Simulation Modelling Practice and Theory 101 (2020) 102019

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Dynamic decision support for resource offloading in heterogeneous Internet of Things environments

Ali Jaddoa[a], Georgia Sakellari[*,a], Emmanouil Panaousis[a], George Loukas[a], Panagiotis G. Sarigiannidis[b]

[a] Computing and Mathematical Sciences, University of Greenwich, London, UK
[b] Engineering Informatics and Telecommunications, University of Western Macedonia, Kozani, Greece

ARTICLE INFO

Keywords:
Internet of things
IoT offloading
Computation offloading
Edge computing
Cloud computing
Decision support
EdgeCloudSim simulator

ABSTRACT

Computation offloading is one of the primary technological enablers of the Internet of Things (IoT), as it helps address individual devices' resource restrictions. In the past, offloading would always utilise remote cloud infrastructures, but the increasing size of IoT data traffic and the real-time response requirements of modern and future IoT applications have led to the adoption of the edge computing paradigm, where the data is processed at the edge of the network. The decision as to whether cloud or edge resources will be utilised is typically taken at the design stage based on the type of the IoT device. Yet, the conditions that determine the optimality of this decision, such as the arrival rate, nature and sizes of the tasks, and crucially the real-time condition of the networks involved, keep changing. At the same time, the energy consumption of IoT devices is usually a key requirement, which is affected primarily by the time it takes to complete tasks, whether for the actual computation or for offloading them through the network.

Here, we model the expected time and energy costs for the different options of offloading a task to the edge or the cloud, as well as of carrying out on the device itself. We use this model to allow the device to take the offloading decision dynamically as a new task arrives and based on the available information on the network connections and the states of the edge and the cloud. Having extended EdgeCloudSim to provide support for such dynamic decision making, we are able to compare this approach against IoT-first, edge-first, cloud-only, random and application-oriented probabilistic strategies. Our simulations on four different types of IoT applications show that allowing customisation and dynamic offloading decision support can improve drastically the response time of time-critical and small-size applications, and the energy consumption not only of the individual IoT devices but also of the system as a whole. This paves the way for future IoT devices that optimise their application response times, as well as their own energy autonomy and overall energy efficiency, in a decentralised and autonomous manner.

1. Introduction

As a result of their resource restrictions, Internet of Things (IoT) devices typically rely on the storage, communication, and most significantly, computation resources of remote cloud infrastructures, for example to run computationally intensive artificial

* Corresponding author.
E-mail addresses: A.H.Jaddoa@gre.ac.uk (A. Jaddoa), G.Sakellari@gre.ac.uk (G. Sakellari), E.Panaousis@gre.ac.uk (E. Panaousis), G.Loukas@gre.ac.uk (G. Loukas), PSarigiannidis@uowm.gr (P.G. Sarigiannidis).

https://doi.org/10.1016/j.simpat.2019.102019
Received 2 July 2019; Received in revised form 11 October 2019; Accepted 4 November 2019
Available online 13 November 2019
1569-190X/

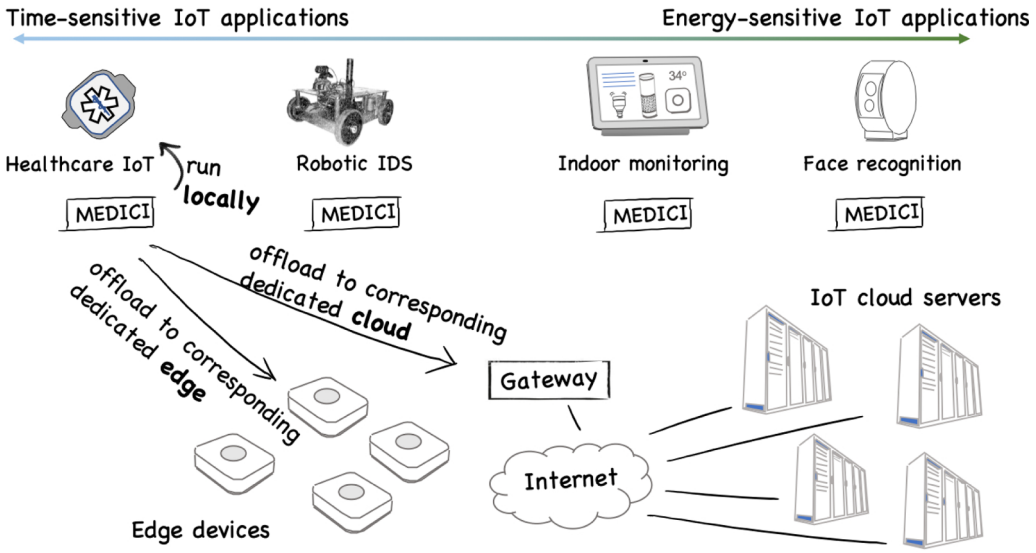

Fig. 1. Conceptual IoT offloading decision making architecture.

intelligence algorithms. This traditional IoT-cloud approach has worked well in the first years of IoT, but is unlikely to be able to efficiently meet the requirements of future IoT applications [1,2]. IoT applications are becoming increasingly demanding in terms of real-time response requirements, and at the same time, the data they produce is increasing dramatically. Indicatively, the Cisco Global Cloud Index has estimated that IoT data will reach 507 ZB per year by 2019, which is 49 times higher than the current total global data centre traffic (10.4 ZB) [3]. Pushing the computations and data to the cloud from IoT devices that have limited bandwidth or are connected to the cloud through unreliable networks costs IoT services in terms of response time and availability. Moreover, due to their energy limitations, a primary goal is to reduce transmission and scheduling computation to what is practical for the power capabilities of IoT devices.

The edge computing paradigm, where the data is processed or even produced at the edge of a network, was introduced in response to these requirements and is now considered a core enabler of the 5th generation of mobile communications (5G) and the IoT [1,4]. The rationale is that with the significant increase of IoT data and the limited speed of IoT data transportation, offloading computation to the edge may allow most of the benefits of the cloud without its key communication disadvantage [5]. However, its fundamental weakness is that unlike the cloud, which does not have geographical restrictions and has considerably greater resources, edge devices have to be in proximity to the IoT devices and their resources are also limited. The choice of whether a task should be processed locally, at the edge or at the cloud has been traditionally based almost entirely on the original design of the system. Here, we propose a Multi-critEria DecIsion support meChanism for IoT offloading (MEDICI) mechanism that decides dynamically whether an IoT task should be processed locally, at the edge or at the cloud.

Without loss of generality, we assume an IoT architecture where a variety of heterogeneous IoT devices and their corresponding applications are connected to the Internet (and the cloud) through a gateway (Fig. 1) and are directly connected to the edge devices in proximity. Note that we assume that each application has a dedicated corresponding device (edge or cloud). Also, in some IoT architectures the gateway itself can be the edge device for offloading, but this does not materially change our analysis and modelling.

In detail, this paper's contributions are the following:

- A response time and energy consumption model for edge IoT, which takes into consideration the application, device and network characteristics of the system.
- An offloading decision support mechanism for heterogeneous IoT devices, each with their own individual requirements, which have the option of processing a task locally or offloading it to the cloud or the edge.
- An extension of the EdgeCloudSim simulator with support for dynamic offloading decision support to validate our model and compare against previous families of approaches used in the literature. Our results show that both response time and energy are improved considerably, not only at the individual level of each application but also at the global system level, as the total energy consumption too is improved.

2. Related work on decision making in IoT offloading

The majority of edge offloading decision making mechanisms proposed in the literature refer to mobile edge computing [6]. For example, Meurisch et al. [7] address the issue of heterogeneity of the edge or cloud infrastructures for mobile offloading, where the resource availability of the different edge or cloud devices can vary considerably, as might the resource requirements of the tasks to be offloaded. They propose an offloading decision support system that predicts the completion time and energy consumption by

probing edge or cloud devices with micro tasks only lasting a few microseconds, and using regression models. Probing has indeed proven very useful in task allocation and admission control [8] problems but introduces a delay overhead which can be inappropriate for time-sensitive IoT services.

