same saving pattern is achieved with computation complexity. The behavioural trend we observed is, when only 2 nearby Offloadees are executing the FDR, the battery power consumption cost increased by 19.52%. However, when the number of Offloadees increases in EOS, we record a power saving of 28.8% for 4 Offloadees running FDR in parallel, which is almost similar with ESS and CSS which record 31.8% power saving.

Fig.6 shows the RTT latency when the MCCS_host communicate with ESS, EOS and CSS, in comparison to the standard latency of Amazon Web Service Server (AWS-S), (red dot line in the Figure). EOS outperforms ESS and CSS, it achieves a decrease of 62.83% resulting in less RTT latency. However, both ESS and CSS achieved a latency reduction by up to 32.75% compared to MDC offloading solution [17].

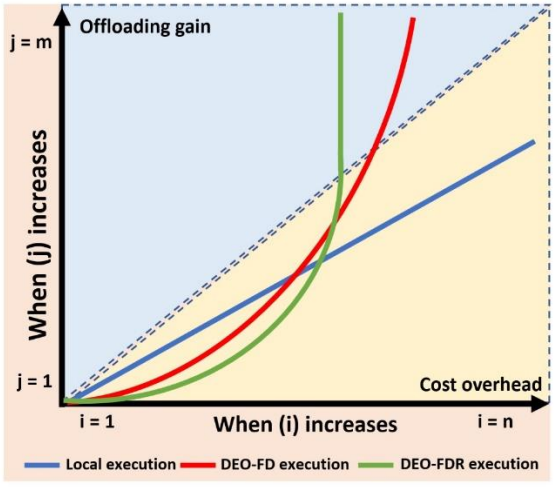


Fig. 7. Cost overhead

This proves the concept of using DEO to offload FDR sub-tasks to edge server is a better solution than offloading to the cloud, due to the fact that, it sends less data to the cloud which reduces network cost and bandwidth traffic.

Fig.7 shows the cost overhead of DEO obtained from using MDP model to estimate the cost function as detailed in section B. It shows the cost overhead occurred from running FD & FDR sub-tasks locally on the MCCS_host and by using DEO scheme. At the starting point, when we have only one nearby Offloadee helping with the execution of FD & FDR, the cost overhead increases by 24.2% compared to the local execution. When the number of Offloadees increases, (i.e. >1 Offloadee), we observed that DEO outperform the local execution, and the offloading gain raised up steadily by 75.5% compared to the local execution. Also, when running more intensive sub-tasks, using DEO to execute FDR records a better offloading gain than running FD, which shows the advantages of DEO when more intensive sub-tasks are running on more Offloadee devices, (here 4 sub-tasks running on an MCCS_host and up to 3 Offloadees). This proves that forming an edge solution is the ultimate solution to reduce the network cost and bandwidth traffic.

## V. CONCLUSION & FUTURE WORK

The impact of connectivity between our local edge resource network and the cloud is significant and depends on the location of the MCCS_host. For example, if the Cloud server is only accessible by cellular link, then the overheads will be 10x more than if a Wi-Fi link is available to connect to the server. DEO eliminate the extra traffic occurred from pushing a tremendous amount of data to the cloud/edge server. DEO cost overhead is very small, it achieves up to 75.5% reduction when we execute more sub-tasks on 4 edge devices, which justifies that forming a low cost local peer2peer network of nearby available nodes/devices, (even with small number of 4 devices), is promising and can reduce the RTT latency resulting in less network congestion. Our future study on this thread will focus on the granularity and partition of the sub-tasks so to maximise the benefit from the Offloadees without having to run their battery to the ground or increasing the local connectivity traffic with them. However, having only a single Offloadee to help with the FDR is not an option.

## ACKNOWLEDGMENT

Gratitude to the University of Basra, and MOHESR (Ministry of Higher Education & Scientific Research) for sponsoring this work.

## REFERENCES

[1] Saha, Sajeeb and Habib, Md Ahsan and Razzaque, Md Abdur, “Compute intensive code offloading in mobile device cloud,” Region 10 Conference (TENCON), 2016 IEEE, 2016.

[2] Elmannai, Wafa, and Khaled Elleithy. "Sensor-based assistive devices for visually-impaired people: current status, challenges, and future directions." Sensors 17.3 (2017): 565.

