UNIVERSITY OF GREENWICH

FACULTY OF LIBERAL ARTS AND SCIENCES

SCHOOL OF COMPUTING AND MATHEMATICAL SCIENCES

MULTI-CRITERIA DECISION SUPPORT FOR ENERGY–EFFICIENT IOT EDGE COMPUTING OFFLOADING

ALI JADDOA

This thesis is submitted in partial fulfilment

of the requirements of the University of Greenwich

for the degree of Doctor of Philosophy

May 2022

DECLARATION

I certify that the work contained in this thesis, or any part of it, has not been accepted in substance for any previous degree awarded to me or any other person, and is not concurrently being submitted for any other degree other than that of Doctorate of Philosophy which has been studied at the University of Greenwich, London, UK.

I also declare that the work contained in this thesis is the result of my own investigations, except where otherwise identified and acknowledged by references. I further declare that no aspects of the contents of this thesis are the outcome of any form of research misconduct.

I declare any personal, sensitive, or confidential information/data has been removed or participants have been anonymised. I further declare that where any questionnaires, survey answers or other qualitative responses of participants are recorded/included in the appendices, all personal information has been removed or anonymised. Where University forms (such as those from the Research Ethics Committee) have been included in appendices, all handwritten/scanned signatures have been removed.

Student Name: Ali Jaddoa
Date: 17/06/2022

First Supervisor's Name: Dr. Georgia Sakellari
Date: 17/06/2022

Second Supervisor's Name: Prof. George Loukas
Date: 17/06/2022

DEDICATION

To my parents Hamid & Zainab, my two brothers and sister.

You mean the world to me.

ACKNOWLEDGEMENTS

First and foremost I am extremely grateful to my supervisors, **Dr. Georgia Sakellari** and **Prof. George Loukas** for their invaluable advice, continuous support, and patience during my Ph.D. study. Their immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. Also, thanks **Dr. A K M Mahtab Hossain** who was at my initial supervisory team.

I particularly would like to express my gratitude to CARA and the University of Greenwich for the funding opportunity to undertake my studies, and without their support, this would not have possible.

I would also like to thank **Dr. Stelios Timotheou** for his scientific ideas and help.

Finally, I would like to express my gratitude to my parents, my brothers, and my sister. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study. My appreciation also goes to friends for their encouragement and support.

ABSTRACT

Computation offloading is one of the primary technological enablers of the Internet of Things (IoT), as it helps address individual devices' resource restrictions (e.g. processing and memory). 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, closer to the IoT devices. 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 conditions 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.

This thesis presents a dynamic computation offloading mechanism, which improves the performance (i.e. in terms of response time) and energy consumption of IoT devices in a decentralised and autonomous manner. We initially propose the **M**ulti-crit**E**ria **D**ec**I**sion support me**C**hanism for **I**oT offloading(*MEDICI*), which runs independently on an IoT device, enabling it to make offloading decisions dynamically, based on multiple criteria, such as the state of the IoT, edge or cloud devices and the conditions of the network connecting them. It provides mathematical models of the expected time and energy

costs for the different options of offloading a task (i.e. to the edge or the cloud or the IoT device itself). To evaluate its effectiveness, we provide simulation results, by extending the EdgeCloudSim simulator, comparing it against previous families of approaches used in the literature. 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, such as IoT cyber intrusion detection, and the energy consumption not only of the individual IoT devices but also of the system as a whole.

Furthermore, we present an enhancement of our MEDICI mechanism, the **P**robe**L**ess **M**ulti-crit**E**ria **D**ec**I**sion support me**C**hanism for **I**oT offloading (*PL-MEDICI*), which enables MEDICI to operate in real IoT environments without the need for probing or having pre-defined parameters in order to estimate or model the network conditions or the computation capabilities of the different devices involved. This is the first probeless dynamic and decentralised offloading decision support mechanism for IoT environments. The probeless property is achieved by combining lightweight statistical techniques with the concept of age of knowledge (AoK) to allow us to have accurate enough information to use for our estimations.

We provide experimental results performed in a real IoT testbed with three real IoT applications, showcasing that PL-MEDICI outperforms existing techniques in terms of both response time and energy consumption.

Finally, in order to further evaluate our PL-MEDICI mechanism, we formulate a mixed-integer linear program optimisation problem that provides the theoretical optimal centralised solution to our problem. This is used to compare our PL-MEDICI against the theoretical optimum, given the same estimated input. Our results showed that our offloading mechanism is close to the obtained optimal solution in terms of both the response time and energy consumption

LIST OF PUBLICATIONS

Jaddoa, A., Sakellari, G., Panaousis, E., Loukas, G., & Sarigiannidis, P. G. (2020). Dynamic decision support for resource offloading in heterogeneous Internet of Things environments. Simulation Modelling Practice and Theory, 101, 102019.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

AoI Age of Information

AoK Age of Knowledge

AI Artificial Intelligence

AWS Amazon Web Services(AWS)

AR Augmented Reality

API Application Programming Interface

BW Bandwidth

Cisco IOS Cisco Internetwork Operating System

FLOPS Floating Point Operations Per Second

GG Amazon Greengrass

HPE Hewlett-Packard Enterprise

IoT Internet of Things

IP Internet Protocol

IOx Combination of Cisco IOS and Linux OS

ICT Information and Communications Technology

MI Million Instructions

MIPS Million Instructions Per Second

MILP Mixed Integer Linear Program

ML Machine Learning

MEDICI Multi-critEria DecIsion support meChanism for IoT offloading

MEC Mobile Edge Computing

PL-MEDICI ProbeLess Multi-critEriaDecIsion support meChanism for IoT offloading

SDN Software Define Network

Quality of Service QoS

Quality of Experience QoE

VM Virtual Machine

WAN Wireless Area Network

WLAN Wireless Local Area Network

5G Fifth Generation technology

Chapter 1

Introduction

Internet of Things (IoT) is defined as the interconnection of a vast number of devices that can be very different in their scope, size, technologies and protocols used to connect to each other. These devices can range from sensors, actuators, and embedded devices to smart wearable devices, healthcare devices and home appliances. Minerva et al. (2015) claimed that the definition of IoT is often influenced by a particular vision of the proponent entity and what the proponent wants to emphasise. Therefore, the authors in (Minerva et al. 2015) attempted to propose more neutral definitions that differ in detail and encompass the many facets of the Internet of Things.

A common characteristic that they all share, though, is their limited resources. As a result of these resource restrictions, IoT devices typically rely on the storage, communication, and most significantly computation resources of remote cloud infrastructures, such as running computationally intensive artificial intelligence algorithms, most notably for cyber security and image processing. This traditional IoT-cloud approach has worked well in the first years of IoT but is unlikely to be able to meet the requirements of future IoT devices/applications efficiently (Shi et al. 2016, Yu et al. 2018). 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, not to mention the

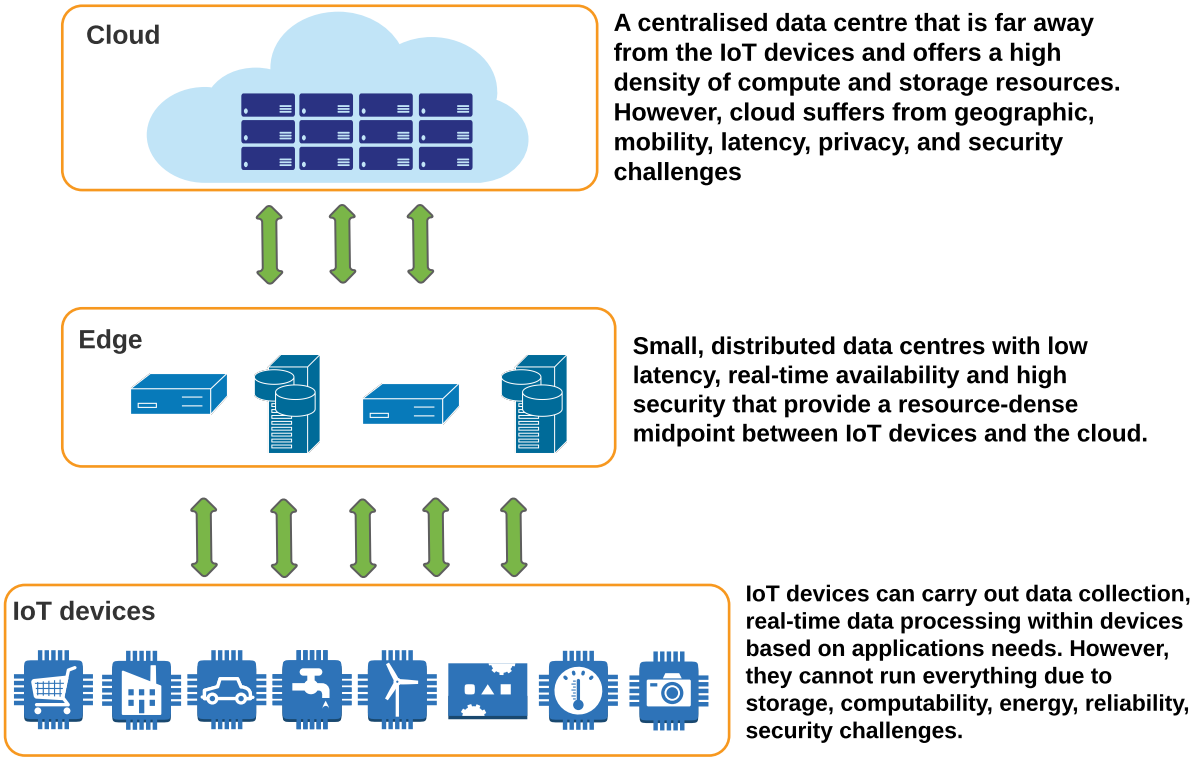
advancement in sensor-networking and digital devices.

Indicatively, the Cisco Global Cloud Index Cisco (2018) has estimated that IoT data will reach 847 ZB per year by the end of 2021, which is 43 times higher than the current total global data centre traffic (19.5.4 ZB) . The generated data, as well as the connected devices, would be heterogeneous which leaves those devices unable to be self-sufficient to manage their associated challenges on their own (Rahmani et al. 2017).

Moreover, pushing the computations and data to the cloud from IoT devices that have limited bandwidth or are connected to the cloud through unreliable networks increases the computational burden of the cloud centre and 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 IoT devices' power capabilities.

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 (Shi et al. 2016, Shi & Dustdar 2016). 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 (Sultana 2017). 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, see Fig. 1.1.

Figure 1.1: Edge-cloud computing scenario



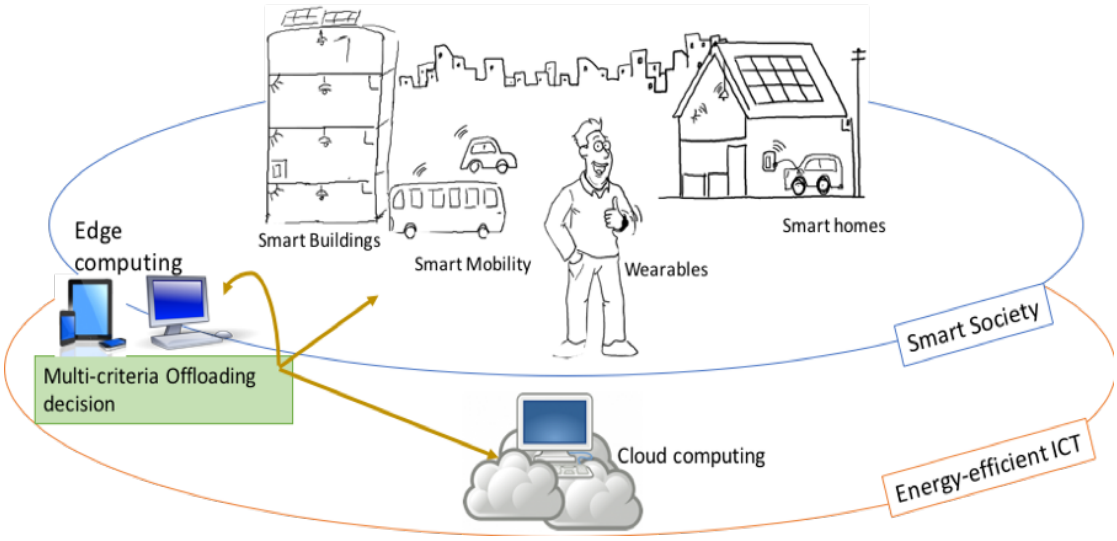
Traditionally, the choice of whether a task should be processed locally, at the edge or at the cloud has been based almost entirely on the original design of the system or on predefined parameters. Chowdhery et al. (2018) point out two extremes that the IoT paradigm relies on. On the one hand, enterprise computing (e.g. IBM, Siemens and ABB, etc.) relies primarily on pushing all the data to the cloud due to its high storage capacity and efficiency as well as to maximise the benefits of the high-cost equipment. On the other hand, critical applications such as healthcare applications, autonomous robots, and self-driving vehicles rely largely on local computation resources due to their severe low latency requirement.

As network and device conditions change rapidly and sometimes unpredictable, this can be highly inefficient. At the extreme, pushing all computation and data of the resource restricted IoT devices to edges and remote clouds through often unreliable net-

works can result in degraded IoT services especially in terms of response time and availability (Shi et al. 2016, Yu et al. 2018). This traditional IoT-cloud approach cannot meet the requirements of the future with more resource demanding IoT applications. Furthermore, choosing the right resources and changing platforms at runtime dynamically and anatomically are not applicable for the majority of IoT applications/devices (Jalali et al. 2019*a*).

Therefore, dynamic decision mechanisms are needed, that take into consideration the various tradeoffs in terms of e.g. performance, communication and energy, when deciding whether edge, cloud or even local IoT devices' resources should be used for performing a task, (see Figure 1.2). These tradeoffs depend on the nature of the IoT task, as well as the current load and the condition of the networks utilised for computation offloading, which change continuously (Jaddoa et al. 2020).

Figure 1.2: Conceptual diagram of edge computing decision making



Minimising energy consumption with a maximum response time value as a constraint will demonstrate this work's usefulness in reducing CO2 emissions in the ICT sector. This will be highly beneficial as, for example, the UK government has specified the ICT sector's energy efficiency as an area of priority in its strategy to reduce CO2 emissions

by 34% below the 1990 baseline levels by 2020 and 80% by 2050 (UK-Government 2011). Similarly, minimising response time with a maximum energy consumption value as a constraint will enable edge computing to dramatically reduce response time by ensuring that processing-heavy applications which run normally in a remote cloud can now run efficiently in a local or quasi-local servlet. With the expectation that connected devices will reach over 28.5 billion by 2022 according to Cisco (Cisco 2018), generating very unpredictable energy and response time demands, a decision support module that will optimise either can become an enabling technology.

In this thesis, we propose a dynamic edge computing offloading decision mechanism which decides where an IoT task should be processed, by estimating both the response times to send and process a task in an external device, as well as the energy consumed for the processing of that task. We have evaluated our proposed algorithm, by comparing it to existing solutions and to an optimal centralised solution.

1.1 Research questions

In this section, we identify the research questions that this thesis is trying to answer:

1. Can taking into account real-time network and processing conditions improve IoT offloading performance in terms of energy and time savings?

2. What is the theoretical optimal performance that can be achieved by an ideal centralised IoT offloading mechanism with full a priori information?

3. Can a decentralised decision mechanism be developed so that it can reach energy and time saving performance that is comparable to the theoretical centralised optimal?

1.2 Aim and objectives

1.2.1 Aim

The aim of this work is to improve dynamic IoT computation offloading by reducing the processing and networking overheads that are inherent in the current state of the art.

1.2.2 Objectives

1. Provide a comprehensive review of the current state of the art in resource offloading that is applicable to IoT.
2. Develop a mathematical model for estimating the response time and energy consumption expected as a result of offloading an IoT task to an edge or cloud device.
3. Develop a decentralised offloading decision mechanism that avoids the overheads caused by probing or resource demanding estimation algorithms used in the existing state of the art.
4. Formulate an optimisation problem to describe the theoretical optimal centralised offloading allocation in any given scenario for the purpose of comparing to the decentralised mechanism developed.
5. Develop simulation environment for IoT offloading evaluations
6. Develop a testbed for IoT offloading experimentation in realistic conditions.

1.3 Contributions

The key contributions to knowledge of this thesis are:

1. We have developed a taxonomy of resource offloading mechanisms that are applicable in IoT, including the resource, infrastructure, implementation, further enhancement technique and timing characteristics in the existing literature.

2. We have proposed a response time and energy consumption model for edge IoT, which takes into consideration the application, device and network characteristics of the system.

3. We have developed a multi-criteria offloading decision support mechanism (MEDICI) for heterogeneous IoT devices, which dynamically and autonomously decides where an IoT task should be processed. For this we have modelled the response time and the energy consumption of IoT tasks in different devises. We also evaluated our mechanism at both the individual (selfish IoT devices) level and at the level of the system as a whole (altruistic IoT devices) and for different applications such as cyber intrusion detection, face recognition and health monitoring.

4. We have extended the EdgeCloudSim simulator to support dynamic offloading decisions to validate our model and compare it against previous approaches used in the literature.

5. We have introduced the first probeless dynamic and decentralised offloading decision support mechanism (PL-MEDICI) for IoT environments. The probeless property is achieved by combining lightweight statistical techniques with the concept of age of knowledge (AoK).

6. We have evaluated our PL-MEDICI mechanism on a real testbed, composed of Raspberry Pis connected to edge and cloud servers, and showed its performance compared to the existing strategies in the literature.

7. We have presented a optimisation formulation that provides the theoretically optimal centralised solution, based on a priori information by formulating and solving a

mixed-integer mathematical programming (MILP) problem and simulating it using MATLAB and Gurobi optimiser. Our comparison shows that PL-MEDICI mechanism achieved a performance close to the theoretical centralised optimum.

1.4 Thesis structure

This thesis is organised as follows. In Chapter 1, we present the introduction, research questions, aims and objectives.

In Chapter 2, we survey a large variety of different IoT resource offloading works presented in the literature, and we propose a comprehensive taxonomy based on the type of resources to be offloaded, the infrastructure and the way the offloading is implemented. We also look into offloading from the industry perspective, by reviewing industrial IoT resource offloading applications.

In Chapter 3, we firstly look into the state of the art on decision making in computational offloading. Then, we describe our proposed mathematical model for dynamic computation offloading. After that, we survey a few existing simulation toolkits in the edge-cloud computing context before present our simulation environment by describing the extended EdgeCloudESim simulator and then we present our simulation results.

In Chapter 4, we look into the state of the art and most popular solutions of dynamic decision support systems for IoT offloading. We present an enhancement of our initial offloading model to be able to operate in real environments. We propose a dynamic and decentralised offloading decision support mechanism for IoT environments that is probeless (PL-MEDICI). Then, we present the configuration of our real-testbed and present the evaluation of PL-MEDICI mechanism in the real-world setup and compare it to several offloading strategies proposed in the literature, and for three different applications.

In Chapter 5, we formulate and solve a mixed-integer linear programming problem that provides the centralised theoretical optimal offloading decisions. Then, we compare

the optimal solution to our PL-MEDICI mechanism.

Finally, in Chapter 6, we conclude and present the key contributions of this research project and our future work.

Chapter 2

Literature review on resource offloading in the Internet of Things (IoT)

2.1 Introduction

Resource offloading is commonly used in application areas where there are limited resources available, originally for mobile computing and currently in the Internet of Things (IoT), where processing, memory or storage can be severely constrained. Previous surveys on computation offloading have focused on mobile computing, and some others on in mobile edge computing (Mach & Becvar 2017) and opportunistic offloading (Xu et al. 2018).