Offloading in IoT environments can differ considerably, not only in terms of the type of networking protocols and architectures involved, but also in terms of the nature of typical applications. Additionally, most published work focusing on IoT offloading mechanisms addresses decisions between two entities only, the cloud or the edge(s), and omits the IoT device itself. An exception is [9], where it is up to the individual IoT devices to decide themselves whether they wish to optimise offloading based on time or energy, and advertise to the other IoT devices in their network, while a centralised network controller allocates the available bandwidth among the nodes, giving higher priority to time-sensitive tasks. The authors demonstrate the usefulness of employing the edge computing paradigm in comparison to just offloading to the cloud.

In [10], Shah-Mansouri and Wong formulate the problem as a computation offloading game of multiple IoT users requiring access to the limited resources of close-by edge devices. The assumption is that the users are selfish and only care about maximising their own quality of experience, measured in terms of reducing computation energy and delay. The authors have proposed a near-optimal algorithm to reduce the complexity of reaching a Nash equilibrium but it is not evident how it can be implemented in an online, dynamic way.

Ma et al.[11] also propose a computation offloading game to model the competition for cloudlet resources between IoT devices. It aims to minimise energy and delay of the IoT sensors. The authors consider the different technologies of communication between the different entities. When offloading computation tasks to cloudlets, IoT sensors transmit data blocks via wireless access points, while when offloading tasks to the cloud the IoT device connects to the Internet via the base station. They propose a finite improvement iteration algorithm to keep the computation complexity of the game algorithm relatively low. The IoT devices are not considered capable of processing the tasks, and thus are not included in the offloading decision. However, the actual decision and estimation of where to offload can be centralised or distributed, whereby the sensors are assumed to run the algorithm. Their evaluations were against *Random Selection* (choosing between edge and cloud) and *Cloud-only* (selecting always the cloud for offloading) strategies.

An IoT offloading technique is proposed in [12] to manage the offloading of computing tasks between IoT devices (smart home controller) and the cloud, based on energy consumption under service time delay restrictions. The authors propose to use the gateway as a middleware platform to decide between local processing and the best cloud infrastructure. They propose a static allocation of resources processed in the smart home controller, based on the application's resource requirements and then, at the gateway, a dynamic allocation to the appropriate cloud server based on the energy savings under QoS delay constraints. The paper does not consider dynamic allocation at the local level or that classes of IoT devices may be able to perform local processing.

Similarly, Igarashi et al. [13] propose a cloud-enhanced home controller that enables computation offloading of smart home applications from the home controller to the cloud, by ranking them based on predefined requirements, priorities, compute resources and network bandwidth values. Apart from choosing between processing at the IoT controller or offloading in the cloud, they also propose a degraded mode of operation when network connection is not possible. The predefined values can be updated but energy consumption of the devices or local processing are not considered.

Samie et al.[14] impose communication bandwidth constraints to manage computation offloading and increase the utilisation of the edge node which will also lead to energy savings for the IoT devices. They propose an iterative bandwidth allocation algorithm to better utilise the usage of the edge device. They consider IoT devices with different transmission rates and different offloading levels. They assume that the devices will always process locally until their capacity is reached and then offload. Henceforth, we refer to this type of strategy as *IoT-first*.

In [15], Lyu et al. propose a simple offloading scheme, where delay-sensitive tasks are always given high priority and are executed immediately at the edge while other tasks are offloaded to a remote cloud. Thus, the edge only executes delay-sensitive tasks, while the cloud is used for the rest.

Several researchers address the need of optimising both time energy. For example, Du et al. [16] formulate an optimisation problem to minimise the energy consumption or latency when offloading to fog or cloud nodes. Similarly, in [17], Liu et al. propose a multi-objective optimisation model which tries to optimise energy consumption, computation latency and payment overhead for fog computing offloading by finding the optimal transmission power and offloading probability. However, these optimisation solutions are based on exhaustive search and traditional iteration methods, which makes their convergence too slow for practical real-world applications [18].

Benedetto and Co-authors [6,19] propose MobiCOP-IoT which is a mobile code offloading system for IoT applications. For estimating the network parameters, MobiCOP-IoT samples the network every 15 minutes or probes whenever it is needed. To calculate the execution time for a given task, it uses historical data, assuming that the same tasks always take the same time to be executed.

Recently, Jalali et al. [20] introduced a task allocation strategy called DEFT. It tries to allocate tasks to nearby discovered devices within a local trusted network and a cloud server. The offloading decision is made based on the assumption that the IoT device can act as a task requester or a task performer. To do so, an IoT requester broadcasts a request to all known devices, enquiring about parameters such as CPU, memory, and transmission rate. These parameters are fed to machine learning algorithms (regression and ensemble models) for predicting how the available nodes would behave for a given task. However, the variation of a task's size or the energy consumption are not considered when making the decision.

Recent work involving computation offloading proposes to use machine learning for the estimation of the response times when making the decision of where to offload. For instance, Alam et al. [21] propose an offloading mechanism that uses Deep Q-learning based code offloading to decide amongst a set of distributed fog nodes, a cloud and mobile devices, while in [22], Alelaiwi proposes to use deep-learning in order to predict the response times of a task based on historical observations or pre-defined parameters such

Table 1
Variables and notations.

Variable	Description/Definition
n	The task initiated at the IoT device
D	The target device where the processing will be performed. This can be the IoT device itself, the edge or the cloud.
z_n	Size of task n
d_n^x	Input data size of task n
d_n^r	Output data size of task n
S_D	The processing speed of device D
$T_{n,D}^{proc}$	The time it takes for task n to be processed locally at device D
$W_{n,D}^{procQueue}$	The time a task has to wait at the queue of device D before it is processed
R_D^x	The end-to-end throughput when device D transmits data to another device)
R_D^r	The end-to-end throughput when device D receives data from another device
$T_{n,D}^x$	The time it takes to transmit the input data of task n to device D
$T_{n,D}^r$	The time it takes to receive the output data of task n from device D
p	Packet loss probability
l	A delay factor due to loss
$W_{n,D}^{netQueue}$	The delay due to network queuing for task n
$T_{n,D}$	The total response time of task n processed at device D
P_D^{proc}	The average processing power consumption of device D when busy
P_D^{idle}	The average power consumption of device D when idle
P_D^x	The average power consumption of device D when transmitting data
P_D^r	The average power consumption of device D when receiving data
$E_{n,D}^{local}$	The energy consumed by device D for processing task n locally
$E_{n,IoT}^{off}$	The energy consumed by an IoT device for offloading task n
$E_{n,D}^{off}$	The energy consumed by device D receiving the offloaded task n
E_n^{off}	The total energy consumed for offloading and processing task n
$E_{n,D}^{selfish}$	The total energy cost of task n in the case of a selfish IoT device
$E_{n,D}^{altruistic}$	The total energy cost of task n in the case of a altruistic IoT device
α	Weight denoting preference in minimising response time over energy, $\alpha \in [0, 1]$
γ	Parameter for bringing $T_{n,D}$ and $E_{n,D}$ into a mutually comparable range of values

as CPU, memory and bandwidth consumption. The use of deep learning though usually translates in high computation cost, making these mechanisms often unsuitable for IoT environments with limited resources.

Reflecting on the literature review, we can observe that the related work does not take into consideration all aspects of an IoT environment. Most work does not consider the IoT device capable of processing tasks and mainly concentrates on the decision between edge and cloud. Additionally, existing work mostly concentrates on one parameter (either time or energy), and in the case of time, it doesn't always include both processing and network delays. When both are considered, they are either equally weighted for all applications or do not have provisions to consider the total energy of the system (including the edge or cloud), and thus not allowing different applications to have different energy/time requirements or goals (such as reducing their personal energy or the energy of the system).