[3] Song, Chuang, et al. "Hierarchical Edge Cloud-based Traffic Offloading Enabling Low-Latency in 5G Optical and Radio Network." Asia Communications and Photonics Conference. Optical Society of America, 2017.

[4] Ali Al-ameri and Ihsan Alshahib Lami, “SCCOF: Smart Cooperative Computation Offloading Framework for Mobile Cloud Computing

Services,”, in the 8th Annual International Conference: Big Data, Cloud and Security, 2017.

[5] Dwivedi, Akhilesh, et al. "Internet of Things'(IoT's) Impact on Decision Oriented Applications of Big Data Sentiment Analysis." 2018 3rd International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU). IEEE, 2018.

[6] Wei, Xiaojuan and Wang, Shangguang and Zhou, Ao and Xu, Jinliang and Su, Sen and Kumar, Sathish and Yang, Fangchun, “MVR: An architecture for computation offloading in mobile edge computing,”, in the IEEE International Conference on Edge Computing, 2017.

[7] Cao, Bin, et al. "Intelligent Offloading in Multi-Access Edge Computing: A State-of-the-Art Review and Framework." IEEE Communications Magazine 57.3 (2019): 56-62.

[8] Xiao, Kaile, et al. "A Heuristic Algorithm Based on Resource Requirements Forecasting for Server Placement in Edge Computing." 2018 IEEE/ACM Symposium on Edge Computing (SEC). IEEE, 2018.

[9] Calo, Seraphin B., et al. "Edge computing architecture for applying AI to IoT." 2017 IEEE International Conference on Big Data (Big Data). IEEE, 2017.

[10] “Amazon Rekognition: Developer Guide” [Online]. Available: http:// docs.aws.amazon.com/rekognition/latest/dg/rekognition. [Accessed January 2019].

[11] Mtibaa, Abderrahmen, Khaled A. Harras, and Afnan Fahim. "Towards computational offloading in mobile device clouds." 2013 IEEE 5th international conference on cloud computing technology and science. Vol. 1. IEEE, 2013.

[12] Su, Wei-Tsung, Chiang-Sheng Liang, and Cheng-Yi Dai. "Secure computation offloading based on social trust in mobile networks." 2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN). IEEE, 2014.

[13] Shibin, D., and G. Jaspher W. Kathrine. "A comprehensive overview on secure offloading in mobile cloud computing." 2017 4th International Conference on Electronics and Communication Systems (ICECS). IEEE, 2017.

[14] “Nearby Connections API” [Online]. Available: https://developers.google.com/nearby /connections/android/exchange-data. [Accessed July 2018].

[15] Sirivianos, Michael, et al. "Dandelion: Cooperative Content Distribution with Robust Incentives." USENIX Annual Technical Conference. Vol. 7. 2007.

[16] Goudarzi, Shidrokh, et al. "A hybrid intelligent model for network selection in the industrial Internet of Things." Applied Soft Computing 74 (2019): 529-546.

[17] A. Mtibaa, M. A. Snober, A. Carelli, R. Beraldi and H. Alnuweiri, "Collaborative mobile-to-mobile computation offloading," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on, 2014.

# CloudSummit19 notification for paper 24

C

CloudSummit19 <cloudsummit19@easychair.org>

Reply all

Mon 24/06, 05:23

ALI AL-AMERI

Dear Ali,

Congratulations! Your paper 24 titled DEO: A Smart Dynamic Edge Offloading Scheme using Processing Resources of Nearby Wireless Devices to Form an Edge Computing Engine has been accepted by IEEE Cloud Summit 2019.

Please note that you must present at the conference before your paper can be published on the conference's proceedings with IEEE Xplore. If the decision for your paper is "accept with revision", you should be preparing the final camera-ready version based on reviews' comments below.

The conference program/schedule will be available July 2019. Registration is scheduled to open on June 24, 2019. Registration website: https://ieeecloudcomputing.regfox.com/ieee-cloud-summit-2019

IEEE Cloud Summit 2019
https://www.ieeecloudsummit.org

SUBMISSION: 24
TITLE: DEO: A Smart Dynamic Edge Offloading Scheme using Processing Resources of Nearby Wireless Devices to Form an Edge Computing Engine