However, 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. An exception is the survey carried out in (Mahmud, Kotagiri & Buyya 2018), which includes related work in fog computing, but only briefly mentions computation offloading. Here in this chapter, we address this gap in the literature by systematically surveying the existing work in IoT offloading and proposing a comprehensive taxonomy that leads to a better understanding of resource offloading. We also look into

offloading from the industry perspective by reviewing industrial applications relevant to IoT offloading.

2.2 The proposed IoT offloading taxonomy structure

We start by asking the following questions:

- What resource restriction is addressed by offloading?
- What infrastructure is the target of offloading?
- How is offloading carried out?
- When and how often is offloading carried out?

Based on the above questions, we have constructed the taxonomy summarised in Fig. 2.1.

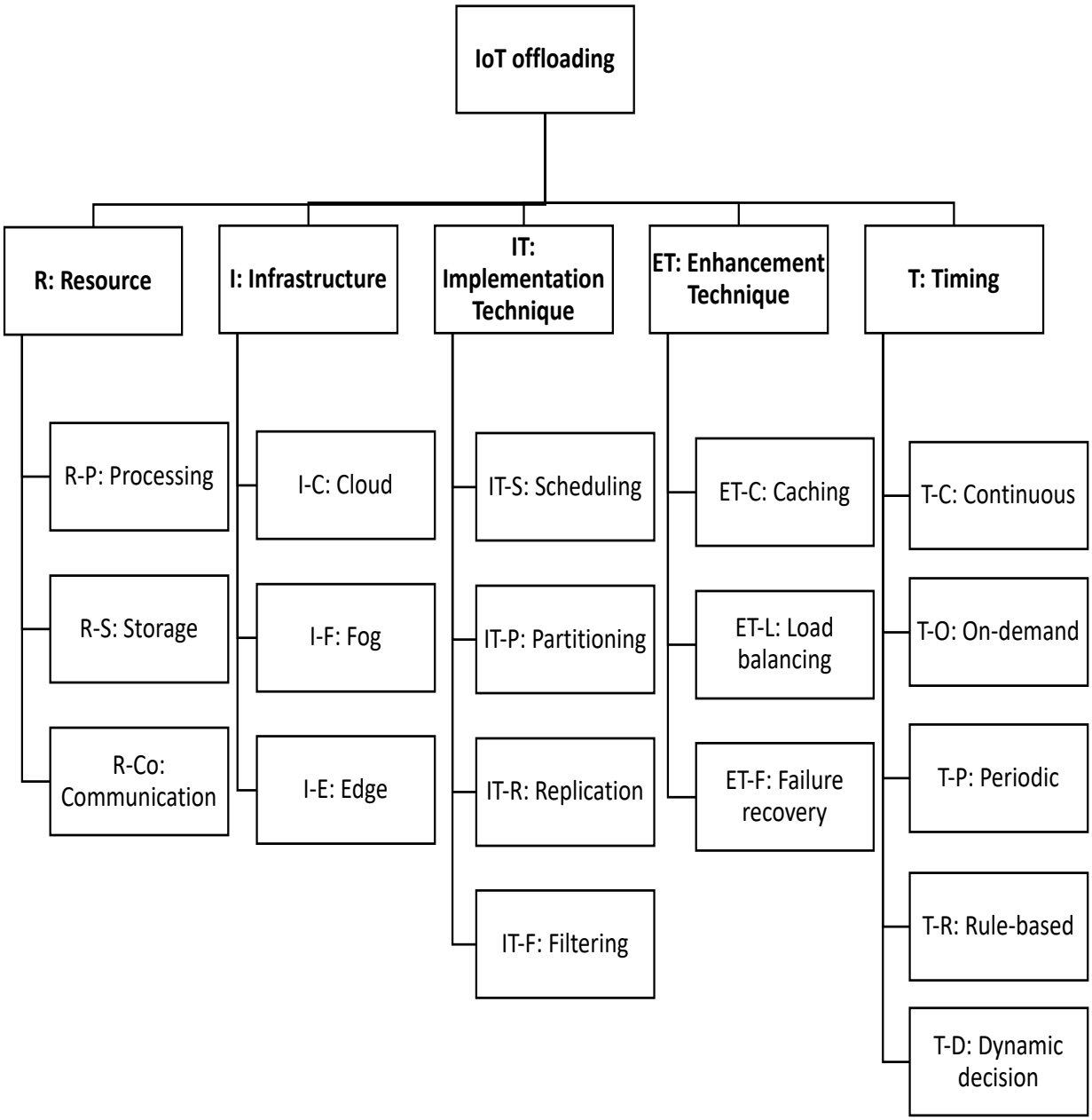
2.2.1 R: Resource

IoT systems are typically resource-restricted in terms of storage, memory and/or processing. The nature of offloading depends on the resource that is limited.

2.2.1.1 R-P: Processing

Offloading the computational processing of IoT devices to an external infrastructure, such as a cloud or an edge device, may offer access to more powerful processing resources (Yu et al. 2018, Botta et al. 2016) and may also lead to significant energy saving (Botta et al. 2016, Fox et al. 2012, Terzopoulos & Karatza 2016, Rao et al. 2012, Suciu et al. 2013). However, it can also introduce additional network delays and longer response times (Finnegan 2013), as well as additional security and privacy concerns.

Figure 2.1: IoT resource offloading taxonomy



There are clear trade-offs between response times and energy savings as seen in (Yoon et al. 2016, Loukas et al. 2017) and determining whether to run the processing functions locally or offload them is needed (Yu et al. 2018, Lin et al. 2015*a*) (see section 2.2.5).

2.2.1.2 R-S: Storage

Storage has been the resource that is very commonly offloaded since the very beginning of IoT. For instance, a small IoT surveillance camera may have the processing power to carry out basic video processing tasks, such as detecting movement, but is unlikely to be able to store onboard months worth of footage captured. To address the storage resource limitation (as well as for other reasons, such as the convenience of remote access), IoT cameras commonly transmit the data to a remote cloud provided by their manufacturer. For applications though, such as real-time video analytics, where time is important and continuous synchronisation between the IoT devices and the cloud is needed, a multi-edge solution where storage is also handled closer to the devices is proposed in (Ananthanarayanan et al. 2017).

Offloading storage can also lead to more resilient IoT systems, for example, through replication at more than one location (e.g. multiple cloud servers or multiple edge devices) (Yu et al. 2018, Chang et al. 2008) allowing storage redundancy and thus lower likelihood of data loss (Zhao et al. 2016).

2.2.1.3 R-Co: Communication

In (Kemp et al. 2011), the authors have proposed the offloading of communication for communication-intensive applications, such as Internet information monitoring applications which constantly poll information from web sources. Examples include Rich Site Summary (RSS) readers, social network applications, sports score services, weather information widgets, traffic information widgets, etc. In communication offloading, it is

a cloud server that constantly polls the web sources at a rate frequent enough to allow maximum accuracy. Communication with the IoT device only occurs when specific conditions are met, such as when a specific value has changed rather than at the same frequency as the polling, thus saving energy and bandwidth.

2.2.2 I: Infrastructure

Depending on the performance and other requirements of a given application, the offloading infrastructure may relate to cloud computing, edge computing or fog computing (see table 2.1).

2.2.2.1 I-C: Cloud

Cloud and IoT are closely interconnected and complementary to each other. IoT gives cloud infrastructures the opportunity to extend their scope into dealing with real-life applications (e.g. eHealth, smart cities, etc.) in a more dynamic and distributed environment. At the same time, cloud computing offers IoT devices resources and computation capabilities that compensate for the technological barriers of their resource restrictions (Lee et al. 2010), DaCosta (DaCosta 2013) though argues that cloud offloading is not sufficient anymore since IP protocols overburden the IoT devices by requiring too much bandwidth from energy-constrained devices. The introduction of Edge computing can be attributed to this weakness in cloud offloading.

2.2.2.2 I-E: Edge

The traditional IoT-cloud approach may not be able to efficiently meet the requirements of future IoT applications (Shi et al. 2016, Yu et al. 2018). Pushing the computation and data to the cloud from IoT devices that have limited bandwidth or are connected to the cloud on unreliable networks can increase communication latency. Moreover, due

to IoT devices' power limitations, it is vital to balance power consumption by reducing transmission and scheduling computation based on the power capabilities of devices. As such, the concept of edge computing was introduced, where the data is processed or sometimes produced at the edge of a network rather than being uploaded directly to the cloud. 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 (Yu et al. 2018). However, its fundamental weakness is that unlike cloud, which is geographically unconstrained and processing or storage can be carried out anywhere, edge devices are by definition in proximity to the IoT devices.

In Edge IoT, the edge devices could be any type of device from a computer or a server with powerful resources to a Raspberry Pi or a mobile phone with less powerful ones. IoT devices can request data from the edge as a consumer or send data to the edge as a producer. Depending on the resources of the edge, additional services such as data caching/storage, data processing and management can take place at the edge (Shi et al. 2016, Frankston 2016). The paradigm of edge computing though invested into separate devices to handle the data and either process it locally or sending further along to the cloud. The configuration and interconnection of such devices could get more complicated as the IoT devices and the data produced increase. Therefore a concept of using existing network devices as edge devices was introduced. These devices would be able to interconnect into a separate intermediate layer (fog) to allow more straightforward configuration. This has evolved into what is typically referred to as Fog IoT, which is described next.

2.2.2.3 I-F: Fog

Although the terms fog and edge are often used interchangeably, currently the terms have become more distinguished, with fog being used for the combination of edge nodes

Table 2.1: Infrastructure specs

Device Type	**Location**	**Specs**	**Example Usage**
IoT	Local	-	Sensing and actuating
Edge	Close proximity	4 cores, 4 GB	Rapid computation, filtering or small
Fog	Constrained	Six core, 8 GB per core	Moderate processing or storage
Cloud	Unconstrained	Multi-core, 16 GB per core	High processing or large storage

with the cloud (Varghese et al. 2017, Bonomi et al. 2012). Fog IoT involves the IoT devices connecting to a middleware layer (proxy), which is then connected to the cloud. Therefore, the middleware acts as a bridge between the IoT device and the cloud, adding an extra layer to the communication (Stavrinides & Karatza 2019). A common example of such a setup is when a mobile phone acts as a gateway running an IoT application that collects data from the device and sends it to the cloud for processing.

This middleware though might not be a single device but rather a group of usually network devices, such as routers and switches. Through virtualised software, their role is enhanced beyond the communication aspect enabling them to perform computational functions (i.e., pre-processing) and improve not only performance but also other aspects, such as security. For example, in (Wortmann & Flüchter 2015, Al-Fuqaha et al. 2015, Yu et al. 2018) this extra layer is used to reduce the size of data (i.e. pre-processing) through aggregation (Nazmudeen et al. 2016)and filtering (Lin et al. 2015*a*, Zao et al. 2014, Craciunescu et al. 2015) of the data that is then forwarded to the cloud. After the appropriate processing of the data at the cloud, the information is sent back to the IoT device through one or more fog devices, not necessarily the same as before. Bonomi Bonomi et al. (2012) though argue that this procedure affects the quality of service (i.e. response time) and energy consumption due to network variability and high computation demands, making the choice of which of the multiple layers that handle the workload a significant challenge.

2.2.3 IT: Implementation Technique

This refers to the offloading technique, which is commonly based on scheduling or partitioning.

2.2.3.1 IT-S: Scheduling

With increasing the number of applications and devices that are connected within the IoT paradigm, forcing all processing tasks to the cloud would lead to high latency and a high burden on communications costs. Thus, fog/edge computing has been proposed in order to extend the cloud to nearby resources(Karatza 2021, Pham & Huh 2016, Lin et al. 2017). To this end, the question here is how to manage the execution of offloaded tasks whether partial or full between these entities (i.e. local device, fog/edge, and cloud). Task scheduling is being used in order to maximise the number of executed tasks by considering the limited bandwidth and service capabilities (Tychalas & Karatza 2020). Thus, the aim of scheduling is to schedule tasks with respect to their deadlines and minimising the schedule-makespan (Pham & Huh 2016).

Broadly, scheduling offloaded tasks have been studied extensively to propose an algorithm or strategy to carry out the procedure within a set of heterogeneous entities such as (Li, Ota & Dong 2018, Oueis et al. 2015, He et al. 2017, Intharawijitr et al. 2016, Stavrinides & Karatza 2020). Li, Ota & Dong (2018) have proposed an offline and online scheduling algorithm to address the difference in size of intermedia data of various deep learning models with the guarantee of QoS. In (Oueis et al. 2015) task scheduling was studied in multi fog stratum by which schedule the tasks based on the computational resource of a fog node. Two steps are being carried to run the scheduling: the first step is the allocation of computational resources for each node through an order list of users associated rely on an objective. While the second step is building computation clusters to process them. Additionally, Intharawijitr et al. (2016) presented a task scheduling al-

gorithm that aims to find a mapping between the offloaded tasks and compute servers in order to reduce the probability of task blocking with respecting the constraint of latency on the end-user side. In doing so, three policies have been proposed 1) random approach (i.e. a choosing a fog node randomly to execute a task), 2) lower-latency (i.e. selecting a fog node with lower latency based on the system state), and 3) while in this policy capacity and attributes of an arrival task are targeted in order to select from a candidate list a fog node with maximum remaining resources. They proved that the second policy would lead to the lowest blocking probability.

Furthermore, He et al. (2017) utilised a fog platform where the transcoding workload is offloaded using multiple regional data centres formed as a novel fog-based transcoding framework for crowdsourced livecast service platforms. In the fog border, a transcoder is employed to transcoding the live stream into various quality forms based on the network availability and receiver characteristics. In doing so, a scheduling algorithm is introduced to select the most adequate node for executing a transcoding task. The study shows that the hybridisation proposed system offers better streaming performance with low cost in terms of energy consumption and compunction costs.

2.2.3.2 IT-P: Partitioning

A workflow may be partitioned and sent to different locations for execution. Here, the challenge is in deciding the layer that will handle the workloads and the number of tasks that will be allocated, with Shi et al. (2016) proposing either even partitioning on each layer, or each layer completes tasks as much as possible. Their criteria for an optimal allocation strategy were latency, bandwidth, energy and cost. Offloading the whole of a task/application from an IoT device may be inefficient in terms of response times. Wu et al. (2018) argue that offloading some tasks of a process may lead to high communication cost or may not always be possible because they might require local resources. Therefore, offloading only parts of an application could lead to better response times. In

partitioning, we identify two kinds of tasks, the ones that if offloaded, their performance will be enhanced (offloadable). And the ones that cannot/should not be offloaded (unoffloadable) and should run locally, either because they need access to local components (e.g. camera, GPS etc.) or offloading them would result in high delays and energy consumption (Cuervo et al. 2010). How and at what granularity this partitioning will happen depends mostly on the application.

Generally speaking, partitioning for offloading in IoT can be divided into two major steps. Firstly, we partition an application based on the chosen level of granularity, e.g. the smallest computation unit. This can be for example objects (Niu et al. 2014), classes (Abebe & Ryan 2012), threads (Chun et al. 2011) or methods (Cuervo et al. 2010). Secondly, we identify which of them are offloadable and which are unoffloadable, and we offload based on the available resources using several techniques such as weighted graph (Hassan et al. 2015, Wu et al. 2016), linear programming-based technique (Ra et al. 2012), and Cyber Foraging technique(Balan et al. 2007).

Partitioning can be carried out dynamically or statically. In static partitioning, computation components are partitioned manually beforehand (Hassan et al. 2015), depending only on the characteristics of the job (e.g. size, computational intensity etc.) and not on the current state of the system, such as network conditions and infrastructure (edge, fog, cloud) availability (Wu 2018, Elazhary 2017). Although static partitioning has low overhead in terms of execution time, since it is applied only to applications/tasks that have a predefined and constant number of partitions, it can only be applied to applications that have parameters that are known in advance or can be accurately predicted (Wu et al. 2016). Therefore, using static partition is inefficient in IoT offloading where workloads and resources such as energy and bandwidth constantly change (Ali & Lhoták 2012, Yang et al. 2013, Sinha & Kulkarni 2011).

The dynamic partitioning is being carried out at runtime and in a context-aware manner (Hassan et al. 2015). The decision is based on the system's state, taking into ac-

count the status and heterogeneity of the environment, e.g. the network conditions (e.g. bandwidth) (Niu et al. 2014), and latency and the computational availability (Cuervo et al. 2010, Kosta et al. 2012). Many optimisation algorithms have been proposed for dividing the workflow into partitions. Wu (Wu 2018) estimates the computation times, communication costs and energy consumption at runtime, constantly deciding how and which part of the application should be offloaded.

There are also methods that combine static and dynamic partitioning. For example, Chun et al. (2011) combine static partitioning of specific tasks of an application and dynamic profiling of others in order to optimise the energy consumption and time latency based on the current system status (e.g. CPU utilisation, energy consumption and network conditions). However, the implementation is restricted since bootstrapping is needed with each new application/task. Also, in (Hassan et al. 2014) relationships among the factors that influence partitioning are characterised in advance and then a classifier collects information about a method and dynamically calculates the size of its arguments while providing feedback for self-learning and decision adaptation.

Partitioning can also be considered in terms of the location that the application/task will be offloaded. Therefore, the decision is based on the computation capabilities of the execution location (e.g. fog, edge and cloud) rather than the task itself (Ketykó et al. 2016, Min et al. 2017, Lyu et al. 2018, Chen, Shi, Yang & Xu 2018). For example, Min et al. (2017) have investigated offloading the computation of IoT devices in a dynamic, multiple edge environment. They used Reinforcement Learning-based computation offloading to choose between edge devices to execute an application (i.e. partitioning applications among edges) based on bandwidth usage and the current battery level of the IoT devices along with the predicted energy that can be harvested. They showed that after a learning period, an optimal offloading policy can be achieved.

In (Lyu et al. 2018), a selective offloading technique is presented to reduce the signalling overhead of edges and the energy consumption of IoT devices, using an inte-

grated architecture of IoT, Edge and cloud. The authors have used a lightweight request and admission framework to carry out partitioning by encapsulating the latency requirements of each device in the offloading requests. This decouples the dependency of task partitioning among different devices and eliminates the need for coordination between devices and edge servers.

Chen, Shi, Yang & Xu (2018) propose a delay-aware task graph partitioning algorithm (ThriftyEdge) to distribute tasks/applications between IoT devices, edge devices, which they refer to as "helpers" and cloud servers. It aims to optimise the offloading decision by utilising resources efficiently through partitioning computation workload subject to completion time constraints. A direct acyclic graph is used where the nodes represent the required computational resources for an application/task (e.g. CPU) while the edges represent the transmitted data

In this thesis, we decided to go with task-level offloading granularity, which is common for IoT applications (Khoda et al. 2016*a*, Benedetto et al. 2019*a*, Jalali et al. 2019*b*, Lin et al. 2015*b*, Zhao & Zhou 2019).