Here, we propose a dynamic offloading decision support mechanism for choosing among the three entities of an IoT environment: local IoT device, edge and cloud. The decision is based on response time (including processing and transmission times) and energy consumption (both individual or global), and can be taken by an IoT device at the moment that a new task is initiated, based on the current conditions of the network and the device's own individual requirements. This allows it to serve highly heterogeneous and dynamic IoT environments, where individual IoT devices (and their applications) may differ considerably between each other and the network conditions may be changing. The decision for where to offload is taken based on the model described in the next section.

3. Offloading model and multi-criteria decision support mechanism

In this section, we model each component of the system to derive an expression for estimating its response time and energy consumption. The decision on which target device (the IoT device, the edge or the cloud) should process the task is taken autonomously on the IoT device itself. Table 1 summarises the notations used.

Let us consider an IoT application consisting of computational tasks, all of which can be offloaded. A given task n has size z_n, representing the computation requirements of the task, and input data of size d_n^x and output data of size d_n^r. Specifically, d_n^x and d_n^r represent the data block to be transmitted as part of offloading (e.g., a video in a CCTV monitoring system) and the data block to be sent back as the result (e.g., the recognised object in the video) respectively. Each device D can be of the type IoT device, edge device or cloud server ($D \in \{IoT, Edge, Cloud\}$), with processing speed S_D. Normally, the IoT device is the slowest and the cloud is much faster in terms of processing ($S_{IoT} < < S_{edge} < < S_{cloud}$). We assume that the IoT devices can communicate with both the edge (e.g., via Wi-

Fi, Bluetooth or Zigbee) and the cloud (e.g., via WAN or cellular), at different effective transmission rates.

3.1. Processing times

In accordance to the usual representation of processing time in task allocation problems [23], the time it takes for a task n with size z_n to be processed at a device D is:

$$T_{n,D}^{proc} = \frac{z_n}{S_D} \tag{1}$$

Every time a task arrives at a device D it enters a processing queue. This can be a single queue (i.e. in the case of the IoT device itself) or one of multiple queues (in the case of more powerful devices such as the edge or the cloud, where multiple cores and multiple virtual machines are available, dedicated to specific IoT applications). We assume that each such queue constantly keeps track of the number of tasks (and their corresponding sizes) that are currently waiting. When a new task n arrives in device D it cannot be processed until all previous tasks of its corresponding queue Q_n^{proc} are processed. If k is the number of tasks waiting in Q_n^{proc} when task n arrives, then we approximate the time that the task n has to wait in the processing queue as:

$$W_{n,D}^{procQueue} = \sum_{i=1}^{k} T_{i,D}^{proc} = \sum_{i=1}^{k} \frac{z_i}{S_D} \tag{2}$$

Here, we assume a first-in-first-out processing model in all devices, where one virtual machine on the edge and one on the cloud is dedicated to one corresponding application.

3.2. Network delays

When the processing of a task n is offloaded from an IoT device to a target device D (edge or cloud), there is an additional delay to transmit the task ($T_{n,D}^{x}$) to that device and an additional delay ($T_{n,D}^{r}$) to get the result back from that device to the IoT device.

In accordance with the standard practice in computation offloading modelling [11,23–25], the time it takes to transmit a task n from the IoT to another device D depends on the size of the input data that the task is associated with (d_n^x), and the end-to-end throughput when device D transmits data to another device R_D^x.

$$T_{n,D}^{x} = \frac{d_n^x}{R_D^x}$$

Similarly for receiving the result back, where the size of the result data is d_n^r and the end-to-end throughput when device D receives data from another device is R_D^r:

$$T_{n,D}^{r} = \frac{d_n^r}{R_D^r}$$

In non-ideal communication conditions, where we consider packet loss due to congestion or failures, with a probability p, we assume that, the delay in establishing that a packet is lost and re-transmitting means that each bit lost incurs an increase in communication delay by a factor $l = \frac{1}{1-p}$, $l \in \mathbb{R}^+$

Thus, the above equations become:

$$T_{n,D}^{x} = l\,\frac{d_n^x}{R_D^x} \tag{3}$$

$$T_{n,D}^{r} = l\,\frac{d_n^r}{R_D^r} \tag{4}$$

We also model imperfect network conditions in the form of congestion, as expressed by network queuing delays. Similarly to [26], and [27], we assume that the network between an IoT device and the target device D can be expressed as a M/M/1 queue with arrival rates λ_D^x, λ_D^r and service rate μ_D^x, μ_D^r for transmitting and receiving accordingly.

The utilisation of the network used to offload to D is $\rho_D^x = \lambda_D^x/\mu_D^x$, and the average number of tasks waiting in the network queue for reaching D is

$$L_D^x = \frac{\rho_D^x}{1-\rho_D^x} - \rho_D^x = \frac{(\rho_D^x)^2}{1-\rho_D^x}$$

Similarly for receiving the result back from D, the utilisation is $\rho_D^r = \lambda_D^r/\mu_D^r$ and the average number of tasks waiting in the network queue for reaching the source device is

$$L_D^r = \frac{\rho_D^r}{1-\rho_D^r} - \rho_D^r = \frac{(\rho_D^r)^2}{1-\rho_D^r}$$

Applying Little's Law, the average network queue waiting time of task n offloaded to device D is:

$$W_{n,D}^{x,netQueue} = \frac{L_D^x}{\lambda_D^x} = \frac{1}{\mu_D^x - \lambda_D^x} - \frac{1}{\mu_D^x} \tag{5}$$

and for receiving the result back:

$$W_{n,D}^{r,netQueue} = \frac{L_D^r}{\lambda_D^r} = \frac{1}{\mu_D^r - \lambda_D^r} - \frac{1}{\mu_D^r} \tag{6}$$

We have considered the network for the transmitted data from the IoT device to the target device separately from the one for the response back, since the input and output average data sizes are different, and thus the M/M/1 parameters are different.

3.3. Response time

We refer to response time as the total time it takes for a task n to be transmitted (first two terms of Eq. (7)) and processed (next two terms of Eq. (7)) and the result to be returned back to the IoT application (last two terms of Eq. (7)):

$$T_{n,D} = T_{n,D}^x + W_{n,D}^{x,netQueue} + T_{n,D}^{proc} + W_{n,D}^{procQueue} + T_{n,D}^r + W_{n,D}^{r,netQueue} \tag{7}$$

Of course, in the case that D is the *IoT* device, where the task is not offloaded, but is processed locally on the IoT device itself, then $T_{n,IoT}^x = W_{n,IoT}^{x,netQueue} = T_{n,IoT}^r = W_{n,IoT}^{r,netQueue} = 0$ and $T_{n,IoT} = T_{n,IoT}^{proc} + W_{n,IoT}^{procQueue}$.

3.4. Energy consumption

Let P_{IoT}^{proc} be the average power consumed when the processor of the IoT device is busy, and P_{IoT}^{idle} be the power consumed when idle. Also, P_{IoT}^x and P_{IoT}^r are the power consumptions when the IoT device is transmitting to and receiving data respectively.