2.2.3.3 IT-R: Replication

In contrast to partitioning, task replication is about offloading the same task (rather than different tasks) to multiple systems. Generally, this increases the overheads considerably in terms of processing resources, but can be very helpful in application areas where meeting completion time requirements is more important. An example explored in detail by Jiang et al. (2018) relates to vehicular cloud computing, where tasks are replicated across multiple vehicles to minimise the probability of violating a task's deadline. This is a challenging problem because of the uncertainty introduced by vehicle movements. So, the authors have formulated it as a finite-horizon sampled-time Markov decision problem and have obtained the optimal policy by value iterations. Their proposed balanced-task-assignment policy was proven optimal and with a clear structure, always assigning

the task with the minimum number of replicas. They have also derived a tight closed-form performance upper bound for this policy. Interestingly, they have shown that the optimum vehicle speed to minimise the deadline violation probability is greater than the critical vehicle speed, which maximises traffic flow efficiency as derived by traffic theory. Moreover, Benedetto et al. (2019*b*) propose a concurrent mode in which a task is being executed in all available execution resources simultaneously and only the first results are considered with ignoring the rest. Offloading all tasks does not only increase the utilisation of the bandwidth link and computational resources but increases the energy consumption of end-devices and servers.

2.2.3.4 IT-F: Filtering

In the applications that deal with large data size, such as video and voice, the transmission time can be reduced with higher bandwidth. Basically, the higher bandwidth can exist within a short distance. To be precise, in the smart home a higher speed transmission way can be employed, such as Wi-Fi, to transfer the data from IoT devices to the gateway. Also, the reliability of transmission as the distance of transferring data will be short. Additionally, since the edge has limited computation capabilities, the complex tasks will be offloaded to the cloud. The transfer of massive data will use enormous network bandwidth, which will lead to several issues, including the loss of packets and delay (Loukas et al. 2018). Thus, filtering has been suggested by several studies such as (Shi et al. 2016) and (Yu et al. 2018) in order to reduce the size of data to be offloaded to fog (filtration would be carried out at the local device) or cloud (filtration would be carried at the local devices or fog node), by having the basic and initial operations (such as compression and extraction of features) carried out by the end device or at the edge before forwarding the outcomes to the fog or cloud. This results in less bandwidth, more efficient processing, and shorter response times.

Technically, Zao et al. (2014) proposed a framework that employed three tiers of

distributed devices(cloud, fog, and end-device) in order to form an EEG-based Brain-computer interface system (EEG-BCI). It tries to distribute the processing of tasks between the three tiers in order to reduce the raw data, decrease the bandwidth, and reduce the time of response. Specifically, in front-end (dry-electrode EEG headsets), a powerful processor is embedded in the sensors for pre-processing data onboard. While in the near- end, an ad-hoc computing proxy software is utilised to carry out the majority of tasks at the edge, also offloading workload to the cloud in case of complex tasks. Therefore, reducing the size of data through layers is shown as an efficient way of reducing the response time and bandwidth utilisation and producing accurate results in terms of human-computer interactions.

In addition, Dubey et al. (2015), proposed a data mining framework called Fog Data mining. It evaluates the raw sensed data from IoT devices (sensors) in order to find unique features to transmit them to the cloud (i.e. filtration). Two disorders have been taken to evaluate the proposed system (motor disorders and cardiovascular). The result shows that using fog platform for reduction data before the core processing leads to efficient outcomes in terms of response time and energy consumption.

2.2.4 ET: Enhancement Technique

2.2.4.1 ET-C: Caching

Elbamby et al. (2017) have proposed a method for reducing the response times for offloaded tasks by implementing a proactive edge caching mechanism. This is because caching joint data and popular tasks on an edge device can reduce the network traffic overhead and end-to-end latency. In fact, it could be argued that edge caching is a form of offloading that combines storage offloading (R-S) with processing offloading (R-P). To this end, the authors have provided an efficient distribution matching algorithm to find a stable matching between IoT devices and cached tasks in the edge layer. In addition,

due to constraints of IoT resources (e.g. power, and bandwidth), caching some contents at the IoT devices and avoiding unnecessarily caching would save battery power and wireless bandwidth. However, care must be taken, especially in sensor-related applications, to ensure that caching will not lead to the use of outdated sensor readings (Zhang et al. 2015).

2.2.4.2 ET-L: Load balancing

Yu et al. (2018) have argued that the quality of service requirements of offloaded storage (R-S, section 2.2.1.2) can be met with a load balancing scheme that distributes the traffic in the network among different links. In (Lin et al. 2017), the authors have proposed to improve IoT storage offloading to an edge/fog device by allocating resources based on a priority-based user satisfaction function rather than just the volume and whether there are enough resources on the edge to accommodate the service.

Additionally, Ananthanarayanan et al. (2017) have proposed a smart traffic-scheduling algorithm to reduce the offloaded content based on a content-aware uploading strategy, as utilising storage balancing to offload the storage in an edge/fog-based network time and capacity of storing can be improved.

Lyu et al. (2018) have had a different take on this challenge. To address the scalability problem of offloading from massive numbers of IoT devices in mobile edge computing, they have proposed a lightweight request and admission framework, which requires no coordination among devices. It is operated at the IoT devices and computing servers separately by encapsulating latency requirements in offloading requests. The signalling overhead (and corresponding energy consumption on each IoT device) is reduced via a selective offloading scheme, whereby the devices are allowed to be self-nominated or self-denied for offloading. Evaluated in simulation, this approach has been shown to satisfy the latency requirements of different services while also reducing the IoT devices' energy consumption.

2.2.4.3 ET-F: Failure recovery

Yu et al. (2018) have also argued that adding a data failure detection and recovery mechanism (for packet loss, power issues, noise, etc.) can enhance storage offloading, especially where massive data flows are involved. Storing and retrieving data reliably and accurately enhances the QoS. Reliability can be improved by either duplicating data in more than one edge/cloud devices or by simply adding an additional check on the status and availability of the nodes before offloading a resource.

In (Dimakis et al. 2011, Chang et al. 2008) a redundant storage server is deployed in cloud-based systems to enhance the reliability of offloading, and a simple periodic heartbeat or pinging is used to identify the availability of the server nodes.

Chang et al. (2008) and Zhao et al. (2016) propose offloading sensitive IoT data to more than one node to enhance reliability by replicating it to an edge/fog computing-based distributed storage system.

2.2.5 T: Timing

In terms of its timing, offloading may be on-demand, rule-based, periodic or continuous, as prescribed by the different applications' needs and the resource availability of the devices involved.

2.2.5.1 T-C: Continuous

In the continuous case, the data is transmitted directly and immediately to the offloading infrastructure as it is collected, which minimises the resources required locally, but may put a considerable load on the network.

2.2.5.2 T-O: On-demand

Offloading can occur on-demand, at the time that a user or application requires it. There is no need for a condition to have been predefined. Here, the timing of offloading cannot be predicted.

2.2.5.3 T-R: Rule-based

This is similar to the on-demand case with the difference that there is a predefined condition, which if met, triggers offloading. A surveillance camera may be able to perform low-complexity computation onboard to determine the presence of a person, and only when this condition is met (that there is a person detected), then transmit the data to a remote, more powerful infrastructure that is able to perform the more complex computation to confirm the presence and additionally identify who the person is. Again, as in the on-demand case, the timing of offloading cannot be predicted.

In (Ahn et al. 2017) and (Shah-Mansouri & Wong 2018) the authors formulate the decision of where to offload as a computation offloading game to model the competition of multiple IoT users to obtain access to the limited resources of close-by fog devices. They assume that the users are selfish and only care about maximising their own quality of experience, which is measured in terms of reducing computation energy and delay. They also assume that each IoT device is equipped with multi-radio access technology and different wireless interfaces allow the IoT devices to connect to different offloading devices (e.g. fog or cloud).

Shah-Mansouri and Wong (Shah-Mansouri & Wong 2018) also propose a computation offloading game to model the competition for fog resources between IoT devices. It aims to minimise energy and delay, which is translated as the maximisation of IoT users' quality of experience.

Tan et al. (2017) proposes an online job dispatching algorithm for offloading jobs to

either an edge node or a cloud and also an online scheduling algorithm that decides the order of execution of the jobs once they have been offloaded either in an edge or a cloud server. They also claim that their dispatching algorithm works when local execution is an option.

2.2.5.4 T-P: Periodic

In the periodic case, offloading occurs with a predefined frequency (Stavrinides & Karatza 2017). For instance, in (Loukas et al. 2017), a battery-powered robotic vehicle sends every 1 second a sensor data to a remote server, which in turn processes the data for the purposes of intrusion detection and sends back to the vehicle the detection decision. Here, the challenge is in choosing an appropriate offloading period. If it is too long, then it may also take too long to detect an ongoing cyber attack, which for the particular application area of the connected vehicle may lead to physical impact. If it is too short, then the overall delay, including the processing time and the network delays involved, may be such that the remote infrastructure will be receiving the next sample of data before the previous one has been processed. Contrary to the on-demand and rule-based cases, here the timing of offloading is predictable.

2.2.5.5 T-DD: Dynamic decision

Since the end-device has limited computational and energy resources, offloading the content to the cloud would help to address the limitation. However, transmission costs to the cloud might be more than computation cost locally. Thus, approximating the cloud to the edge of the network would address the gap. Therefore, context-aware should be considered before deciding to offload. For example, with an application that requires high traffic, offloading to the nearby resource would be more beneficial, while offloading applications with high computation demands and low traffic to the cloud will result in

satisfactory results (Jalali et al. 2017). Machine learning, AI, and context-aware services provide abilities to making a context-aware decision through the runtime to make the right decision ”dynamically” as the decision of offloading is not practical always (Elazhary 2017).

Hassan et al. (2014) stated several requirements for a good offloader in which able to select the most important features influenced the decision with low biased, high variant, indulgent with noise. Precisely, lightweight package to run on limited computation resources; intelligent enough to capture the interchange among key features; self-learning over time; and high accuracy.

Therefore, in this thesis, an intelligent and decentralised decision-maker that eases task offloading implementation in a scalable and reliable way is proposed. Our aim is to both systematise the process of making the decision and make it dynamic by developing a decision support module that decides whether and what part of a task should be processed where according to a heterogeneous set of criteria. To this extent, we plan to produce a servlet for carrying out the offloading processing, to propose mathematical models for determining the optimal processing decision, and crucially a software module for materialising the models developed and carrying out the necessary experiments with real data.

2.3 IoT resource offloading applications

Long et al. (2018) have applied the edge offloading paradigm in the Multimedia Internet-of-Things domain, whereby mobile devices, such as smartphones and tablets, which are typically less resource-constrained than IoT cameras, extract features from videos and only send a few of them to the cloud servers that are responsible for further processing. This primarily aims to avoid bandwidth starvation due to delivering original video chunks directly to the cloud. The authors have proposed a framework for cooperative

processing on mobile devices and corresponding algorithms for the optimal allocation of mobile devices into video processing groups and the optimal allocation of video chunks to groups.

Cao et al. (2015) have presented FAST, which is a real-time and portable fall detection for Stroke Mitigation based on fog computing system. They used pervasive and low-cost end devices for sensing data from patients and employed a middleware platform to analyse the data. An interesting characteristic of their system is that it employs a simpler algorithm for fall detection at the edge and a more complex (and more accurate) one in the cloud. There was no benefit in terms of energy consumption, but using a fog approach as opposed to an IoT-cloud one was shown to reduce the network delays and response times significantly. In contrast, the conventional way of fall detection (Abbate et al. 2011, Bourke et al. 2007), which uses wearable devices and sensors to sense and progress data by employing threshold-based scheme, shows poor accuracy, especially in terms of false positives. To address this, sophisticated pattern-matching algorithms can be used (Castillo et al. 2014, Violeta et al. 2014), but these are not always practical to implement on wearable devices due to lack of storage and computational capacity.

2.4 Industry Perspective

The ETSI-driven standard multi-access mobile edge computing (MEC) is an initiative that facilitates both cloud computing and a network edge computing environment for application developers and content distributors (ETSI et al. 2018). This has been formulated to accommodate the vast IoT landscape's many potential business use-cases by involving industry players as follows.

2.4.1 Cisco

Cisco IOx application environment brings together their networking operating system (IOS) running inside switches and routers, and Linux, the leading open-source software development platform (Roberto De La Mora 2018)). The term “fog computing” was introduced by Cisco in 2014 first which provides options to place computing resources and data at the cloud, edge and/or destination (e.g., Cisco Kinetic (Edge & Fog Processing Module 2018)). The fog nodes, which Cisco use interchangeably as edge computing devices as well, enable both fog-fog (e.g., hierarchical arrangement) and fog-cloud interfaces in order to address load balancing and failure recovery (OpenFog Consortium Architecture Working Group 2017). This can be achieved through network planning, i.e., where and how many fog nodes’ distributed deployment would facilitate multipath from IoT data source towards the destination (e.g., edge or cloud).

Cisco IOx enables different IoT-based applications to operate under the same platform. It claims to integrate non-standard and proprietary protocols run IoT devices with a common IP architecture via device abstractions. Furthermore, their data can be offloaded at the edge for real-time IoT driven analytics pioneered by ParStream DB (Bienert 2016) which is characterised by high performance compressed indexing, parallel processing and a small footprint. Depending on the application need, the developers can also push rule-based data filtering, partitioning, and management capabilities at the edge through RESTful API of fog nodes. The placement or distribution of fog nodes throughout the network not only can offload the communication efficiently to the local fog node for faster reaction time, but also lessens the hacker threat where data summaries may only leave the local network towards the cloud periodically for historical and big data analysis.

In short, Cisco strives to be one of the major players in the IoT-driven application scenarios, and their roadmap towards achieving it (e.g., six pillars(Cisco 2019)) has

seen significant success already.

2.4.2 Intel

The Intel IoT system architecture specification envisions seamless realisation of cyber-physical systems using both smart objects, which are IPSO-alliance.org compatible, and legacy devices that were not originally conceptualised with intelligence or Internet connectivity in mind (The Intel IoT Platform 2015). Arguably 85% of all devices fall under this legacy category according to an IDC (International Data Corporation) study (The Intel White Paper 2015). The component "Intel IoT Gateway" of the architecture is Intel's own term for edge devices. i.e., the primary on-premises devices. The purpose of such devices is two-fold, i.e., connecting both legacy and smart devices by acting as intermediaries and enabling secure data flow between these devices and the cloud-powered by McAfee embedded control security technologies. In other words, these devices gather data from the endpoint sensors/actuators and provide provisions for filtering and aggregation before forwarding them to the cloud. Depending on the application need, it can also facilitate intelligent computational offloads, and local data analytics, and subsequent real-time control decisions at the edge devices.

Several such case studies listed on their website ranging from smart home, retail, building to smart city and transportation (Intel 2018) typically use Intel products, e.g., Intel IoT Gateway together with Arduino board, and third-party cloud infrastructure (e.g., Microsoft Azure). In terms of software, the gateways, i.e., edge devices, support Wind River, Microsoft and Linux Operating Systems, enabling APIs (e.g., REST) to facilitate both edge-IoT and edge-cloud communication protocol realisation for developers. MongoDB and SQLite have been the choice for storage at the edge to enable local data analytics.

2.4.3 Huawei

Huawei's most popular adopted choice of technology NB-IoT (Narrow-band IoT) was conceptualised to leverage low-power and wide-area coverage communications that can be deployed over existing mobile networks operating in licensed spectrum backed by telecom industries (Huawei Technologies Co., Ltd. 2015).

Huawei's EC-IoT platform consists of an edge computing gateway with various IoT-access interfaces, and even supports typical RF communication mode (e.g., ZigBee and Bluetooth) (Huawei Technologies Co., Ltd. 2017). The northbound RESTful interface (towards the cloud) of this gateway enables seamless integration of management, big data analytics, and third-party applications. Huawei claims the incorporation of local intelligence, analysis and near real-time response at the edge through several case studies in the field of smart elevator, city and lighting IoT, smart energy and manufacturing, etc. (Huawei Technologies Co., Ltd. 2018).

The adoption of network virtualisation and SDN open architecture facilitates scalability, efficient placement and management of multiple gateways within the same application. Huawei's cloud platform, CloudEPC, is an open platform based on Cloud Native architecture that is argued to provide a hybrid cloud deployment option in terms of Huawei public cloud together with local data centres (Huawei Technologies Co., Ltd. 2018). Huawei is also striving for seamless integration of various smart objects to improve interoperability through the incorporation of an IoT-oriented software platform, LiteOS inside them (Huawei Technologies Co., Ltd. 2018).

2.4.4 Hewlett-Packard

Hewlett-Packard Enterprise (HPE) partnered with Saguna (Saguna 2018) and Amazon Web Services (AWS) (Amazon 2018) to enable IoT applications platform at the network edge. In (HPE, Saguna and AWS 2018), three different application scenarios, namely,

"Smart City Surveillance", Augmented Reality (AR)/Virtual Reality (VR), Connected Vehicle (V2X), were explained to rationalise such design choice. The applications have a few things in common: i) they generate massive dataset, ii) require near real-time response to certain scenarios, and iii) involve computationally intensive operations that are part of Artificial Intelligence (AI) algorithms. These operations are generally delegated to the cloud, whereas the data analytics required for near real-time inference is pushed to the edge. For example, these AI applications' training phases are completed at the cloud after which the inference model is communicated with the edge device that will then be expected to provide faster response to the on-field IoT sensors' readings. HPE-manufactured multi-access edge device (MEC) is equipped with Intel™core processor and also hosts Saguna's network virtualisation solution. MEC talks with eNodeB of mobile telecom networks, which gives freedom in terms of its placement, e.g., distributed over the network. Saguna's virtualisation solution is argued to extend its functionality to include other type of networks as well (e.g., Wi-Fi). Part of AWS (AWS Greengrass) is situated inside the MEC to enable local cache and cloud functionalities at the edge, whereas the typical AWS cloud hosts policies, management and historical data analytics.

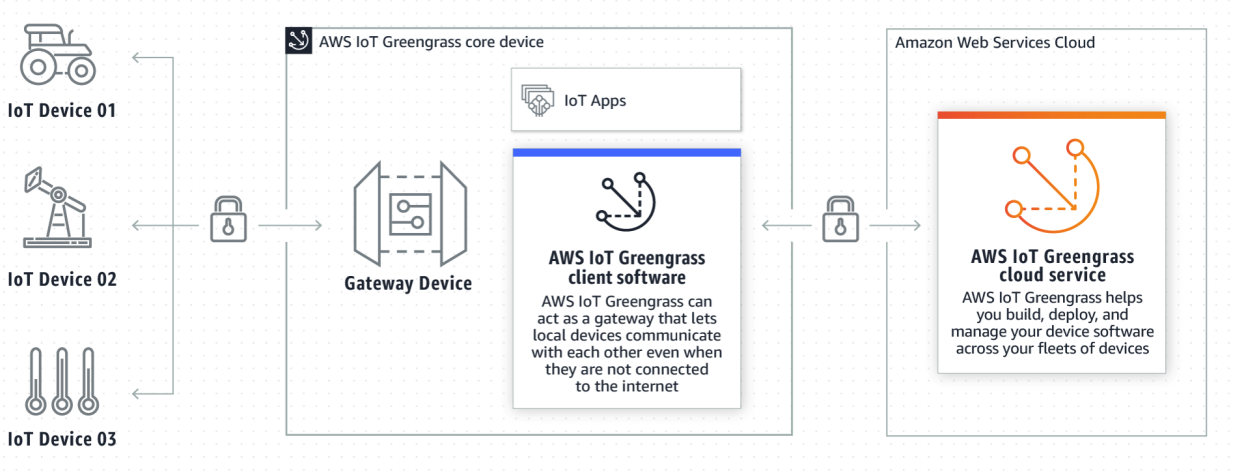
2.4.5 Amazon Greengrass

AWS Greengrass (GG) is an edge software that approximates the cloud capabilities to nearby devices at the edge of the network between IoT devices and cloud, see Figure 2.2, (AWS 2019). AWS Greengrass brings cloud programming and functionality to the edge of the IoT network empowering them to communicate and react when cloud connection is not possible through a set of pre-defined functions called Lambda functions (LF). LF is used to create serverless applications for the IoT device at the edge. The devices, including IoT, edge, and cloud, are called Greengrass groups.

Greengrass group is always defined and configured from the cloud. The first step

in creating a new group is to establish a Greengrass core in the cloud definition. Every group needs a Greengrass core to function. This core software securely connects the devices to AWS. Also, to preserve inbound and outbound messages to the cloud, a local sub message manager is provided that makes GG able to buffer messages intelligently in case of lost connectivity.

Figure 2.2: AWS IoT Greengrass interactions



We have provided only a brief overview of the offered edge computing solutions and relevant products of a few important industry players above. Others are also trying to be involved as well. For example, Google Google Cloud (2018) announced its own edge computing gateway product “Edge TPU (Tensor Processing Unit)” in order to push their TensorFlow Lite ML inference models at the edge. IBM’s Watson IoT Platform Edge feature (IBM Cloud 2018), and Microsoft’s Azure IoT Edge service (Microsoft Azure 2018) strive to achieve the same purpose. SAP’s IoT Gateway product is built with third-party hardware and acts as an intermediary between SAP cloud and sensors (SAP Interface Integration Certification 2017). All the major telecom industries like Ericsson, Nokia, Vodafone, etc., are largely dedicated towards enabling IoT applications over mobile networks on top of narrow-band communication though.

To sum up, it can be seen that based on a particular industry's already available products and market strength, their IoT-driven future prospects also differ. For example, the telecom industries are solely trying to push narrow-band technologies and the applications built on top of it. This may not be suitable in a wide range of IoT-driven application landscape as can be seen from some of Huawei's case studies itself (Huawei Technologies Co., Ltd. 2015).

All industry players acknowledge the need for near real-time response, maintaining privacy and security, and reducing costs as factors promoting edge computing. However, they are only providing platforms to accommodate as many diverse applications as possible, leaving the computational and data offloading decisions largely to the application developer based on their unique needs.

2.5 Summary

In this chapter, we have discussed the resource restrictions that IoT devices and environments are facing. By offloading those resources to other, more powerful devices, such as edge or cloud servers, the IoT infrastructures of the future could perform better and become more energy efficient. We also proposed a comprehensive taxonomy that allows us to better understand the different types of resources that can be offloaded, the different techniques and decision mechanisms, as well as the different device choices that participate in those decisions.

IoT offloading is generally perceived as the necessary way to enable IoT-driven applications to reduce response times and improve energy efficiency. It could also lead to reducing the use of cloud infrastructures to the minimum, thus reducing cloud usage costs and the attack surface for such environments. However, providing offloading decisions in real-time (or close to real-time) is proving to be challenging, especially since network conditions are constantly changing. Thus, predicting them can be challenging.

Chapter 3

Dynamic decision support for computation offloading in heterogeneous Internet of Things environments (MEDICI)

3.1 Introduction

In this chapter, we first review the related work on decision making in IoT computation offloading, and then, we describe the mathematical model of our initial offloading mechanism. Then, we survey a few existing simulation toolkits in the edge-cloud computing context before presenting our simulation environment by describing how we extended the EdgeCloudSim simulator. Finally, we present our simulation results, which compare our MEDICI algorithm to five offloading strategies used in the literature and for four different applications (Jaddoa et al. 2020).

3.2 Related work

The majority of edge offloading decision making mechanisms proposed in the literature refer to mobile edge computing (Mach & Becvar 2017). For example, Meurisch et al. (2017) 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 (Gelenbe et al. 2008) 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 (Ahn et al. 2017), 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 (Shah-Mansouri & Wong 2018), the authors 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. (2018) also proposes 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 (Vakilinia et al. 2017) 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. (2015) proposed 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.

The authors in (Samie et al. 2016) 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 (Lyu et al. 2018), 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 for optimising both time energy. For example, Du et al. (2018) formulate an optimisation problem to minimise the energy consumption or latency when offloading to fog or cloud nodes. Similarly, in (Liu et al. 2017), 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 (Chen et al. 2019).

Benedetto et al. (2019*b*) 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. (2019*c*) 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. (2019) 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 (Alelaiwi 2019), Alelaiwi proposes to use deep-learning in order to predict the response times of a task based on historical observations or predefined parameters such 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 Delay. 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 (in-

cluding 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 in this chapter, 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.

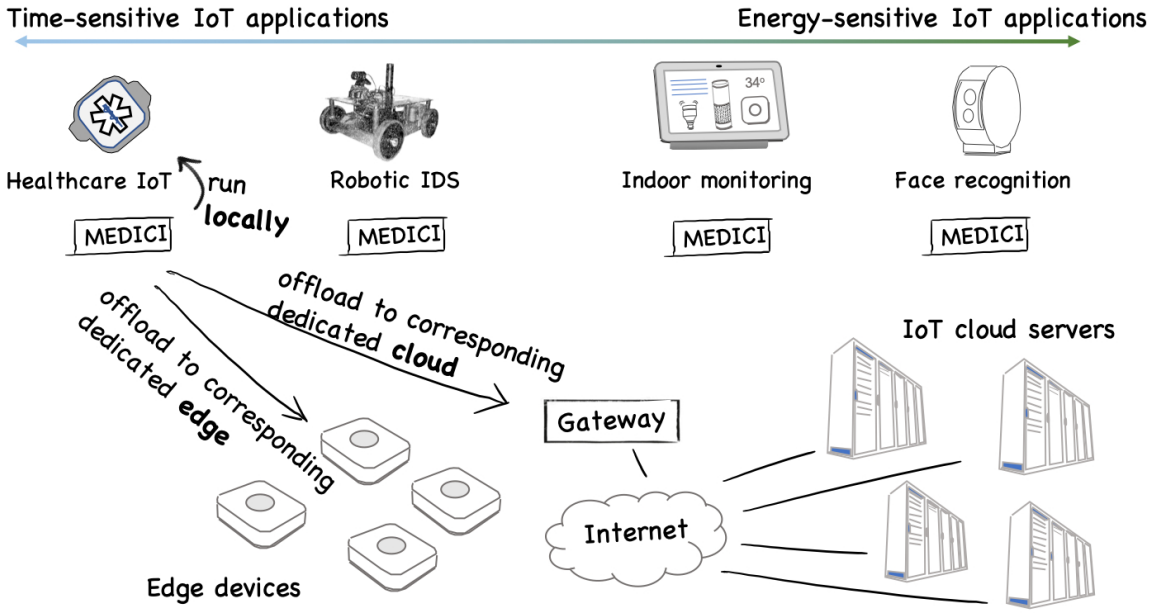
Without loss of generality, we assume in this chapter an IoT architecture where a variety of heterogeneous IoT devices and their corresponding applications are connected to the Internet (and the cloud) through a gateway (Fig. 3.1) and are directly connected to the edge devices in proximity. Note that 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.

The decision for where to offload is taken based on the model described in the next section.

3.3 MEDICI offloading model

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

Figure 3.1: Conceptual IoT offloading decision making architecture



on the IoT device itself. Table 3.1 summarises the notations used in this chapter.

Table 3.1: 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

Continued on next page..

Table 3.1 Variables and Notations – continued from previous page

Variable	Description/Definition
$W_{n,D}^{procQueue}$	The time a task has to wait at the queue of device D before it is processed
RTT_D^x	Round trip time(RTT) when device D transmits data to another device
RTT_D^r	Round trip time(RTT) when device D receives data from another device
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

Continued on next page..

Table 3.1 Variables and Notations – continued from previous page

Variable	Description/Definition
$E_{n,D}^{selfish}$	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

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.

3.3.1 Processing times model

In accordance to the usual representation of processing time in task allocation problems (Kumar & Lu 2010), the time it takes for a task n with size z_n to be processed at a device

D is:

$$T_{n,D}^{proc} = \frac{z_n}{S_D} \tag{3.1}$$

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:

$$W_{n,D}^{procQueue} = \sum_{i=1}^{k} T_{i,D}^{proc} = \sum_{i=1}^{k} \frac{z_i}{S_D} \tag{3.2}$$

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.

3.3.2 Network Delay model

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 (Kumar & Lu 2010),(Loukas et al. 2017), (Niu et al. 2014), (Ma et al. 2018), 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:

$$T_{n,D}^x = l\,\frac{d_n^x}{R_D^x} \tag{3.3}$$

$$T_{n,D}^r = l\,\frac{d_n^r}{R_D^r} \tag{3.4}$$

We also model imperfect network conditions in the form of congestion, as expressed by network queuing Delay. Similarly to (Mehmeti & Spyropoulos 2014), and (Wu et al. 2015), 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:

$$W_{n,D}^{x,netQueue} = \frac{L_D^x}{\lambda_D^x} = \frac{1}{\mu_D^x - \lambda_D^x} - \frac{1}{\mu_D^x} \tag{3.5}$$

and for receiving the result back:

$$W_{n,D}^{r,netQueue} = \frac{L_D^r}{\lambda_D^r} = \frac{1}{\mu_D^r - \lambda_D^r} - \frac{1}{\mu_D^r} \tag{3.6}$$

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.

3.3.3 Response time model

We refer to response time as the total time it takes for a task n to be transmitted (first two terms of equation 3.7) and processed (next two terms of equation 3.7) and the result to

be returned back to the IoT application (last two terms of equation 3.7):

$$T_{n,D} = T^{x}_{n,D} + W^{x,netQueue}_{n,D} + T^{proc}_{n,D} + W^{procQueue}_{n,D} + T^{r}_{n,D} + W^{r,netQueue}_{n,D} \tag{3.7}$$

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 $T^{x}_{n,IoT} = W^{x,netQueue}_{n,IoT} = T^{r}_{n,IoT} = W^{r,netQueue}_{n,IoT} = 0$ and $T_{n,IoT} = T^{proc}_{n,IoT} + W^{procQueue}_{n,IoT}$.

3.3.4 Energy consumption model

Let P^{proc}_{IoT} be the average power consumed when the processor of the IoT device is busy, and P^{idle}_{IoT} be the power consumed when idle. Also, P^{x}_{IoT} and P^{r}_{IoT} 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:

$$E^{local}_{n,IoT} = P^{proc}_{IoT} T^{proc}_{n,IoT} \tag{3.8}$$

If task n is offloaded to a target device D other than the IoT device, then the energy consumed by the target device $E^{off}_{n,D}$ 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 $E^{off}_{n,IoT}$ 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.

$$\begin{aligned} E_{n,IoT}^{off} = & P_{IoT}^{x} T_{n,D}^{x} \\ & + P_{IoT}^{idle}(W_{n,D}^{x,netQueue} + W_{n,D}^{procQueue} \\ & + T_{n,D}^{proc} + W_{n,D}^{r,netQueue}) \\ & + P_{IoT}^{r} T_{n,D}^{r} \end{aligned} \tag{3.9}$$

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

$$E_{n,D}^{off} = P_{D}^{r} T_{n,D}^{r} + P_{D}^{proc} T_{n,D}^{proc} + P_{D}^{x} T_{n,D}^{x} \tag{3.10}$$

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:

$$E_{n}^{off} = E_{n,IoT}^{off} + E_{n,D}^{off} \tag{3.11}$$

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

$$E_{n,D}^{selfish} = \mathbb{1}[D = IoT] E_{n,IoT}^{local} + \mathbb{1}[D \neq IoT] E_{n,IoT}^{off} \tag{3.12}$$

Similarly, for altruistic IoT devices, it is:

$$E_{n,D}^{altruistic} = \mathbb{1}[D = IoT] E_{n,IoT}^{local} + \mathbb{1}[D \neq IoT] E_{n}^{off} \tag{3.13}$$

3.3.5 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:

$$G_{n,D}^{selfish} = \alpha T_{n,D} + (1-\alpha)\gamma E_{n,D}^{selfish} \tag{3.14}$$

Similarly, for altruistic IoT devices:

$$G_{n,D}^{altruistic} = \alpha T_{n,D} + (1-\alpha)\gamma E_{n,D}^{altruistic} \tag{3.15}$$

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:

$$C_n^{selfish} = \underset{D \in \{IoT, edge, cloud\}}{\operatorname{argmin}} G_{n,D}^{selfish} \tag{3.16}$$

$$C_n^{altruistic} = \underset{D \in \{IoT, edge, cloud\}}{\operatorname{argmin}} G_{n,D}^{altruistic} \tag{3.17}$$

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 (Tan et al. 2017).

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.

3.4 Experimental setup

3.4.1 Simulation environment

In general, analysing empirical data on edge/cloud computing environment would help to represent the behaviour of a proposed approach by studying the interactions between the components of that system, and network traffic and execution environments, etc.

However, since edge computing integrates with IoT devices, cloud datacentre, and edge nodes, working with real-world testbed would be costly (Mahmud & Buyya 2019). Thus, simulation is the proper method that can be used in order to explore, develop and evaluate resource management strategies of our proposed approach. Furthermore, simulation provides frameworks of desired experiments, as well as repeatable evaluation.

Generally, there are a few available simulation toolkits that can be used to simulate the fog/edge computing environment such as CloudSim (Goyal et al. 2012), iFogSim (Gupta et al. 2017), and EdgecloudSim(Sonmez et al. 2017). However, obtaining one of them requires significant programming effort to meet the actual needs. Baktir et al. (2018) point out five challenges that are needed to be addressed for an ideal edge/fog-cloud computing simulation environment, which are computational resource, mobility, network and efficiency modelling.

CloudSim (Goyal et al. 2012) is one of the most popular simulators in the cloud computing sector, and it is widely adopted by a large number of research to simulate cloud environment such as (Beloglazov et al. 2011), and (Malawski et al. 2015). CloudSim tool supports both behaviour modelling of the cloud system components such as data centres, virtual machines (VM's) and resource provisioning policies and application services. However, Wireless Area Network (WAN)and Wireless Local Area Network (WLAN) communication model is not implemented, and it is designed for pure cloud computing evaluation (i.e. principally for evaluating datacentre performance).

Another simulator is iFogSim (Gupta et al. 2017). It is a toolkit that is run on top of the fundamental framework of CloudSim (Goyal et al. 2012); it is designed for modelling an IoT-fog computing environment including sensors, actuators, fog devices and cloud datacentres. It gives users the ability to define their own topology and application model. Moreover, since it built on top of CloudSim, where communication is performed by passing messages or sending events, so no network traffic is simulated, therefore, fine-grained network details, communication results and latencies cannot be achieved.

Worthily mentioned, having no prior knowledge about CloudSim means setup the desired configuration becomes more challenging as it is required to go through many modules of CloudSim in order to implement the testbed. However, iFogSim has been widely used in the literature for example (Bittencourt et al. 2017), (Taneja & Davy 2017) and (Mahmud, Ramamohanarao & Buyya 2018)

EdgecloudSim (Sonmez et al. 2017) is another simulator that also built on top of CloudSim simulator Goyal et al. (2012). It is introduced to cover the missing features in the mentioned simulators (iFogSim and CloudSim) to suit the more advanced scenarios by adding mobility aspect, network modules (e.g. WAN, and WLAN) and edge Orchestrating (i.e. balancing between edge nodes). Therefore, we performed our simulations by extending the discrete event simulator EdgeCloudSim.

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, for instance (Zhang et al. 2018), (Lee & Lee 2018), and (Scoca et al. 2018). 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, see Figure 3.2. 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).

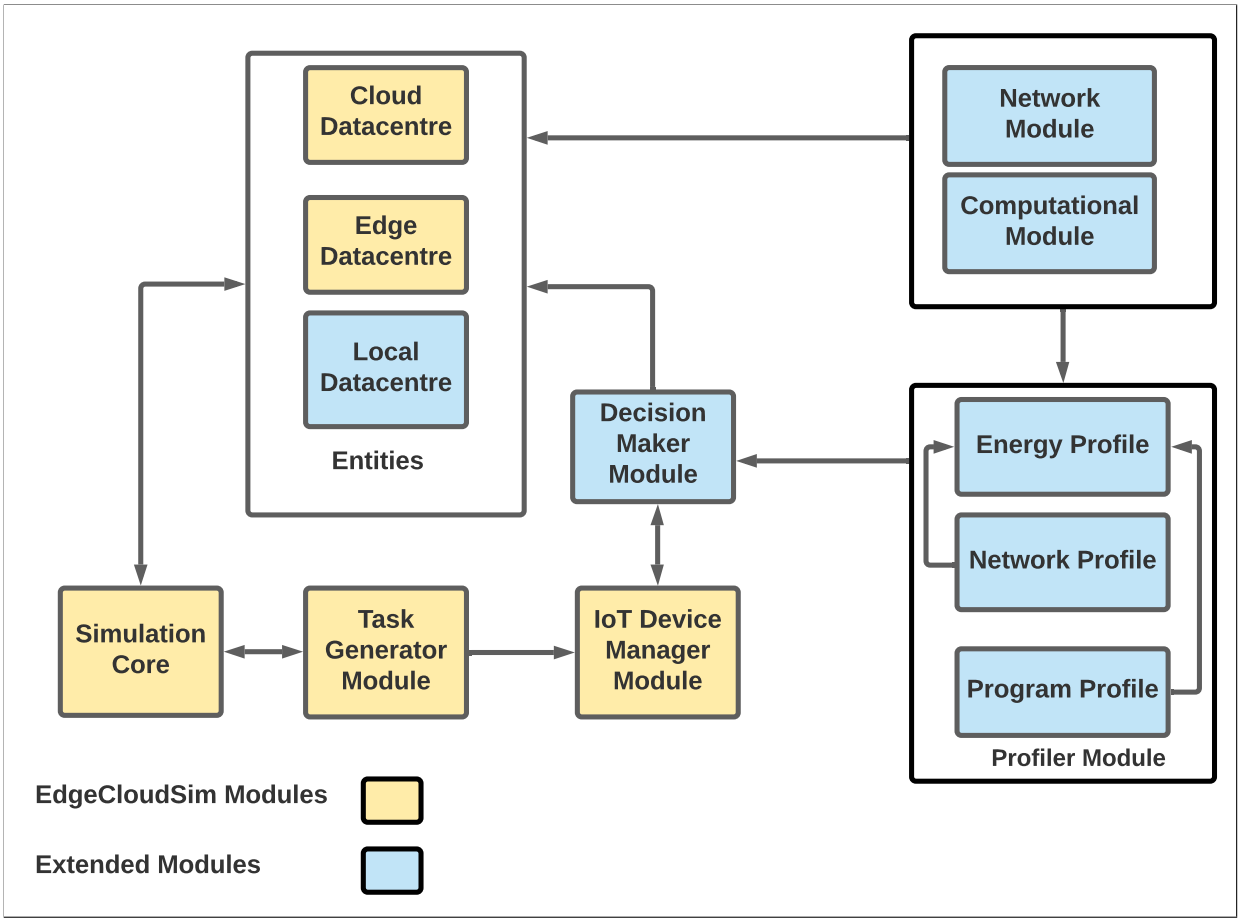
Each simulation starts with an initialisation phase, where a load generator creates a set of tasks based on a Poisson distribution for each application with parameters such as application type, start time, size, input and output data size. These parameters are

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.