If the task n is run locally at the IoT device, then the energy consumption due to that task is the energy consumed by the IoT device to process it:

$$E_{n,IoT}^{local} = P_{IoT}^{proc} T_{n,IoT}^{proc} \tag{8}$$

If task n is offloaded to a target device D other than the IoT device, then the energy consumed by the target device $E_{n,D}^{off}$ is the energy consumed for receiving the offloaded data at D, processing the task at D, returning the result to the IoT device, plus the energy consumed by the *IoT* device $E_{n,IoT}^{off}$ for sending the data to D, remaining idle while waiting, and receiving the result back from D. We assume that the IoT device does not initiate a new task until it receives the result from the previous task and is therefore idle during the offloading response time.

$$E_{n,IoT}^{off} = P_{IoT}^x T_{n,D}^x + P_{IoT}^{idle}(W_{n,D}^{x,netQueue} + W_{n,D}^{procQueue} + T_{n,D}^{proc} + W_{n,D}^{r,netQueue}) + P_{IoT}^r T_{n,D}^r \tag{9}$$

Also, the energy consumed by D receiving the offloaded task, processing it and returning the result is:

$$E_{n,D}^{off} = P_D^r T_{n,D}^r + P_D^{proc} T_{n,D}^{proc} + P_D^x T_{n,D}^x \tag{10}$$

A "selfish" IoT device, which is interested only in its own energy efficiency will aim to minimise $E_{n,IoT}^{off}$. An "altruistic" IoT device that is interested in helping improve overall energy efficiency, will aim to minimise the total E_n^{off}, where:

$$E_n^{off} = E_{n,IoT}^{off} + E_{n,D}^{off} \tag{11}$$

Summarising in a single expression, the energy cost of n in the case of a selfish IoT device is:

$$E_{n,D}^{selfish} = \mathbb{1}[D = IoT] E_{n,IoT}^{local} + \mathbb{1}[D \neq IoT] E_{n,IoT}^{off} \tag{12}$$

Similarly, for altruistic IoT devices, it is:

$$E_{n,D}^{altruistic} = \mathbb{1}[D = IoT] E_{n,IoT}^{local} + \mathbb{1}[D \neq IoT] E_n^{off} \tag{13}$$

3.5. The decision mechanism

The decision is taken at the IoT device based on a weighted goal metric consisting of response time and energy. For selfish IoT devices, this is:

$$G_{n,D}^{selfish} = \alpha T_{n,D} + (1 - \alpha)\gamma E_{n,D}^{selfish} \tag{14}$$

Similarly, for altruistic IoT devices:

$$G_{n,D}^{altruistic} = \alpha T_{n,D} + (1 - \alpha)\gamma E_{n,D}^{altruistic} \tag{15}$$

Note that $\alpha \in [0, 1]$ is a weight denoting the application's preference in minimising response time over energy, defined at each IoT device. For time-critical applications, such as in healthcare IoT, where energy efficiency is not important, α is chosen to be close to 1,

while for applications where energy efficiency is important but time is not, such as in environmental sensing, α is chosen to be close to 0. Also, we use parameter γ for bringing $T_{n,D}$ and $E_{n,D}$ into a mutually comparable range of values (e.g. since the energy of a cloud node could be in range of thousand joules whilst the time could be in the range of some seconds or less).

For a task n initiated at the IoT device, the device chosen to perform its processing is the one that minimises the goal metric for $D \in \{IoT, edge, cloud\}$, denoted here as $C_n^{selfish}$ and $C_n^{altruistic}$ for the two cases:

$$C_n^{selfish} = \underset{D \in \{IoT, edge, cloud\}}{\operatorname{argmin}} \; G_{n,D}^{selfish} \tag{16}$$

$$C_n^{altruistic} = \underset{D \in \{IoT, edge, cloud\}}{\operatorname{argmin}} \; G_{n,D}^{altruistic} \tag{17}$$

This decision is taken by each IoT device for its own applications independently, based on information requests for obtaining the local queuing states and processing times, which are communicated between the IoT and the edge, and the IoT and the cloud. We assume that this communication is given priority and as such the total time it takes to receive and process these request/response packets is negligible in comparison with the processing times even during times of congestion, which is a common assumption in the literature [28].

Note that without loss of generalisation, we considered one edge and one cloud device, because our aim is to determine whether edge or cloud (or local processing) should be chosen, rather than which edge or which cloud device should be chosen. This is not only because the single edge/cloud case is by itself realistic for common smart home or smart office environments, but also because additional edge and cloud devices would not make a difference to the model or the decision mechanism. Again, the device chosen would be the one with the minimum goal value, whether there is one or multiple edge and cloud offloading options.

4. Performance evaluation

4.1. Simulation environment

We performed our simulations by extending the discrete event simulator EdgeCloudSim [29], itself based on CloudSim [30] one of most popular simulators for cloud computing environments [31,32]. EdgeCloudSim allows the simulation of mobile devices which can execute tasks locally and incorporates a networking module for WANs and WLANs or a cellular access network model (3G/4G/5G) between devices. Tasks can be migrated between edge and cloud virtual machines and allows to add a probabilistic network failure model to consider the congestion of the network between the devices. It has been used extensively in the related literature (e.g., [33–35]). EdgeCloudSim does not consider the local device (mobile, or in our case IoT device) in the processing options. Here, we have further extended it to provide support for individual decision making and for considering each task's characteristics and requirements, as well as network conditions, energy consumption and processing times, as opposed to only predefined probabilities. Also, in our extension, the tasks are created by a load generator class and can have different requirements (specifically as task size in Million Instructions (MI) and size of input and output data in MB).

Each simulation starts with an initialisation phase, where a load generator creates a set of tasks based on a Poisson distribution for each application with parameters such as application type, start time, size, input and output data size. These parameters are exponentially distributed random numbers based on the application type. After the list is created, the tasks are sorted based on their start time, and parameters such as the network model between the devices are initiated, where λ_D and μ_D are calculated based on the average data task sizes, different for input and output. A network model is responsible for computing the queuing delay of WLAN connections between IoT devices and the edge and WAN connections between IoT device and cloud for both uploading (task input) and downloading (task output) directions. Finally, the virtual machines for the edge and the cloud are initiated and the simulator's initialisation phase finishes. When the simulation starts, the tasks are served in a chronological order based on their start time and regardless which application they belong to. In each IoT device, an end device manager is responsible for taking the decision of selecting a place to process based on the decision algorithm and the results are saved in log files.

4.2. Experimental setup

Our setup consists of an IoT, an edge and a cloud device. For the processing speed configuration of each device, we use Million Instructions Per Second (*MIPS*), which is supported by EdgeCloudSim for measuring processing times in a reliable way. We adopt it here for reasons of practicality, as is extensively used in the literature (e.g., [36]). We set the processing speed of IoT devices at 50 *MIPS*. An edge device is simulated as a single EdgeCloudSim datacentre, consisting of a host with four virtual machines at a processing speed 1000 *MIPS*. In our experiments, we equate the transmission rates of the devices to the available bandwidth for sending or receiving data. We refer to this as the effective bandwidth. The edge communicates with IoT devices through a WLAN connection of 5*Mbps* effective bandwidth. The cloud is also simulated as a single EdgeCloudSim datacentre with one host and four virtual machines, each of which has processing rate of 10000 *MIPS*. The network between IoT devices and cloud is considered a WAN with 2*Mbps* effective bandwidth. Finally, we have set the packet loss probability $p = 0.11$ and thus the delay factor due to loss $l = \frac{1}{1-0.11} = 1.12$.

In accordance to our energy model (Section 3.4), each device has three modes for energy consumption: processing, transmission and idle. The configurations of the edge and the cloud devices were chosen based on the iFogSim energy profiles [37], where for edge

Table 2
Device specifications .

Device	VMs	Processing Rate (MI) per VM	Effective Bandwidth (Mbps)	Processing Power Consumption (W)	Transmission Power Consumption (W)	Idle Power Consumption (W)
IoT	0	50	–	2.3	1.8	1.2
Edge	4	1000	5	107.339	107.339	-
Cloud	4	10,000	2	103	103	-

and cloud, the power to transmit is practically equal to the power to process. For the IoT devices, we have run an experiment using a Raspberry Pi3 device acting as the IoT device and a Watt's Up Pro meter [38] measuring the actual power consumption when idle, transmitting, receiving and processing a computationally intensive application. Table 2 summarises these specifications for all devices. Although a Raspberry Pi3 device is much more powerful of a typical IoT device today, we anticipate that in the future, IoT devices will become more powerful and thus will consume more power.