3.4.2 Simulation setup

As mentioned in the previous section, 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., (Kapsalis et al. 2017)). 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 $5Mbps$ 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 net-

Figure 3.2: Diagram of the our extended simulation environment modules

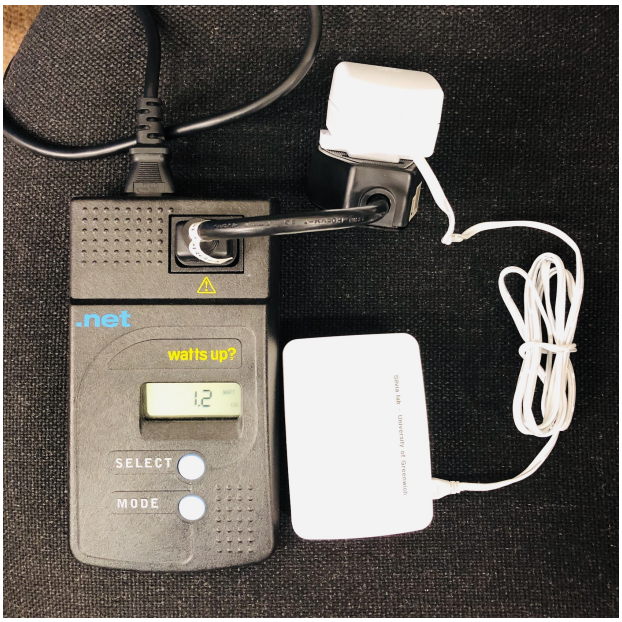


work between between IoT devices and cloud is considered a WAN with $2Mbps$ effective bandwidth. Finally, we have set the packet loss probability $p = 0.11$ and thus the delay factor due to loss $l = \frac{1}{1-0.11} = 1.12$ which is similar to (Loukas et al. 2017).

In accordance to our energy model (Section 3.3.4), 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 (Gupta et al. 2017), where for edge 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 (Devices 2009) measuring the actual power consumption when idle, transmitting, receiving and processing a computationally

intensive application, see Figure 3.3. Table 3.2 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.

Figure 3.3: Testbed of measuring IoT device (Raspberry Pi3) energy consumption modes



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 (Pyzyk 2018)), one for healthcare IoT (as one of the in-home therapeutic IoT applications described in (Rahman et al. 2018)), one for IoT intrusion detection (such as the robotic vehicle intrusion detection implementation in (Loukas et al. 2018)), 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

Table 3.2: Device specifications for MEDICI setup

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	10000	2	103	103	-

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 3.3. The mean task size values were chosen based on the default settings of the EdgeCloudSim simulator, denoting the complexity of the task, similar to Sonmez et al. (2017). The values for the input and output data sizes were chosen empirically for the specific types of the applications. For instance, face recognition applications usually have average sized input images, and the outcome is a smaller description of a much smaller size. Healthcare applications could have video inputs which are relatively big, however their output is small, e.g., a small image or report. The intrusion detection application usually has a small sized input file with e.g. network traffic measurements or sensor readings, and the result is very small, since it might consist of a simple yes or no answer, or a report on whether an attack was detected or not. Finally, for the indoor monitoring, the input could be a small file of sensor data while the output could be a report of e.g., recommendations for turning devices on/off, failure reports, or energy consumption/cost predictions, etc.

Table 3.3: Application Specifications and Requirements for MEDICI setup

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

The weight α is also chosen according to the type of applications, (as previously explained in section 3.3.5). 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 (equation 3.14). However, for altruistic-MEDICI (equation 3.15), 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.

3.5 Experimental Results

Here we evaluate our proposed MEDICI mechanism. 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 3.3) 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 3.3), 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 (Samie et al. 2016). 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 ($500MI$) to avoid a situation where, for applications with computation 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 (Ma et al. 2018).

- Random Selection (***Random***): This is a special case of the probabilistic approach, where the probabilities between the choices are equal as in (Ma et al. 2018).

Each experiment was run for 1 hour of simulated time. The results presented here are the averages of 10 runs for each configuration. Since the results of some strategies are close to each other, we decided to show our results in figures and tables for the purpose of ease of tracking. Table 3.4 and Figure 3.4 show 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 Delay, represented in blue) and downloading delay (transmission of the result (output) data of the offloaded task plus the network queuing delay, represented in orange). Table 3.5 and Figures 3.5 show the energy consumption at only the IoT device while Table 3.6 and Figure 3.6 show and the total energy consumption at both IoT device and offloading target.

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

Figure 3.4: Average total response time per application

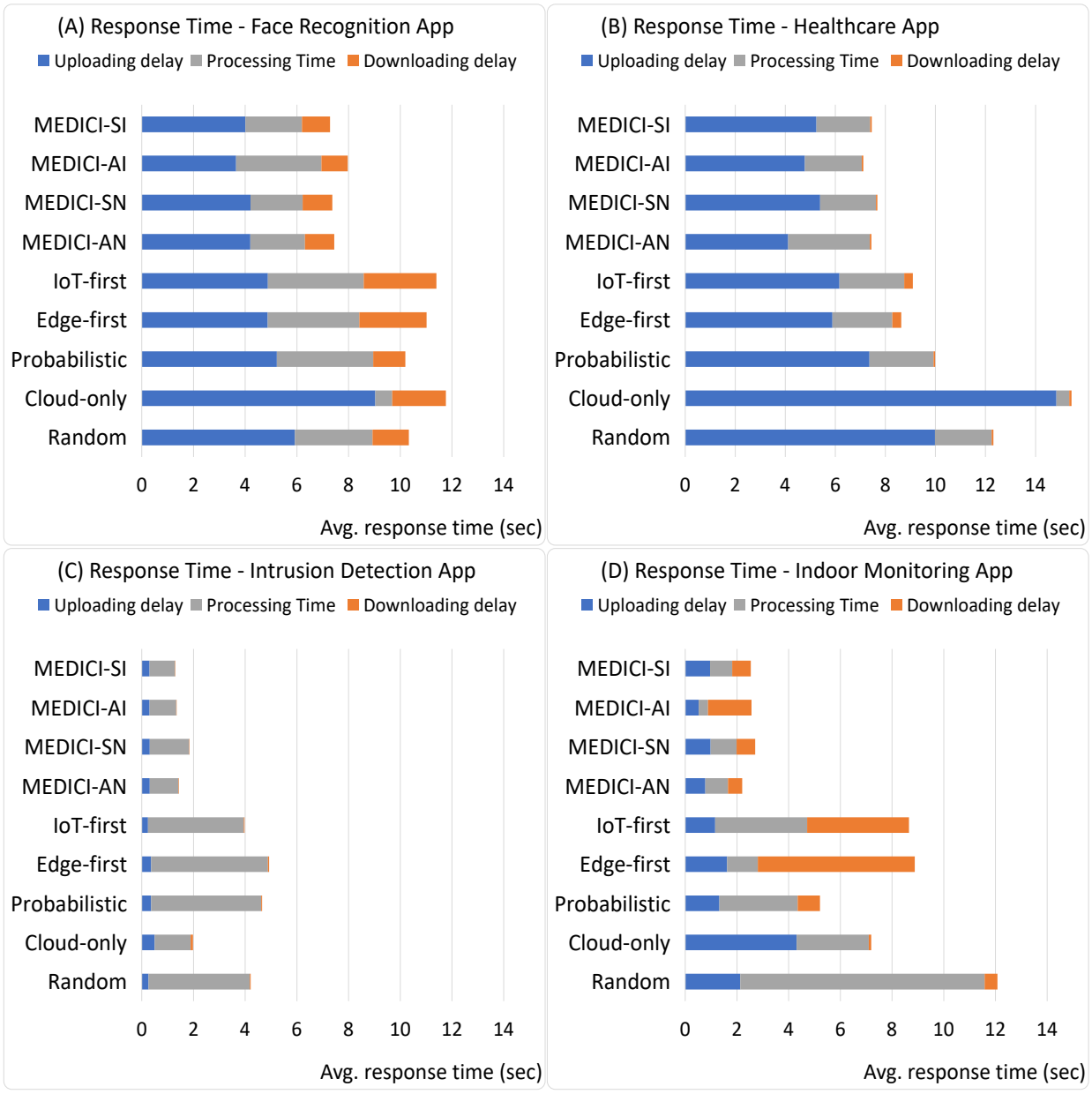


Table 3.4: Average total response time per application (sec)

Face recognition App									
Delay	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Uploading delay	5.92	9.04	5.23	4.87	4.88	4.20	4.21	3.65	4.01
Downloading delay	1.40	2.08	1.24	2.60	2.81	1.14	1.15	1.02	1.09
Processing time	3.01	0.65	3.73	3.54	3.71	2.12	2.01	3.30	2.19
Total response time	10.33	11.76	10.19	11.01	11.39	7.45	7.37	7.97	7.28

Healthcare App									
Delay	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Uploading delay	9.98	14.84	7.36	5.88	6.16	4.11	5.39	4.79	5.24
Downloading delay	0.07	0.09	0.07	0.36	0.34	0.06	0.06	0.06	0.06
Processing time	2.27	0.52	2.56	2.39	2.59	3.27	2.23	2.27	2.15
Total response time	12.32	15.44	9.99	8.63	9.09	7.44	7.68	7.12	7.45

Intrusion detection App									
Delay	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Uploading delay	0.26	0.50	0.36	0.36	0.24	0.31	0.31	0.30	0.30
Downloading delay	0.03	0.09	0.03	0.05	0.03	0.02	0.02	0.02	0.02
Processing time	3.92	1.39	4.26	4.51	3.71	1.11	1.52	1.03	0.99
Total response time	4.21	1.98	4.65	4.93	3.97	1.44	1.85	1.35	1.30

Indoor monitoring App									
Delay	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Uploading delay	2.13	4.31	1.32	1.62	1.16	0.77	0.98	0.52	0.96
Downloading delay	0.49	0.09	0.86	6.07	3.94	0.55	0.72	1.69	0.71
Processing time	9.45	2.79	3.03	1.19	3.56	0.88	1.00	0.36	0.86
Total response time	12.07	7.20	5.21	8.88	8.65	2.20	2.70	2.57	2.53

Figure 3.5: Average energy consumption of each IoT device per app

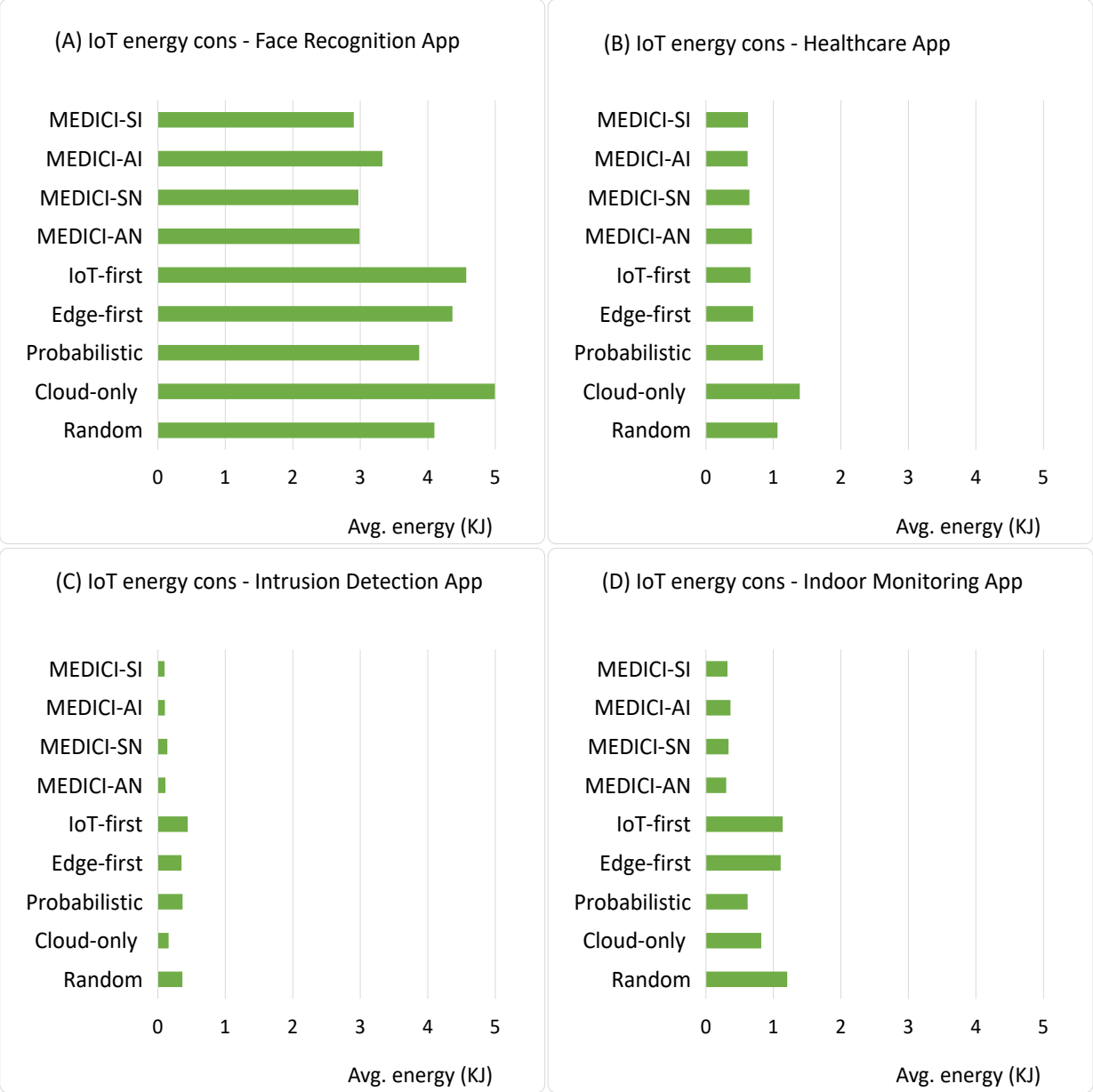


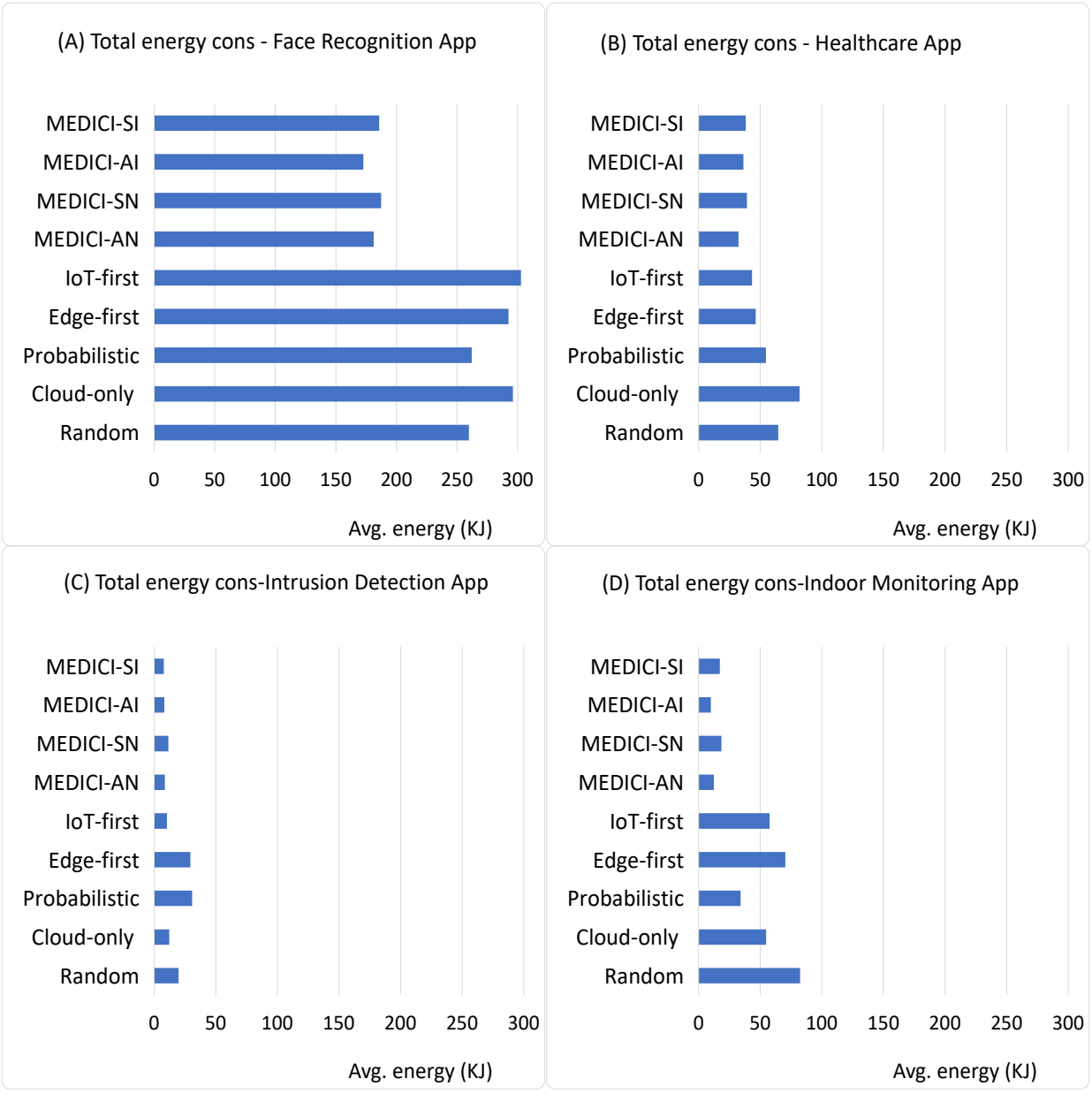
Table 3.5: Average energy consumption of each IoT device per app (KJ)

Strategy	Face Recognition	HealthCare	Intrusion Detection	Indoor Monitoring
Random	4.10	1.06	0.36	1.20
Cloud-only	5.00	1.39	0.16	0.82
Probabilistic	3.87	0.84	0.37	0.62
Edge-first	4.36	0.70	0.35	1.11
IoT-first	4.57	0.66	0.44	1.14
MEDICI-AN	2.99	0.68	0.11	0.30
MEDICI-SN	2.97	0.64	0.14	0.34
MEDICI-AI	3.33	0.62	0.11	0.36
MEDICI-SI	2.90	0.62	0.10	0.32

Table 3.6: Average total energy consumption per application (KJ)

Strategy	Face Recognition	HealthCare	Intrusion Detection	Indoor Monitoring
Random	259.72	64.53	19.91	82.26
Cloud-only	296.15	81.92	12.22	54.81
Probabilistic	262.17	54.54	30.88	34.07
Edge-first	292.51	46.29	29.44	70.34
IoT-first	302.81	43.25	10.36	57.62
MEDICI-AN	181.24	32.34	8.73	12.30
MEDICI-SN	187.28	39.10	11.61	18.49
MEDICI-AI	172.69	36.27	8.28	9.88
MEDICI-SI	185.72	38.11	7.81	17.14

Figure 3.6: Average total energy consumption per application



the selfish MEDICI-SI and -SN. Comparing to the best-performing non-MEDICI strategy (probabilistic), the energy reduction achieved at 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.

Table 3.7: Percentage(%) of tasks run at each device per application

Face Recognition App									
Device	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Local	4%	0%	0%	0%	1%	7%	4%	12%	4%
Edge	48%	0%	64%	68%	65%	51%	56%	50%	60%
Cloud	48%	100%	36%	32%	33%	42%	39%	38%	36%

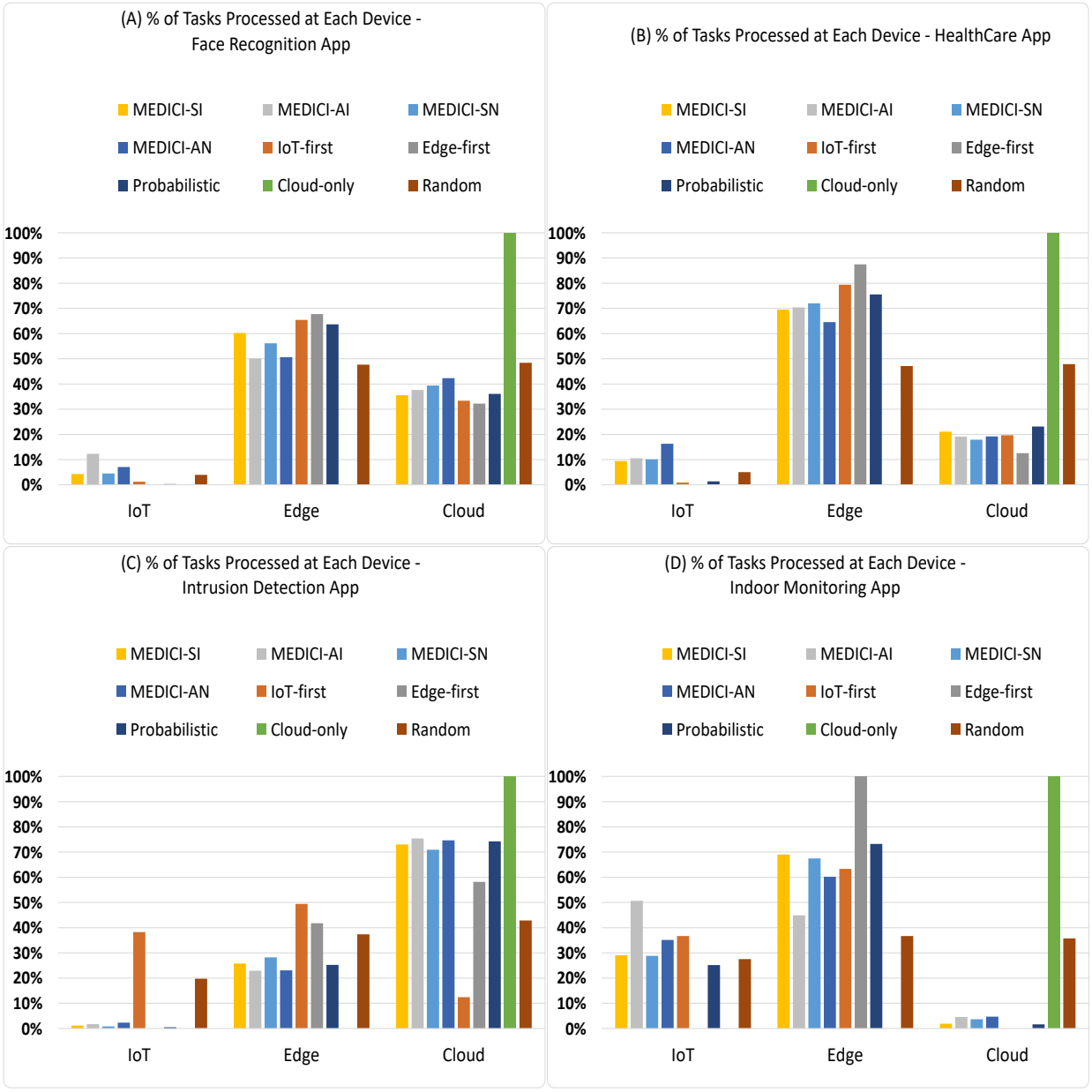
Healthcare App									
Device	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Local	5%	0%	1%	0%	1%	16%	10%	11%	9%
Edge	47%	0%	76%	87%	79%	65%	72%	70%	69%
Cloud	48%	100%	23%	13%	20%	19%	18%	19%	21%

Intrusion Detection App									
Device	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Local	20%	0%	1%	0%	38%	2%	1%	2%	1%
Edge	37%	0%	25%	42%	49%	23%	28%	23%	26%
Cloud	43%	100%	74%	58%	12%	75%	71%	75%	73%

Indoor Monitoring App									
Device	**Random**	**Cloud-only**	**Probabilistic**	**Edge-first**	**IoT-first**	**MEDICI-AN**	**MEDICI-SN**	**MEDICI-AI**	**MEDICI-SI**
Local	28%	0%	25%	0%	37%	35%	29%	51%	29%
Edge	37%	0%	73%	100%	63%	60%	67%	45%	69%
Cloud	36%	100%	2%	0%	0%	5%	4%	5%	2%

Table 3.7 and Figure 3.7 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 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.

Figure 3.7: Percentage(%) of tasks run at each device per application



3.6 Summary

In this chapter, we have proposed a multi-criteria offloading decision mechanism (MEDICI) 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 Delay 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.

Also, we have presented simulation results, using our extension of EdgeCloudSim, and 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. However, it is evident that even when the IoT devices act selfishly, the overall benefits of the system are very close to the benefits of the altruistic mode.

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 chapter 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. In the next chapter, we evaluate these overheads and challenges in real-world implementations.

Chapter 4

Probeless Multi-Criteria Decision Support for IoT Computation Offloading (PL-MEDICI)

4.1 Introduction

In the previous chapter, we introduced a multi-criteria offloading decision mechanism (MEDICI) that takes into account the time it takes to execute a task and to offload its data and the delays incurred by the network, as well as the energy consumption for each of these aspects but with a few assumptions.

Firstly, in section (3.3.1), in order to estimate the processing times of a task in a device, we used equation (3.1) where the task size was given in Million Instructions (MI) and the processing speed in Million Instructions per Second (MIPS) and equation (3.2), where we assumed that we knew how many tasks were waiting to be executed. Although MIPS is an acceptable performance indicator for older or low-end processors and is extensively used to measure task execution time in EdgeCloudSim and similar simulators, it is not the best way to measure CPU performance in real-life situations

and for more recent processors (Kapsalis et al. 2017). Also, as mentioned previously, uncertainties due to sharing of CPU and memory resources could affect the processing times of a task waiting to be executed.

Secondly, in order to estimate the network delays in section (3.3.2) we assumed that we have knowledge of the average end-to-end throughput between the devices for both transmitting and receiving data (equations (3.3) and (3.4). In our simulation environment, both these values were constant. Additionally, to estimate network queuing delays, we assumed the network between an IoT device and the offloading device to be an M/M/1 queue and calculated the network queuing delays for transmitting and receiving according to equations (3.5) and (3.6).

In a real-life situation, where the network and the execution environments constantly change, these assumptions may lead to inaccurate network delays. And such values would need to be obtained by probing each device before each offloading decision. Offloading decision mechanisms in the literature are able to achieve considerable accuracy at predicting the energy and computation costs of an offloadable task, but are typically reliant on a probing phase, which introduces delays, and most of them have low offloading efficiency and cannot be applied in different scenarios.

Here in this chapter, we enhance our proposed model (MEDICI) in Chapter 3 by introducing ***P***robe***L***ess **M**ulti-crit**E**ria **D**ec**I**sion support me**C**hanism for **I**oT offloading (*PL-MEDICI*) that also uses multiple criteria such as processing times, network delays and energy consumption to decide whether a task should be processed locally, at the edge of the network or on remote cloud devices. Improving on the state of the art, it is able to achieve this without relying on probing, by taking into consideration the freshness of the historical data available for each device and task and utilising lightweight statistical techniques.

In this chapter, we first present the recent literature on the dynamic decision support for IoT offloading. Then we introduce PL-MEDICI, the first dynamic and decentralised

offloading decision support mechanism for IoT environments that is ***probeless***, as well as addressing some challenges of applying our offloading mechanism in a real-world implementation. Finally, we present the implementation and configuration of our real testbed, which is consisting of Raspberry Pi's that act as IoT devices and servers that act as edge and cloud devices, in addition to the evaluation of the proposed probeless offloading mechanism, comparing it to three decentralised offloading strategies proposed in the literature, and for three different applications.

4.2 Related work in dynamic decision support for IoT offloading

While in commercial space, edge offloading is based on static policies determined at design level, researchers have proposed dynamic approaches that take into account various aspects of the performance of the devices or the network in between (Chen, Zhang, Zhang & Chen 2018). Some of the early mechanisms for offloading were externally processed, typically in a centralised fashion, rather than autonomously on the IoT device. Data would be collected from the network usually at a specific edge device to process it externally and make the decision, usually solving an optimisation problem based on all the data collected centrally, or applying deep learning. For example, in (Liu et al. 2017), a multi-objective joint optimisation problem was formulated for finding the optimal transmission power of the devices and the probability of offloading that minimises energy, response time and payment cost. The problem was converted into a single objective optimisation problem by using a scalarisation method and assumptions on the expected maximum values for the three criteria, while an Interior Point Method was applied to improve the calculation accuracy. An alternative approach presented in (Park et al. 2020) involved the application of deep reinforcement learning, with a centralised

algorithm for the joint optimisation of delay and energy-efficient communications.

Chen, Zhang, Zhang & Chen (2018) have formulated an optimisation problem to minimise the long-term average offloading cost while providing performance guarantees. It takes into consideration a parameter that denotes the tradeoff between offloading cost and performance (queue length). Contrary to most common optimisation offloading mechanisms, this model does not have knowledge of the task arrival rates beforehand, so as to accommodate dynamic and bursty environments. Instead it makes offloading decisions in a time-slotted manner, where parameters such as the number of tasks change per time-slot. Notably, the proposed algorithm is distributed for reasons of performance, where the optimisation problem is decomposed to subproblems per IoT application that run concurrently in a distributed way. Still, however, the decision needs to be taken and communicated externally per time-slot.

In (Wang, Liang, Zhang, Zheng, Arif, Wang & Jin 2020), Wang et al. classify the IoT applications in four categories, according to their flow and computational load needs and the offloading decision is between edge devices and a cloud. The decision on where to offload is based on predefined thresholds of the maximum execution time and the minimum energy a task or a device must have respectively, according to their application classification. The experimental results were conducted in a simulated environment, where parameters such as CPU cycles and bandwidth were predefined and it is not clear how these could be measured in a real environment, which could require probing.

In (Shah-Mansouri & Wong 2018), Shah-Mansouri and Wong model the competition between IoT devices for computation resources as a game where each users/IoT device tries to maximise the cost reduction achieved when offloading a task, in response to the other user's strategies and they propose an algorithm that can achieve Nash equilibrium (NE) and a near-optimal resource allocation algorithm that can achieve an near NE solution at an acceptable response time. The experiments conducted in a simulated environment where metrics such as CPU clock speed are provided randomly and uniformly

between two constant limits.

In (Samie et al. 2016), Samie et al. proposed a computation offloading algorithm that optimises the battery of an IoT device under specific bandwidth constraints, combined with a centralised bandwidth allocation mechanism. They considered different offloading levels according to the different computations stages that an IoT application could have. The experiments were conducted using real IoT nodes and a simulated network environment with predefined throughput values.

Ning et al. (2018) proposed a partial computation offloading algorithm for IoT applications that have module parts that can be offloaded or need to be processed locally, assuming that the dependencies between the different modules are of linear sequence. The decision is made at the closest edge node, as a iterative heuristic of a mixed integer linear programming problem that takes into consideration the resource competition between IoT users, and more specifically it tries to minimise the total response time which comprises of the processing and transmission times. This model does not consider energy consumption or the time it takes for the results to be returned to the IoT device.

Li et al. (2019) introduced a hybrid computing solution and resource scheduling strategy for a smart manufacturing environment. It is applied in industrial IoT with real-time restrictions, where each task needs to finish within a specific deadline. It takes into consideration the transmission times (sending input file and transmitting back the result), processing queuing and processing time. If the time constraints are not met by a single edge server, it also proposes cooperation between multiple edge servers to achieve the time constraints by offloading sub-tasks of a task to multiple edge servers simultaneously. The proposed mechanism is evaluated in a real testbed, representative of a small scale IoT environment and is compared against some baselines (”cloud server computing” and ”ordinary edge computing without scheduling”). This work does not take into consideration the execution at the IoT level, and also the energy as a metric, although the experiments showed that by reducing time, energy is also reduced, although not in

an optimal way.

In (Ko et al. 2019), the authors proposed an energy-efficient cooperative computation algorithm (EE-CCA) for IoT devices. EE-CCA (controller) sits at the edge of the network collecting information and requests from nearby IoT devices in order to divide the available devices into pairs and conduct the optimal offloading decision table by formulating Constraint Markov Decision Process. Each pair of IoT devices decide whether to offload some parts of their tasks to the opponent with the consideration of their energy harvesting probabilities, task occurrence rates based on the constructed optimal table. However, transmission time between each pair is not considered part of making the decision. Also, neither edge nor cloud servers are part of the offloading sites.

In (Zhao & Zhou 2019), a selective offloading technique between local and server devices (edge, and cloud) called ABSO is proposed. ABSO tries to predicate the future usage of available devices using ARIMA time-series prediction algorithm in order to estimate the completion time with satisfying a formulated optimisation problem (branch-and-bound algorithm). However, ABSO makes the decision without using a cost function for tradeoff purposes between time and energy. Instead, they focus on the time only for the sensitive tasks, and only energy for those not sensitive. Also, the authors did not mention how the prediction has been carried out.

Abro et al. (2019) proposed a joint energy-efficient task assignment mechanism (JEETA) for IoT applications. JEETA tries to minimise the energy consumption for the IoT device, but without considering task/partition response time. To do so, the user submits the workflow of an application to a master node that is based at the edge of the network (centralised). The master node categorises the application workflow into two disjoint sets of sequenced tasks (local and remote sets) based on the tasks data sizes. Then the master node assigns tasks to the available VMs (each edge server has several VMs with different speeds) in which each task/partition is assigned to a VM with the lower power consumption, also each VM can only be assigned to one task. It is worthily

mentioned that the partitioning is carried in the design stage and in a static way.

Misra et al. (2019) propose three tiers offloading architecture to address the offloading problem in nearby devices, edge and cloud. The authors applied the concept of Auction theory to make the offloading decision by allowing each local device (the bidder) to bid mostly based on its local information and the incomplete information of the neighbour devices. And this is due to the decentralisation of making the decision and reducing the communication overhead. In the first tier, a discovery procedure is carried out by each device that wants to offload tasks, which are partitioned and profiled statically beforehand, to identify mobile devices within its range and their states(i.e. clock speed, remain energy and the available bandwidth) in order to make the decision to either nearby devices or to another destination in the next level. In the second tier, the source mobile device carries another discovering procedure to search nearby cloudlets with their configurations and their states in order make the decision of offloading tasks to a cloudlet or move to the third tier in which offloading the task to a distant cloud. In each layer, the decision is made by using the Auction theory based on the static partitioning and profiling, and devices' states that are provided by the discovery phase. However, the centralised version in this study leads to extra overhead in terms of the response time of making the decision, and energy consumption of performing the discovery phases especially with a large number of available nearby devices (mobile device and cloudlets). Also, there is no evidence on how the discovery phase is carrying out, and what is overheads that result from searching of devices' states and availability

In (Hossain et al. 2020) a collaborative task offloading scheme is proposed to handle latency-sensitise tasks and utilise the system resources more efficiently. The system has three layers of execution: local, edge and cloud in a hierarchical structure. If the end-device has no available resources to execute a task (i.e. task workload is higher than the available capacity), three offloading decisions are available which are: offloading the task to the nearest SBS-MEC; execute the task collaboratively between the mobile

device and nearest SBS-MEC; or also collaboratively but between SBS-MEC server and remote cloud. The decision is made based on task execution and transmission times and the capacity of each device. The decision is simple and does not consider energy consumption.

As previously mentioned, external approaches present central points of failure and can lead to excessive communication overheads (Wang, Wang, Huang, Song & Qin 2020). Their weakness is at the point of decision making as offloading decision tasks from all the network are queued at one or more external devices. In response, researchers' attention has turned to decentralised/autonomous decision approaches, where the IoT devices themselves have the autonomy to take offloading decisions without the need to wait for an external entity to coordinate or make the decision for them.

Sheng et al. (2019) proposed a semi-autonomous offloading mechanism, where the offloading decision is taken at each IoT device individually using deep learning based on their own network and resource conditions, and having an extra level of control at an edge node to coordinate the IoT devices and optimise the overall network environment. For the latter, they use federated learning where the deep learning training is happening in a distributed manner at each IoT device and updates are communicated back to the edge node.

Khoda et al. (2016*b*) have proposed ExTrade, which introduced a speed factor representing how much faster the cloud is than the local device at executing a given task. To do so, it performs periodic probing whereby it measures the time it takes a small task to run on the cloud. Similarly, it periodically transmits a small file to estimate the corresponding transmission time between local device and cloud. For the prediction of the local execution time, it uses the previous execution times and a simple statistical regression technique. The decision is then taken based on a metric that combines the expected energy cost and response time against a predefined threshold.

Benedetto et al. (2019*b*) have proposed MobiCopIoT, which periodically samples the

network every 15 minutes or when a network configuration change is detected. Again, the prediction of the execution time is based on the historical data stored for each device, but this time the approach is based on a nearest neighbour algorithm for finding the records that most resemble the new task. Comparison against different scenarios has shown that the paradigm of allowing developers to deploy their code on both distant clouds and proximate nodes located on the edge can have considerable energy and time benefits.

The offloading mechanism "Ternary Decision Maker" (TDM) proposed in (Lin et al. 2015*a*) decides between local execution on CPU, local execution on on-board GPU and on the cloud. Static parameters such as power consumption, speed of memory and CPU for the mobile device and co-processor are measured once when the algorithm first runs, while dynamic parameters such as memory and network bandwidth, input and output size, cloud speed and task execution time are measured or estimated at runtime. The execution times of all devices are estimated using their stored static parameters and by probing for the dynamic parameters. For example, the cloud speed is estimated by probing the cloud to run a small program and the memory bandwidth is estimated by measuring the time of accessing a large amount of data stored in the memory. The transmission time is predicted by probing each destination device (using the ping command, sending 6000 bytes of data) whenever a task needs to be offloaded.

Jalali et al. (2019*c*) proposed a dynamic offloading mechanism where each IoT device predicts performance parameters of a task based on information received by other IoT devices. Each IoT device broadcasts their available computation resources (such as CPU, memory, transmission rates and battery levels) and based on this information they apply multiple machine learning techniques to learn the current state of the system and predict the performance parameters of running or offloading a task in edge or cloud devices. Of course these machine learning techniques introduce computation and energy overheads themselves as is the broadcasting of the information needed for the

prediction.

What these dynamic and autonomous mechanisms have in common is that they opt for lightweight decision making algorithms to minimise the decision making overhead, but also that they require in order to be able to be applied to real environments is a probing phase to estimate parameters such as the processing time or network delays. This probing phase is by itself a cause for increase of the response time and can be highly inefficient if it has to be repeated by several entities in a decentralised setting. Here, we adopt the same logic of a lightweight algorithm for decision making, but we enhance it with the concept of age of knowledge (AoK), which removes the need for probing.

For ease of reference, most of the mentioned works above are summarised in Table 4.1 based on processing destinations (IoT, Edge or Cloud); whereabout the offloading decision carried out: in the IoT(decentralised) or on an external device(centralised); evaluation methodology (simulation or real-testbed); and whether probing required or not.