For the purposes of our simulations, we have configured four different application types, one for face recognition (such as the smart surveillance cameras announced by NVIDIA [39]), one for healthcare IoT (as one of the in-home therapeutic IoT applications described in [40]), one for IoT intrusion detection (as in our robotic vehicle intrusion detection implementation in [41]), and a hypothetical future indoor monitoring device that will be able to not only visualise a household's sensor data, but also perform advanced analytics to provide predictions and recommendations to its users. We have chosen these four applications so that we can have a range of heterogeneous IoT applications with different specifications (in terms of task size and input/output size) and requirements (in terms of how energy demanding or time critical they are). For instance, the face recognition application is very demanding computationally and also needs to send a relatively large input file (e.g. picture) when offloaded. On the other hand, the computation of the intrusion detection application is moderate and the input file size is small and the result back is very small, since it might consist of a simple yes or no answer of whether an attack was detected or not.

These configurations are summarised in Table 3 in terms of task size, input and output data size and weight α. The value of γ, which brings the time and energy values to a comparable range, was empirically chosen. For selfish-MEDICI, the γ is 0.1 for all devices (Eq. (14)). However, for altruistic-MEDICI (Eq. (15)), if the device is the IoT device then $\gamma = 0.1$, but if the goal G is calculated for the edge or the cloud, then $\gamma = 0.001$, since then the equation also includes the energy of those devices (edge or cloud), which are at a different range than before.

5. Simulation results and discussion

Here we evaluate our proposed Multi-critEria DecIsion support meChanism for IoT offloading (MEDICI). The following are four implementation variants of MEDICI, where we evaluate the differences between selfish and altruistic IoT devices, for the two cases of having different (individualised) and identical (non-individualised) α.

- The selfish individualised version (***MEDICI-SI***). Each IoT device has different α according to each application's needs (Table 3) and the energy to be minimised is the energy of each IoT device.
- The altruistic individualised version (***MEDICI-AI***). Each IoT device has different α according to each application's needs (Table 3), and the energy to be minimised is the total energy of the system.
- The selfish non-individualised version (***MEDICI-SN***). All IoT devices have the same α (here chosen as 0.5) for all applications, and the energy to be minimised is the energy of each IoT device.
- The altruistic non-individualised version (***MEDICI-AN***). All IoT devices have the same α (here chosen as 0.5) for all applications, and the energy to be minimised is the total energy of the system.

We compare the above with five approaches which are representative of the landscape of strategies proposed previously in the literature or used for baseline comparison:

- Choosing the IoT device first (***IoT-first***): If the task is small enough, then it is processed locally until the device's capacity is reached as in [14]. Otherwise, the edge is chosen until its capacity is reached, in which case it is offloaded to the cloud. We have put a restriction on the size of the task for IoT devices (500*MI*) to avoid a situation where, for applications with computation

Table 3
Application specifications and requirements.

Application	Mean Task Size (MI)	Mean Input Size (KB)	Mean Output Size (KB)	Individual Weight (α)
Face Recognition	4000	1500	500	0.1
Healthcare	2500	3000	50	0.9
Intrusion detection	750	100	5	0.8
Indoor monitoring	350	300	300	0.2

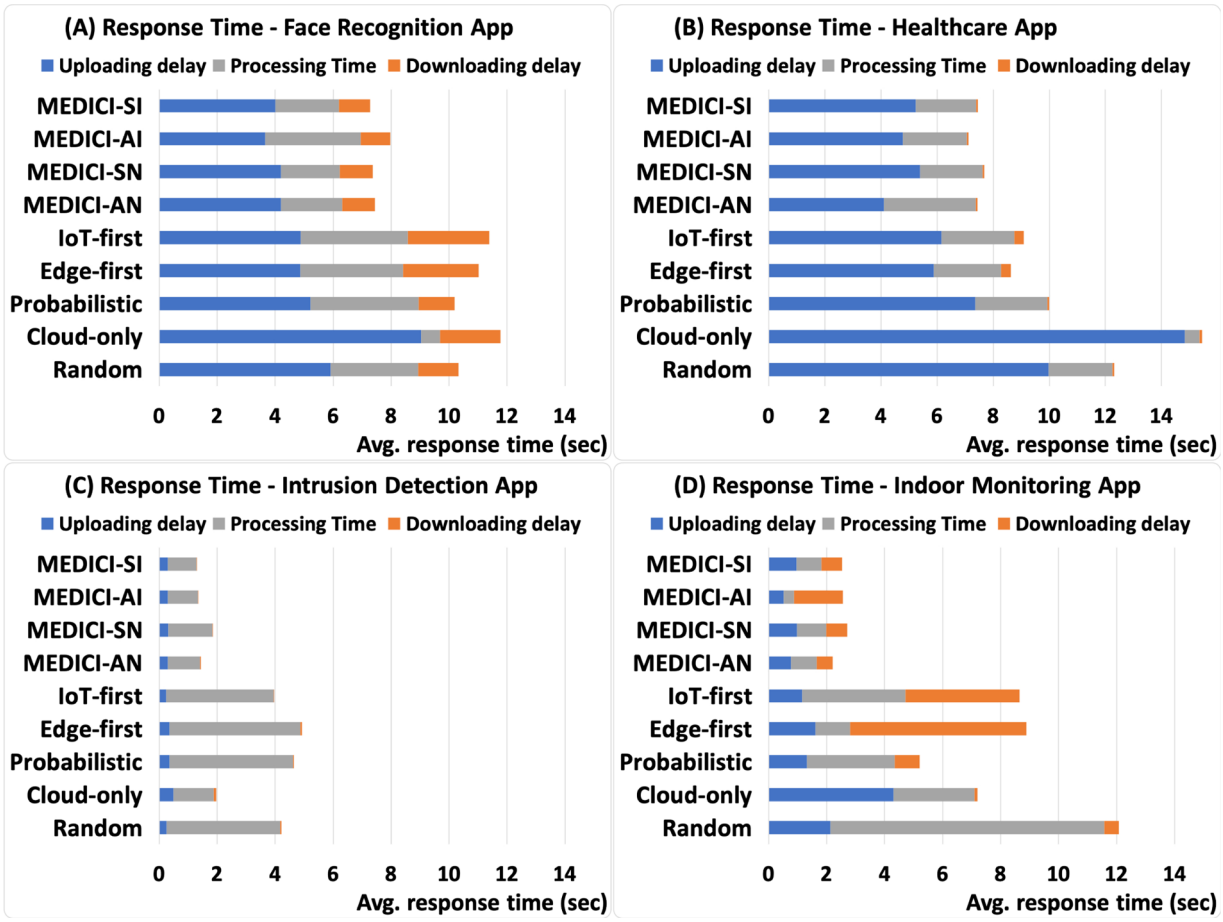

Fig. 2. Average total response time per application.

intensive tasks, the task would not have finished by the end of the experiment.

- Offloading to the edge first (***Edge-first***): The task is offloaded to the edge until its capacity is reached, and then to the cloud. This is the state of play of most commercial IoT devices that use edge computing. The local computation at the IoT device is not considered here.
- Individualised predefined probability (***Probabilistic***): The offloading decision is based on a predefined probability, different for each of the applications. Here we report the result for the set of probabilities that are empirically measured to have the highest performance. Note that our aim is to compare this to our own approach, so, choosing the empirically highest performing configuration of this approach for comparison is meaningful. We have used the same task size restriction as in IoT-first.
- Offloading only to the Cloud (***Cloud-only***): All tasks are offloaded to the cloud. Edge and local computation are not considered. This is the traditional IoT approach used by most commercial IoT applications, and considered for comparison also in [11].
- Random Selection (***Random***): This is a special case of the probabilistic approach, where the probabilities between the choices are equal as in [11].