4.3 Probeless decentralised offloading decision support

In this section we describe our ***P***robe***L***ess **M**ulti-crit**E**ria **D**ec**I**sion support me**C**hanism for **I**oT offloading(*PL-MEDICI*) which makes an offloading decision in a decentralised manner, at each IoT device independently. It uses lightweight ways of estimating the real-time transmission and processing times without the need of information exchange, i.e. without the need for probing, and by taking into consideration how fresh the information used is, using the age of knowledge (AoK) parameter.

Table 4.1: Published work on IoT Computation Offloading

Publication	Possible Processing Destination(s)	Decentralised	Evaluation Platform	Probeless
(Liu et al. 2017)	IoT, Edge, Cloud	✓	Simulation	✗
(Park et al. 2020)	IoT, Edge	✗	Simulation	✗
(Chen, Zhang, Zhang & Chen 2018)	Edge	✗	Simulation	✗
(Wang, Liang, Zhang, Zheng, Arif, Wang & Jin 2020)	Edge, Cloud	✗	Simulation	✗
(Shah-Mansouri & Wong 2018)	IoT, Edge, Cloud	✓	Simulation	✗
(Samie et al. 2016)	IoT, Edge	✓	Real testbed	✗
(Ning et al. 2018)	IoT, Edge, Cloud	✗	Simulation	✗
(Sheng et al. 2019)	IoT, Edge	✓	Simulation	✗
(Khoda et al. 2016*b*)	IoT, Cloud	✓	Real testbed	✗
(Benedetto et al. 2019*b*)	IoT, Edge, Cloud	✓	Real testbed	✗
(Lin et al. 2015*a*)	IoT, Edge, Cloud	✓	Real testbed	✗
(Jalali et al. 2019*c*)	IoT, Edge, Cloud	✓	Real testbed	✗
(Ko et al. 2019)	IoT	✗	Simulation	✗
(Zhao & Zhou 2019)	IoT, Edge	✓	Simulation	✗
(Abro et al. 2019)	IoT, Edge, Cloud	✗	Simulation	✗
(Misra et al. 2019)	IoT, Edge, Cloud	✓	Simulation & Real testbed	✗
(Hossain et al. 2020)	IoT, Edge, Cloud	✓	Simulation	✗
MEDICI	IoT, Edge, Cloud	✓	Simulation	✗
PL-MEDICI	IoT, Edge, Cloud	✓	Real testbed	✓

4.3.1 Estimation Phase

In the estimation phase, PL-MEDICI uses lightweight techniques to estimate the real-time and time-varying values needed in order to make an offloading decision. It therefore predicts the response times and energy consumption of each IoT task for all devices. Of course, as previously seen, the response time consists of processing and transmission times which need to be estimated separately, as described in the following sections.

4.3.1.1 Estimating Processing Time

Several studies have addressed the problem of estimating/predicting the execution/processing time of a task/process/application using historical observations. Some of them, in order to reduce overheads apply simple calculations, such as the time average on historical (Kemp 2014, Kosta et al. 2012), or use simple mathematical models (Fan et al. 2012, Krishnaswamy et al. 2002, Jaddoa et al. 2020). Although these methods may work well

for small and simple tasks, which have constant characteristics and inputs, they might not be very accurate for more complex ones with varied size inputs. For IoT tasks, which could involve different input parameters, e.g. based on the physical environment they are monitoring, more advanced techniques are needed. The use of machine learning techniques for predicting the processing time of an IoT task has been extensively used (Alam et al. 2019, Alelaiwi 2019, Park et al. 2020), however, this comes at a cost, as it introduces computational overheads, which might be unacceptable in decision support for IoT environments.

To estimate the processing time $T_{n,D}^{proc}$ of an IoT task n on device D in an accurate and lightweight manner, we experimented with multiple machine learning and statistical methods. Here, for simplicity we will use parametric regression (James et al. 2013) to estimate the total processing time of future tasks of various task sizes X_n based on historical data from previous tasks, similar to (Zhang et al. 2008, Ostertagová 2012).

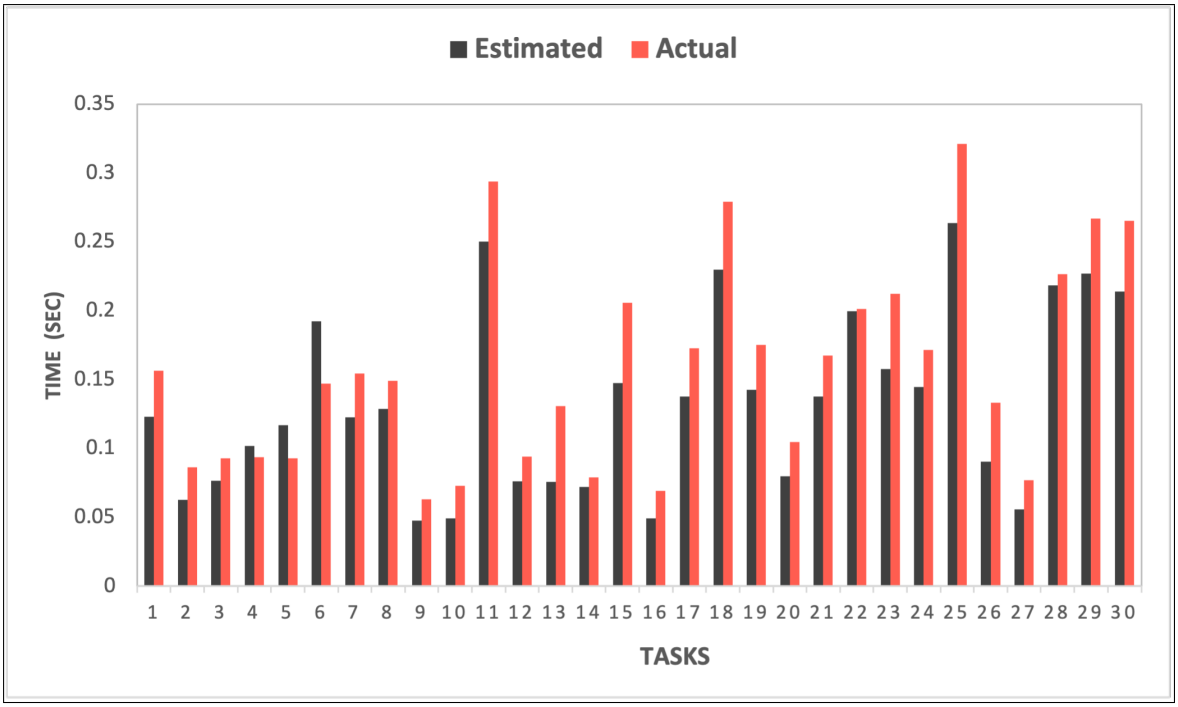
$$T_{n,D}^{proc} = \zeta_{0,D} + \zeta_{1,D}X_n + \zeta_{2,D}X_n^2, ..., \zeta_{r,D}X_n^r + \in_1 \tag{4.1}$$

where X_n is the size of task n, the coefficients $(\zeta_{0,D}, ..., \zeta_{r,D})$ are the estimates of the model coefficients which represent unknown constant values of the polynomial regression model and $\in_1$ is an unobserved random error term(James et al. 2013). The values of coefficients $(\zeta_{0,D}, ..., \zeta_{r,D})$ are calculated using the previously stored historical values of the response times reported by each device D that processed previous tasks. The degree of the polynomial regression model r can be chosen based on the complexity of the IoT applications. For our applications, we have empirically chosen polynomial regression model of second degree ($r = 2$), similar to (Meurisch et al. 2017).

To test its appropriateness, we have compared our predicted processing time to the actual processing time for a face detection application. We run experiments, with different values for the application characteristics (e.g. in the case of face recognition, the

height and width of the input image) for 30 tasks which is a big enough number to allow us to show the accuracy of our prediction model. As illustrated in 4.1 and showing in Table 4.2a, our results showed mean absolute percentage error of 23% and standard deviation 3%.

Figure 4.1: Prediction accuracy for processing time



4.3.1.2 Estimating transmission time

To predict transmission times in a network, it is common to assume that the throughput and latency remain unchanged throughout the runtime of an experiment (Lyu et al. 2018, Alam et al. 2019, Liu et al. 2018, Alelaiwi 2019, Jaddoa et al. 2020). However, in a real-life situation, and especially for IoT environments where communication with remote servers is common, the network conditions constantly change, and these assumptions may lead to inaccurate network delays.

Therefore, to be able to deal with the unpredictability of a network and estimate

throughput more accurately, researchers typically employ probing, which entails sending dummy traffic regularly to obtain network performance measurements. For example, Lin et al. (2015*a*) use ICMP echo (Ping) times to estimate the bandwidth. This however, introduces delay overheads, and has been proved to be unrealistic (Patterson 2004), as it usually overestimates the throughput. In order to have a better estimation of the throughput, one must send a large enough file, transmitted over a longer period of time than a simple Ping probing (Salcedo et al. 2018, Ibrahim et al. 2012, Khoda et al. 2016*b*, Benedetto et al. 2019*b*), which further increases the overheads, in terms of time and energy (Meurisch et al. 2017).

For our proposed mechanism, we have employed a moving average (smoothing) technique to predict the end-to-end throughput between the IoT device and the edge/cloud devices, for both sending and receiving data, similar to (Bors & Nasios 2009). Instead of probing, we obtain the transmission times every time that an offloading decision has been made (i.e. measure the time it took for the offloaded task's input/output files to reach the destination device/IoT device respectively), and use those to calculate the throughput values for the smoothing technique.

The average end-to-end throughput between the devices for transmitting ($\hat{R}^x_D$) or receiving ($\hat{R}^r_D$) data is estimated as follows (Härdle 1990):

$$\hat{R}^x_D = m^x_D(i) + \in_2, \tag{4.2}$$

$$\hat{R}^r_D = m^r_D(i) + \in_3 \tag{4.3}$$

The function $m_D(i)$ is a moving average (smoothing) function, where i is the number of previous historical values that we take into consideration for device D and $\in_2, \in_3$ are

an error terms. $m_D(i)$ can be computed as follows (Härdle 1990):

$$m_D^x(i) = \frac{\sum_{i=1}^{n} W_i * R_{D,i}^x}{\sum_{i=1}^{n} W_i}, \tag{4.4}$$

$$m_D^r(i) = \frac{\sum_{i=1}^{n} W_i * R_{D,i}^r}{\sum_{i=1}^{n} W_i} \tag{4.5}$$

Where W_i is the weighted function assigning higher weights to the most recent measures, and lower to the least recent. In our experiments, described in the next section, we used the four most recent throughput measurements ($i = 4$) and gave a weight of $W_1 = 0.4$ for the most recent measurement ($R_{D,1}^x, R_{D,1}^r$), $W_2 = 0.3$ for the second most recent ($R_{D,2}^x, R_{D,2}^r$), $W_3 = 0.2$ for the third most recent and $W_4 = 0.1$ for least recent, both for sending and receiving data. By using this model, the dynamic changes and fluctuations of the network can be better captured.

Therefore, our estimation of the time it takes to transmit/receive a task n from/to the IoT device I to/from another device $D \neq I$:

$$T_{n,D}^x = \frac{d_n^x}{\hat{R}_D^x} \tag{4.6}$$

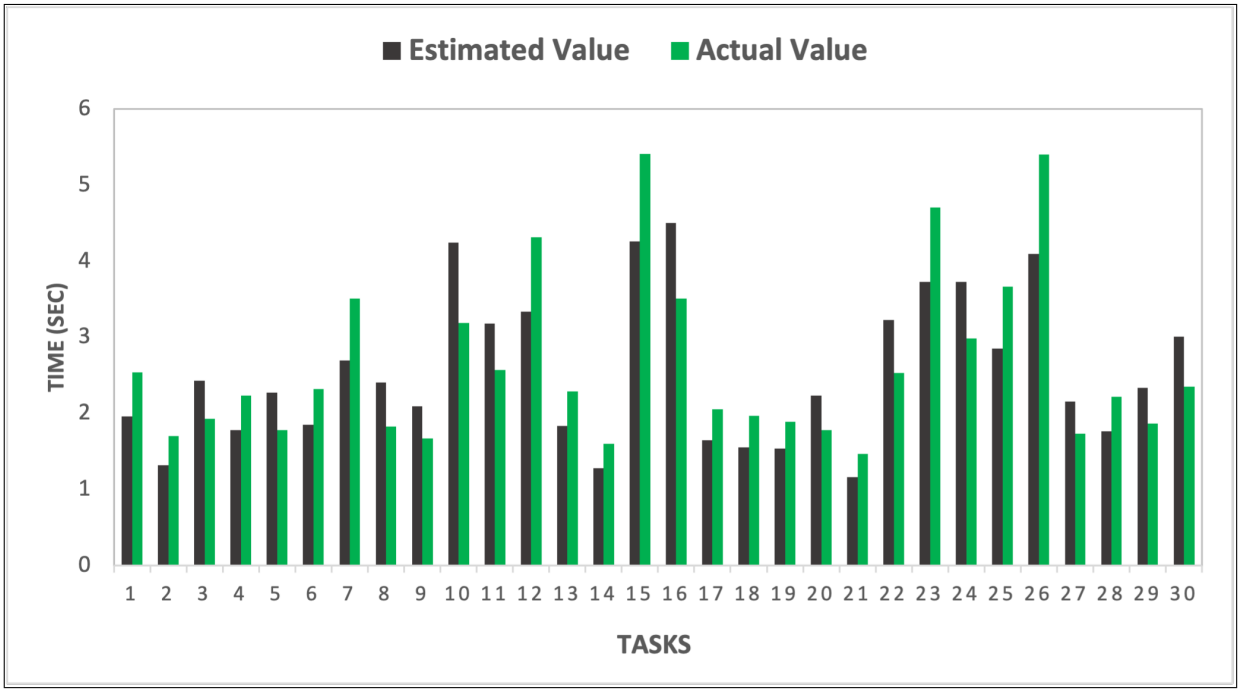
$$T_{n,D}^r = \frac{d_n^r}{\hat{R}_D^r} \tag{4.7}$$

where d_n^x, d_n^x are the sizes of the input and output data respectively that task n is associated with.

To test the accuracy of the above model, we have conducted some experiments comparing our predicted transmission times to the actual transmission times when sending 30 different images between two devices. Our results showed mean absolute percentage error of 20 % and standard deviation 8% (see Figure 4.2 and Table 4.2b) which are

lower or comparable to estimations conducted in the literature (Lin et al. 2015*a*, Khoda et al. 2016*b*, Benedetto et al. 2019*b*).

Figure 4.2: Prediction accuracy for transmission time



4.3.1.3 Estimating energy consumption

After we predict transmission and processing times, energy consumption can be estimated based on the power consumption of the IoT device, similar to 3.3.4.

If a 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:

$$E_{n,I}^{local} = P_I^{proc} T_{n,I}^{proc} \tag{4.8}$$

where P_I^{proc} is the average power consumed when the processor of the IoT device is busy.

Table 4.2: Prediction accuracy

(a) Processing time (s)

Task ID	Estimated value	Actual value
1	0.123	0.157
2	0.063	0.086
3	0.077	0.093
4	0.102	0.093
5	0.117	0.093
6	0.192	0.147
7	0.122	0.155
8	0.129	0.149
9	0.048	0.063
10	0.049	0.073
11	0.250	0.294
12	0.076	0.094
13	0.076	0.131
14	0.072	0.079
15	0.147	0.206
16	0.049	0.069
17	0.138	0.173
18	0.230	0.279
19	0.142	0.175
20	0.080	0.105
21	0.138	0.167
22	0.200	0.201
23	0.157	0.212
24	0.145	0.172
25	0.263	0.321
26	0.091	0.133
27	0.056	0.077
28	0.218	0.226
29	0.227	0.267
30	0.214	0.265

(b) Transmission time (s)

Task ID	Estimated value	Actual value
1	2.0	2.5
2	1.3	1.7
3	2.4	1.9
4	1.8	2.2
5	2.3	1.8
6	1.9	2.3
7	2.7	3.5
8	2.4	1.8
9	2.1	1.7
10	4.2	3.2
11	3.2	2.6
12	3.3	4.3
13	1.8	2.3
14	1.3	1.6
15	4.3	5.4
16	4.5	3.5
17	1.6	2.1
18	1.5	2.0
19	1.5	1.9
20	2.2	1.8
21	1.2	1.5
22	3.2	2.5
23	3.7	4.7
24	3.7	3.0
25	2.9	3.7
26	4.1	5.4
27	2.2	1.7
28	1.8	2.2
29	2.3	1.9
30	3.0	2.4

If task n is offloaded to a target device D other than the IoT device, then the energy consumed by the I device $E_{n,I}^{off}$ consists of the energy consumed for sending the data to D, the energy for remaining idle while waiting, and the energy for receiving the result back from D:

$$E_{n,I}^{off} = P_I^x T_{n,D}^x + P_I^{idle} T_{n,D}^{proc} + P_I^r T_{n,D}^r \tag{4.9}$$

where P_I^{idle} is the average power consumed when the processor of the IoT device is idle, and P_I^x and P_I^r denote the power consumption when the IoT device is transmitting and receiving data respectively.

Summarising in a single expression, the energy cost of IoT device of processing task n is:

$$E_{n,D} = \mathbb{1}[D = I] E_{n,I}^{local} + \mathbb{1}[D \neq I] E_{n,I}^{off} \tag{4.10}$$

4.3.2 Decision Phase

The decision on which device D should be selected to process a task n generated by an IoT device is taken based on a weighted cost metric similar to our MEDICI mechanism in (3.14), where the input values are provided by the estimation phase described previously.

To compensate for the fact that the response time and energy values of this equation are estimations based on historical data from previous decisions, we have enhanced the goal function to include an additional parameter which we denote as Age of Knowledge (AoK).

4.3.2.1 Age of Knowledge (AoK)

Although the use of probing covers the need for continuous and up-to-date historical data for the purpose of maximising estimation accuracy by artificially adding values to

the history of previous times, it is often the case that the same task may have recently run on the same device and this information can be good enough for that device. Hence, for applications that run often, it may be preferable to take into account the age of this knowledge as opposed to requesting and waiting for an update through probing.

Generally, the age of information (AoI) is defined as the time elapsed since the data was generated (Zhong et al. 2019) in a device. In this work we introduce the concept of Age of Knowledge (AoK) which we define as *the time elapsed since the latest historical data concerning a device D was recorded in the IoT device I*, to capture the age of the historical information/knowledge that I holds for each possible device D that can run tasks from I and make more informed decisions. Contrary to the AoI, AoK captures also the time it takes to report the information to I, so that we can consider whether it is up-to-date knowledge or not and to avoid clock synchronisation issues in different devices.

When a new task n is generated in device I, the AoK for each possible device D is calculated as:

$$A_D = t - t_D \tag{4.11}$$

where t is the current timestamp and t_D is the timestamp recorded on the IoT device I when it had last received information from device D and updated its records.

4.3.2.2 PL-MEDICI Decision

Therefore, if we take into consideration the AoK, the weighted goal metric G used to decide which device should process a task n becomes:

$$G_{n,D} = (\beta \bar{T}_{n,D} + (1-\beta)\bar{E}_{n,D})(1-\gamma) + \gamma \bar{A}_D \tag{4.12}$$

where, parameters $\bar{T}_{n,D}$, $\bar{E}_{n,D}$, $\bar{A}_D$, denote normalised values of the estimated response time, energy and AoK values, calculated in the estimation phase Section 4.3.1.