Each experiment was run for 1 hour of simulated time. The results presented here are the averages of 10 runs for each configuration. Fig. 2 shows the average response time for all tasks of each application, in terms of processing time (including local processing queues, represented in grey), uploading delay (the time transmitting the input data for the task to be offloaded plus the network queuing delays, represented in blue) and downloading delay (transmission of the result (output) data of the offloaded task plus the network queuing delay, represented in orange). Figs. 3 and 4 show the energy consumption at only the IoT device and the total energy consumption at both IoT device and offloading target respectively.

Across all cases, the dynamic decision support offered by MEDICI yields consistently better results for all applications in terms of response time and energy consumption, both at the IoT device and overall.

For the face recognition application, which is characterised by large mean task size, input and output, as well as very low α (i.e., strong preference for energy efficiency over response time minimisation), the lowest energy cost on the IoT device is achieved by the selfish MEDICI-SI and -SN. Comparing to the best-performing non-MEDICI strategy (probabilistic), the energy reduction achieved at

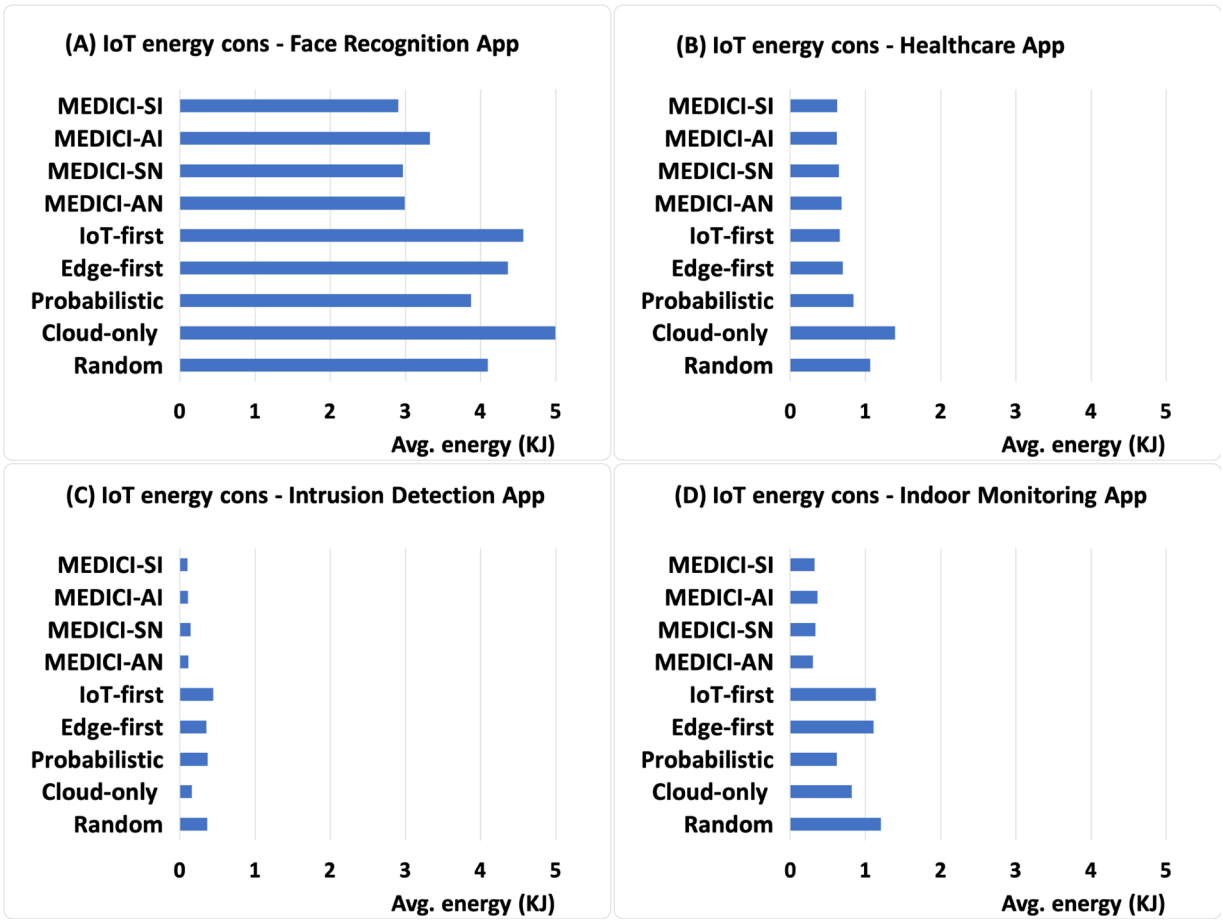

Fig. 3. Average energy consumption of each IoT device per app.

the IoT device was just over 26%. As intended, the lowest total energy costs are achieved by the altruistic mechanisms, MEDICI-AI and -AN, with the biggest reduction achieved by MEDICI-AI of almost 35%. In terms of response time, MEDICI-SI achieved a reduction of 27%.

For the healthcare application, which is characterised by large mean task size, large mean input size, low output size and very high α (i.e., strong preference for response time over energy efficiency minimisation), MEDICI-AI was able to reduce the response time by 14% against Edge-first and the total energy cost against IoT-first by around 10%. The IoT energy cost reduction, however, was not significant. This is expected given the application's very low α.

For the intrusion detection application, which can be offloaded relatively quickly (lowest input and output size) and has a high value of α, the best-performing non-MEDICI strategy is cloud-only. Against it, MEDICI-SI achieves a reduction of over 35% in response time and IoT energy cost. In terms of total energy, again as intended, the best performing variant is MEDICI-AI with a reduction of around 31%.

For the indoor monitoring application, which has the lowest mean task size, significant network traffic when offloaded, and a low value of α, MEDICI-AN achieves the lowest response time (54% reduction against the probabilistic strategy) and IoT energy cost (40% reduction), while MEDICI-AI achieves the lowest total energy cost (68% reduction).

Comparing between the two selfish variants, the individualised MEDICI-SI outperforms the non-individualised MEDICI-SN across all four applications and for all three metrics. This showcases the importance of addressing the individual trade-off preference of each application, as expressed by the parameter α. For the two altruistic variants, the superiority of the individualised MEDICI-AI over -AN is observed only in relation to the energy metrics, as their goal functions differ only in terms of energy and not in terms of response time.

Fig. 5 shows the average percentage of tasks that were allocated to each device per application. It shows that MEDICI appropriately minimises cloud usage for IoT applications that are not computationally demanding, are time-sensitive or return large

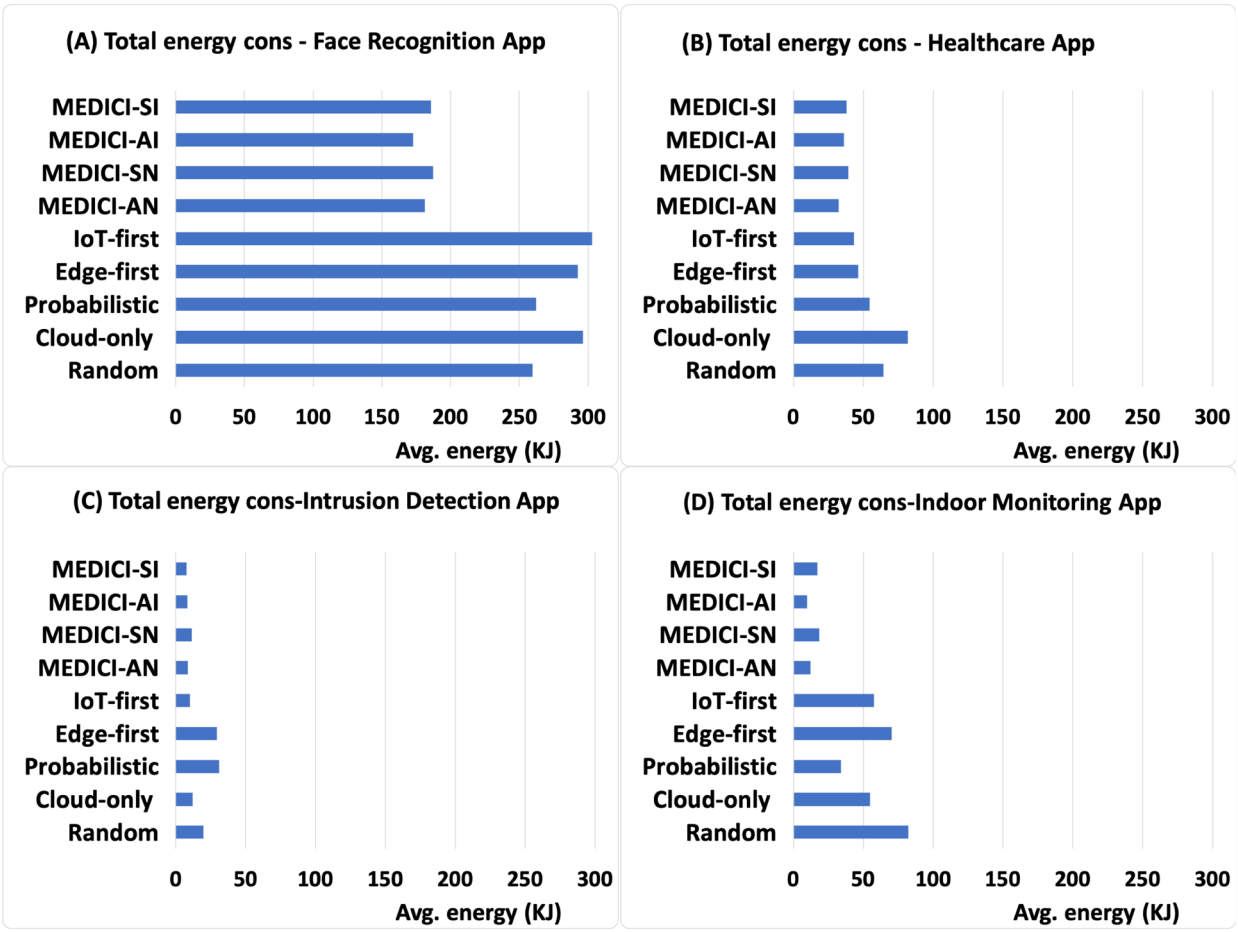

Fig. 4. Average total energy consumption per application.

amounts of data as their output. It also appropriately makes more usage of the cloud when IoT energy consumption is important and a task is computationally demanding.

6. Conclusions and future work

We have proposed a multi-criteria offloading decision mechanism for heterogeneous IoT devices which takes into account not only the execution time of a task in a device but also the time it takes to offload its data and the delays incurred by the network. Depending on the energy-consciousness of an IoT user, the device can choose to minimise its own energy or the total energy of all the devices involved. Our simulations on our extension of EdgeCloudSim have demonstrated MEDICI's effectiveness compared to five offloading strategies across all metrics and applications used, and especially for those with lower mean task and input sizes (intrusion detection and indoor monitoring). They have also demonstrated the importance of the weight α of the application's preference in minimising response time over energy, especially given the often extreme heterogeneity of IoT devices. As intended, the energy-altruistic variants of MEDICI are able to minimise the total energy cost by taking into account the energy cost of the offloading target too, while the selfish variants minimise the energy cost to the IoT device itself. Our evaluation has assumed that the decision is taken locally at each IoT device. However, in a real testbed it might be more realistic to have the decisions taken at the edge device. Of course this will introduce networking overheads incurred for the IoT requests to the edge and the collection of the utilisation data at each device required to take the MEDICI decisions. Another assumption taken in this paper is that the information required for making the decision is (i.e. the local queuing states and processing times) is obtained through high priority request/response packets between the IoT device and the edge/cloud and that we have knowledge of the average end-to-end throughput between the devices. In a real-life situation, the latter could be obtained by regularly probing the network. Of course these communications will introduce some overheads. Additionally, aspects of the cloud, such as its elastic properties, might also be affecting the processing times of a task. Our next steps are to evaluate all these overheads and challenges in different real-world implementations.

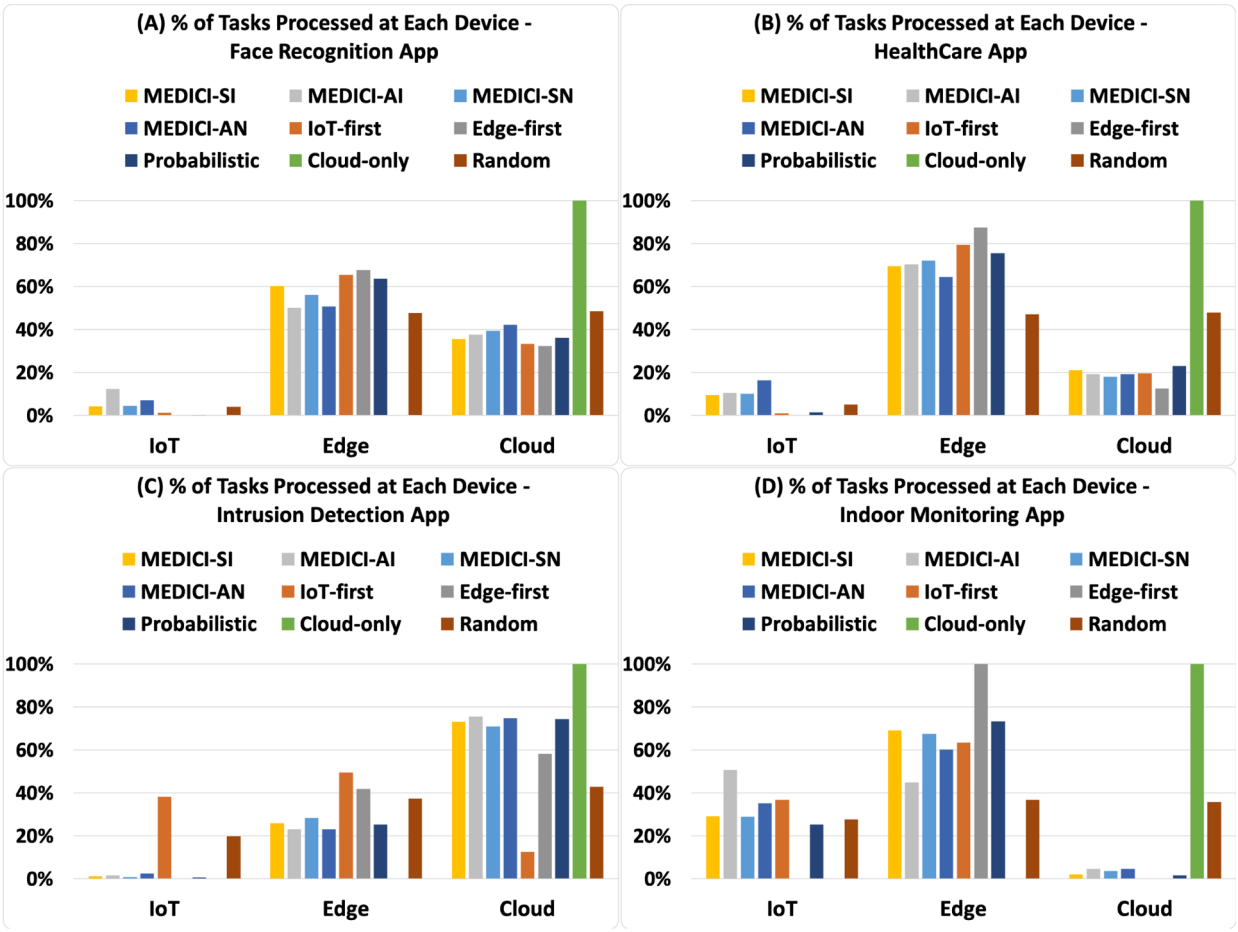

Fig. 5. Percentage of tasks run at each device per application.

Acknowledgement

This work was supported by the H2020 project C4IIOT, under Grant Agreement No 833828.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[2] W. Yu, F. Liang, X. He, W.G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the internet of things, IEEE Access 6 (2018) 6900–6919.
[3] Cisco, Cisco Global Cloud Index: Forecast and Methodology, 2014–2019, (2014).
[4] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81.
[5] S. Sultana, An experimental survey of fog computing and IoT: escalate the cloud to where the things are, IJSRST 3 (8) (2017) 444–450.
[6] P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading, IEEE Commun. Surv. Tut. 19 (3) (2017) 1628–1656.
[7] C. Meurisch, J. Gedeon, T.A.B. Nguyen, F. Kaup, M. Muhlhauser, Decision support for computational offloading by probing unknown services, ICCCN, IEEE, 2017, pp. 1–9.
[8] E. Gelenbe, G. Sakellari, M. D'arienzo, Admission of QoS aware users in a smart network, ACM Trans. Auton. Adapt. Syst. 3 (1) (2008) 4.
[9] S. Ahn, M. Gorlatova, M. Chiang, Leveraging fog and cloud computing for efficient computational offloading, URTC, IEEE, 2017, pp. 1–4.
[10] H. Shah-Mansouri, V.W. Wong, Hierarchical fog-cloud computing for IoT systems: acomputation offloading game, IEEE Internet Things J. (2018).
[11] X. Ma, C. Lin, H. Zhang, J. Liu, Energy-aware computation offloading of IoT sensors in cloudlet-based mobile edge computing, Sensors 18 (6) (2018) 1945.
[12] S. Vakilinia, I. Vakilinia, M. Cheriet, Green process offloading in smart home, IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), IEEE, 2017, pp. 1–5.
[13] Y. Igarashi, M. Hiltunen, K. Joshi, R. Schlichting, An extensible home automation architecture based on cloud offloading, 18th International Conference on Network-Based Information Systems (NBiS), IEEE, 2015, pp. 187–194.
[14] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, J. Henkel, Computation offloading and resource allocation for low-power IoT edge devices, WF-IoT, IEEE, 2016, pp. 7–12.
[15] X. Lyu, H. Tian, L. Jiang, A. Vinel, S. Maharjan, S. Gjessing, Y. Zhang, Selective offloading in mobile edge computing for the green internet of things, IEEE Network 32 (1) (2018) 54–60.
[16] J. Du, L. Zhao, J. Feng, X. Chu, Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee, IEEE Trans. Commun. 66 (4) (2018) 1594–1608.
[17] L. Liu, Z. Chang, X. Guo, S. Mao, T. Ristaniemi, Multiobjective optimization for computation offloading in fog computing, IEEE Internet Things J. 5 (1) (2017) 283–294.

[18] S. Chen, Y. Zheng, K. Wang, W. Lu, Delay guaranteed energy-efficient computation offloading for industrial IoT in fog computing, ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–6.
[19] J.I. Benedetto, L.A. González, P. Sanabria, A. Neyem, J. Navon, Towards a practical framework for code offloading in the internet of things, Fut. Gener. Comput. Syst. 92 (2019) 424–437.
[20] F. Jalali, T. Lynar, O.J. Smith, R.R. Kolluri, C.V. Hardgrove, N. Waywood, F. Suits, Dynamic edge fabric environment: seamless and automatic switching among resources at the edge of IoT network and cloud, 2019 IEEE International Conference on Edge Computing (EDGE), IEEE, 2019, pp. 77–86.
[21] M.G.R. Alam, M.M. Hassan, M.Z. Uddin, A. Almogren, G. Fortino, Autonomic computation offloading in mobile edge for IoT applications, Fut. Gener. Comput. Syst. 90 (2019) 149–157.
[22] A. Alelaiwi, An efficient method of computation offloading in an edge cloud platform, J. Parallel Distrib. Comput. 127 (2019) 58–64.
[23] K. Kumar, Y.-H. Lu, Cloud computing for mobile users: can offloading computation save energy? Computer 43 (4) (2010) 51–56.
[24] G. Loukas, Y. Yoon, G. Sakellari, T. Vuong, R. Heartfield, Computation offloading of a vehicle's continuous intrusion detection workload for energy efficiency and performance, SIMPAT 73 (2017) 83–94.
[25] J. Niu, W. Song, M. Atiquzzaman, Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications, J Netw. Comput. Appl. 37 (2014) 334–347.
[26] F. Mehmeti, T. Spyropoulos, Is it worth to be patient? Analysis and optimization of delayed mobile data offloading, INFOCOM, IEEE, 2014, pp. 2364–2372.
[27] H. Wu, W. Knottenbelt, K. Wolter, Analysis of the energy-response time tradeoff for mobile cloud offloading using combined metrics, ITC, IEEE, 2015, pp. 134–142.
[28] H. Tan, Z. Han, X.-Y. Li, F.C. Lau, Online job dispatching and scheduling in edge-clouds, IEEE INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.
[29] C. Sonmez, A. Ozgovde, C. Ersoy, Edgecloudsim: an environment for performance evaluation of edge computing systems, Second International Conference on Fog and Mobile Edge Computing (FMEC), IEEE, 2017, pp. 39–44.
[30] T. Goyal, A. Singh, A. Agrawal, Cloudsim: simulator for cloud computing infrastructure and modeling, Procedia Eng. 38 (4) (2012) 3566–3572.
[31] A. Beloglazov, R. Buyya, Y.C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, Advances in computers, 82 Elsevier, 2011, pp. 47–111.
[32] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IAAS clouds, Fut. Gener. Comput. Syst. 48 (2015) 1–18.
[33] Q. Zhang, M. Lin, L.T. Yang, Z. Chen, S.U. Khan, P. Li, A double deep q-learning model for energy-efficient edge scheduling, IEEE Trans. Serv. Comput. (2018).
[34] J. Lee, J. Lee, Hierarchical mobile edge computing architecture based on context awareness, Appl. Sci. 8 (7) (2018) 1160.
[35] V. Scoca, A. Aral, I. Brandic, R. De Nicola, R.B. Uriarte, Scheduling latency-sensitive applications in edge computing. CLOSER, (2018), pp. 158–168.
[36] A. Kapsalis, P. Kasnesis, I.S. Venieris, D.I. Kaklamani, C.Z. Patrikakis, A cooperative fog approach for effective workload balancing, IEEE Cloud Comput. 4 (2) (2017) 36–45.
[37] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, ifogsim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Software 47 (9) (2017) 1275–1296.
[38] E.E. Devices, Watts Up Pro, (2009).
[39] K. Pyzyk, Anyvision, Nvidia to Develop Surveillance Cameras with Facial Recognition Tech, (2018).
[40] M.A. Rahman, M.S. Hossain, G. Loukas, E. Hassanain, S.S. Rahman, M.F. Alhamid, M. Guizani, Blockchain-based mobile edge computing framework for secure therapy applications, IEEE Access (2018).
[41] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, D. Gan, Cloud-based cyber-physical intrusion detection for vehicles using deep learning, Access 6 (2018) 3491–3508.