$T_{n,D}$ is the response time, meaning the total time it takes for n to be transmitted ($T^x_{n,D}$), processed ($T^{proc}_{n,D}$) and its result to be returned back to the IoT device ($T^r_{n,D}$):

$$T_{n,D} = T^x_{n,D} + T^{proc}_{n,D} + T^r_{n,D} \tag{4.13}$$

Of course, in the case that $D = I$, where the task is not offloaded but is processed locally, then $T^x_{n,I} = T^r_{n,I} = 0$.

$E_{n,D}$ is the total energy consumed at I when task n is processed in device D as estimated in the estimation phase Section 4.3.1.3).

A_D is the AoK parameter, showns previously in Section 4.3.2.1.

Note that $\beta \in [0, 1]$ is a weight denoting the application's preference in minimising response time over energy, defined at each IoT device, while $\gamma \in [0, 1]$ is a weight denoting the degree of consideration of the AoK in the offloading decision. Intuitively, a more stable environment puts less emphasis on AoK than a highly dynamic one.

The normalisation technique used to bring $T_{n,D}$, $E_{n,D}$ and A_D into a mutually comparable range of values (since e.g., the energy of an IoT node could be in the range of hundreds of joules whilst the times could be in the range of seconds or less) is as follows:

$$\bar{T}_{n,D} = \frac{T_{n,D}}{\sum_{y \in \mathcal{S}^D_i} T_{n,y}}, \tag{4.14}$$

$$\bar{E}_{n,D} = \frac{E_{n,D}}{\sum_{y \in \mathcal{S}^D_i} E_{n,y}}, \tag{4.15}$$

$$\bar{A}_{n,D} = \frac{A_{n,D}}{\sum_{y \in \mathcal{S}_i^D} A_{n,y}}. \tag{4.16}$$

For a task n initiated at I, the device chosen to perform its processing is the one that minimises the goal metric for $D \in \{I, \mathcal{S}^E, \mathcal{S}^C\}$, as per equation 4.12:

$$C_n = \underset{D \in \{I, \mathcal{S}^E, \mathcal{S}^C\}}{\operatorname{argmin}} G_{n,D} \tag{4.17}$$

This decision is taken by each IoT device for its own applications independently, based on knowledge gathered by previous processed tasks.

4.4 Real testbed experimental setup and implementation

In this section we present the implementation and configuration of our real-testbed and present the evaluation of our probeless offloading mechanism, comparing it to three decentralised offloading strategies proposed in the literature, and for three different applications.

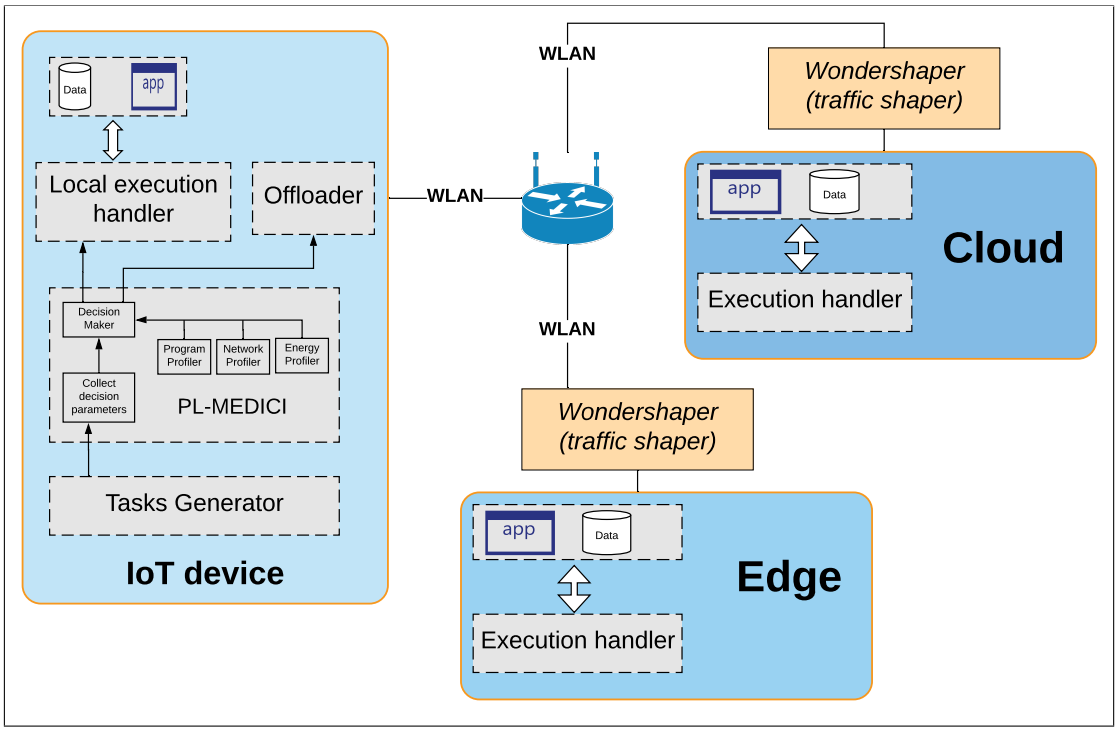
4.4.1 PL-MEDICI Architecture

We have chosen to implement PL-MEDICI as a decentralised decision mechanism, where the decision is taken locally at the IoT device. This choice was made for three reasons firstly, to reduce the communication overheads introduced by requesting a decision from the edge and send the result back to the IoT device. Secondly, for the sake of fair comparison to the existing work from the literature as they are decentralised. Thirdly,

running PL-MEDICI locally, not at the edge, would help to avoid the disadvantages of having a central decision making place (e.g. single place of failure, scalability, etc.)

Figure 4.3 shows the architecture of our mechanism and its communication with the edge and cloud. The Task Generator creates the different tasks of the application running on the IoT device. The Program, Network and Energy profilers, are components to store the required historical information and estimate the corresponding values needed by the decision maker which calculates the decision metrics and provides the chosen device to the offloader, which carries out the offloading (i.e. sending the application data and receiving the results back). All devices (IoT, edge, and cloud) have an execution handler, which handles the communication and the processing of the tasks.

Figure 4.3: Overview of the system components



Each IoT device communicates with other devices (Edge and Cloud) through a wireless local area network (WLAN). To emulate the physical distance between different devices, we have different transmission rates (upload/download speeds) between an IoT

device and an edge and an IoT device and a cloud device, as seen in Table 4.3. For this we used a Linux-based package called Wondershaper (Hubert 2002) to control the maximum throughput of each offloading device to reflect a different geographical distance from the IoT device and ensure that each offloading device does not go beyond its maximum allowable transmission rate. Moreover, to make our network more realistic, we added background traffic for both the network and the device processing, using the *random python package*. More specifically, at each IoT device we generated network background traffic by randomly choosing between 10 files of various sizes (ranging from 500 KB to 1.5 MB) and sending them to a randomly chosen device every 5 seconds. For the processing background load, we generated requests to randomly chosen devices (every 3 seconds) to execute a computationally demanding programme (the N-Queen game (Kosta et al. 2012)) of random input values (between 8-12). For both cases, each IoT device picks a device from all possible devices randomly but the same device cannot be chosen twice in a row.

4.4.2 Experimental setup

As shown in Figure 4.4, our setup consists of 6 Raspberry Pi's (PI 3 MODEL B+, 1.4GH-ARMv7) that act as IoT devices, running Raspbian 9.11. Each of them runs one of the three applications (see Section 4.4.3) and our offloading mechanism described before. More specifically, IoT devices 1 (IoT-D1) and 4 (IoT-D4) run Application 1(APP1), IoT devices 2 and 5 run Application 2 and IoT devices 3 and 6 run Application 3.

In respect to energy, each IoT device has three modes for energy consumption: processing, transmission and idle. In order to obtain the average values for these modes, we have run experiments using a Watt's Up Pro meter (Devices 2009) to measures the average power consumption when idle (measured at 2 Watts), transmitting/receiving data (measured at 2.5 Watts) and processing a computationally intensive application

Table 4.3: Specifications of the edge and cloud devices for PL-MEDICI setup

Device	Physical machine CPU	Physical machine cores	MAX cores per container	Max speed of upload/download	Physical machine memory	No. of docker containers
Edge1	Core i5 3.2 GHz	4	3	2 MB	4GB	3
Edge2	Core i5 2.5 GHz	4	4	1.7 MB	4GB	3
Edge3	Core i5 3.2 GHz	6	4	1.5 MB	6GB	3
Edge4	Core i5 3.2 GHz	6	5	1.2 MB	6GB	3
Cloud1	Core i7 2.6 GHz	8	6	900 KB	8GB	3
Cloud2	Core i7 2.6 GHz	8	7	700 KB	8GB	3

(measured at 3 Watts).

We also have four edge servers and two cloud servers that run Ubuntu 20.04 LTS. Each server runs 3 Docker containers(C1, C2, and C3), one for each of the three applications described in Section 4.4.3.

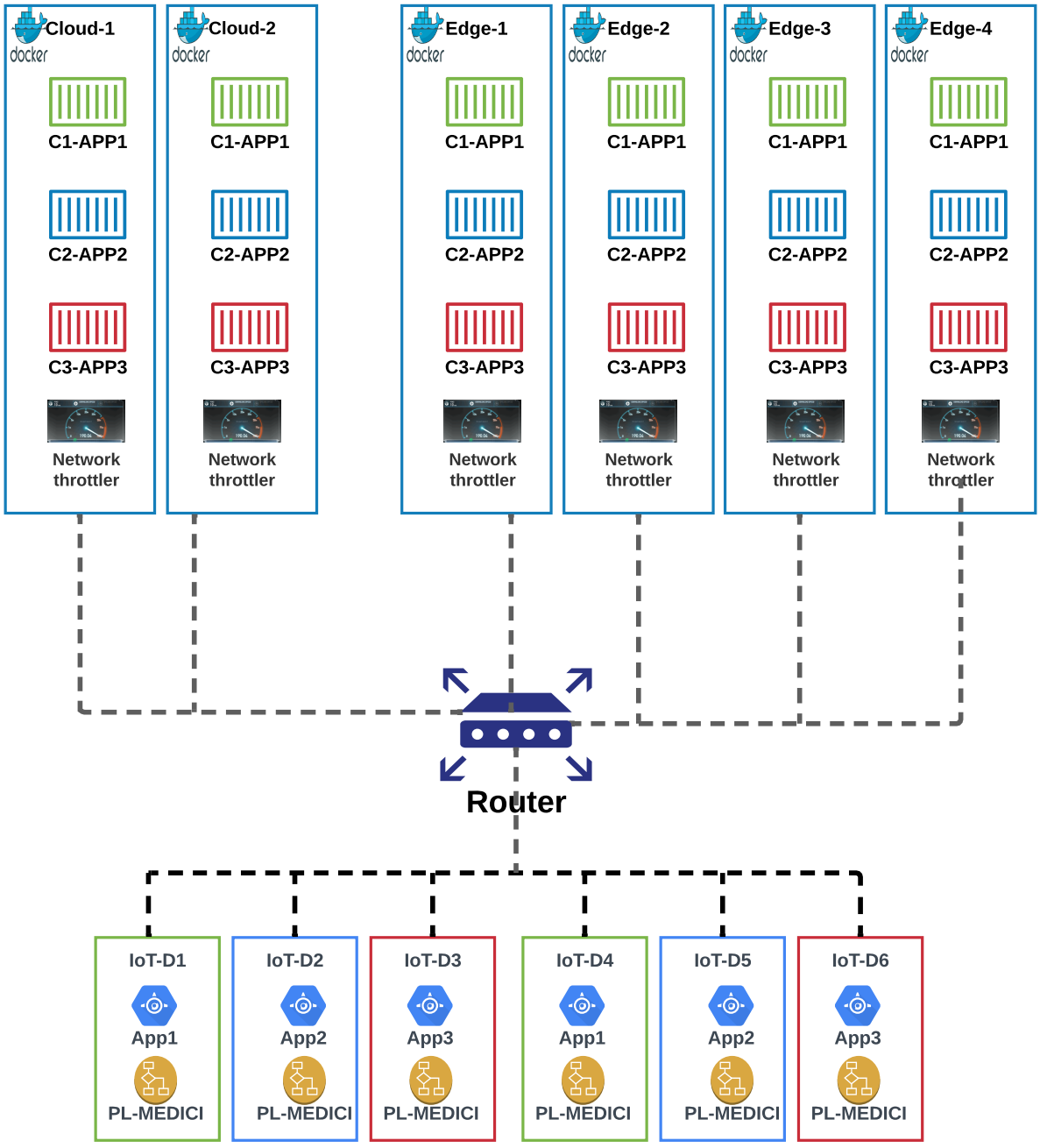
The devices specifications are given in Table 4.3.

4.4.3 Applications

Three different applications have been used in our experiments, with different complexities and input/output size files, typically used in the literature (Khoda et al. 2016*b*, Benedetto et al. 2019*b*, Mahini et al. 2021, Meurisch et al. 2017, Jalali et al. 2019*c*) to represent IoT applications, detailed below:

1. ***Face Detection***. This application could be used by IoT devices such as smart surveillance cameras (Pyzyk 2018) and has been extensively used in the literature as a computationally demanding IoT application (*OpenCV* 2019, Khoda et al. 2016*b*, Jalali et al. 2019*c*, Kemp 2014, Kosta et al. 2012). It takes an image as an input with dimensions $h \times w$ and tries to detect faces in the image and if offloaded,

Figure 4.4: Testbed architecture



it needs to send the original image, which could be relatively big in size. Therefore, this application is a good example of heavy computation and high input data transfer, while the output is of medium size, as it returns a smaller image with just the detected face.

2. ***Video-Based Heart Rate Measurement (VBHRM)***: This application can be considered representative of healthcare IoT (Li, Ding, Liu, Yan, Xu, Gao & Zheng 2018)). It is used to analyse a video of the tip of an individual's finger in order to extract the heartbeat information. This is done by measuring the variations of blood volume through the variations of light absorption or reflection for a stream of picture frames, using the Discrete Fourier Transform (DFT). If offloaded, this application will need to send over the video and thus has a big size input data. Computationally, it is not as demanding as the face detection with medium computation requirements (Meurisch et al. 2017, Pelegris et al. 2010, Yu et al. 2013). The output file is of small size, as it returns a small image that shows the heart rate over time.

3. ***Radix sort algorithm***: This application could be used for sorting in IoT data environments (Kristo et al. 2020) and is a non-comparative sorting algorithm used for time series analysis (Meurisch et al. 2017). It tries to sort elements (e.g. a file consisting of sensor readings) by creating and distributing elements into buckets according to their radix to avoid comparison (Zagha & Blelloch 1991). It is a simple algorithm with linear time complexity, while the input and output files of e.g. sensor readings could be big in size.

In Table 4.4, these applications are summarised in terms of input and output data size, and computation requirements.

The fifth column of the table provides the value of β per application, which denotes its preference in minimising response time over energy. For more time-critical applications,

Table 4.4: Application specifications and requirements for PL-MEDICI setup

Application	Mean Input Size	Mean Task Size	Computation Complexity	Mean Output Size	β	γ
Face detection	600KB	1600*1200 (h*w)	$O(n^2)$	300KB	0.2	0.6
VBHRM	1.5MB	900K (frames)	$O(n^2)$	50KB	0.9	0.2
Radix Sort	1MB	600K (elements)	$O(n)$	500KB	0.5	0.4

such as our healthcare IoT application, we have chosen β to be closer to 1, while for more computationally demanding applications where energy consumption is important, such as the face detection one, β is chosen to be closer to 0. Finally, the last column of the table provides the value of γ for each application, which denotes how important it is to have up-to-date knowledge. We have obtained those γ values empirically by running experiments for each application and found the best γ corresponding to the best goal function G.

4.5 Real testbed experimental results

Here we evaluate our proposed mechanism by comparing it to three other offloading mechanisms: TDM (Lin et al. 2015*a*), ExTrade (Khoda et al. 2016*b*), and MobiCOP-IoT (Benedetto et al. 2019*b*), that are proposed in the literature and that we have implemented on the same testbed.

- ExTrade (Khoda et al. 2016*b*) takes into consideration the tradeoff between energy consumption and performance (execution time) of end-devices and clouds. It uses a statistical regressing technique (smoothing) to estimate the execution time of a task, based on the calculated CPU speed and the previous observations of the local execution time. To calculate the CPU speed, a small sample program is executed periodically in each offloading destination (edge and cloud). To calculate

the execution time on a cloud device it multiplies the local execution time by a speed factor which represents how faster the CPU of the cloud is compared to the local device. The model for ExTrade, described in (Khoda et al. 2016*b*), is for choosing to offload between the local device and one cloud. Because here we have multiple edges and clouds, we calculate the goal function G for all possible offloading devices and check whether the maximum G is above the threshold to decide to offload to the corresponding device or run locally. The threshold was calculated after running a series of experiments, just as in (Khoda et al. 2016*b*) and we chose the ExTrade's parameter α, which denotes the weight factor to tradeoff between the energy and computation (similar to our β value) to be independent of the battery life and equal to $\alpha = 0.5$ throughout the experiments.

- MobiCOP-IoT (Benedetto et al. 2019*b*) aims to minimise task processing times of mobile IoT applications, choosing between local processing, edge and cloud. The decision-maker has two components, the network profilers (in charge of estimating network latency and throughput by sampling the network every 15 min or on request) and the code profilers (estimating the execution time for a given task using historical data, assuming that repeated tasks take the same amount of time to execute).

- TDM (Ternary Decision Maker) (Lin et al. 2015*a*) aims to reduce energy consumption and shorten the response time for mobile device applications, by choosing where to execute a task between an on-board CPU, an on-board GPU and a cloud. Here we have adjusted the mechanism to consider the CPU of different edge and cloud devices. The TDM has two types of parameters, deterministic or independent (measured before execution and obtained from datasheets, such as power consumption, speed of memory and CPU for the mobile device and co-processor) and non-deterministic or dependent (measured during the runtime and saved in a

table called factor table, such as bandwidth, input & output size, speed of the cloud and task execution time). When a task is created, TDM searches the task in the existing factor table. If the task does not exist there, then it is executed locally at the mobile device. Else, the TDM estimates the execution time for the new task based on its previous execution time and a speed factor according to the destination device. It also estimates the transmission time using the *ping* command, sending 6000 bytes of data before each offloading decision. The decision is based on the response time (execution and transmission) as well as the energy consumption. In order to calculate the speed of the processor of a destination device, TDM queries each device to execute a small program that pre-exists in each of them.

In each IoT device, we have generated 100 tasks with various input parameters, that make requests for an offloading decision using an exponential distribution. We have then measured the total response time of each task (which includes processing, network times and decision overheads (OH)) and the energy consumption of each task and each IoT device. The results presented here are the average of 5 runs for each decision approach (i.e. PL-MEDICI, TDM, MobiCopIoT, and ExTrade).

Figures 4.5, 4.6 and 4.7 show on the left the average response time for all tasks for each application, in terms of decision overheads (represented in grey), network time which includes transmission times and network delays when a task is offloaded (represented in blue) and processing times (represented in orange). On the right side we see the energy consumption of the IoT devices for each application in terms of decision overhead (represented in light grey) and in terms of the energy consumed in the IoT device for running a task locally or remotely (represented in green).

We have compared our proposed algorithm (PL-MEDICI) with the three mechanisms described before (TDM, MobiCOP-IoT and ExTrade). We have also compared our mechanism when we do not use the AoK in the decision (PL-MEDICI-noAoK) to see if the lack of recent information has an effect on PL-MEDICI's performance.

Figure 4.5: Response time and energy for App1 (Face detection)

(a) Response time

(b) Energy consumption

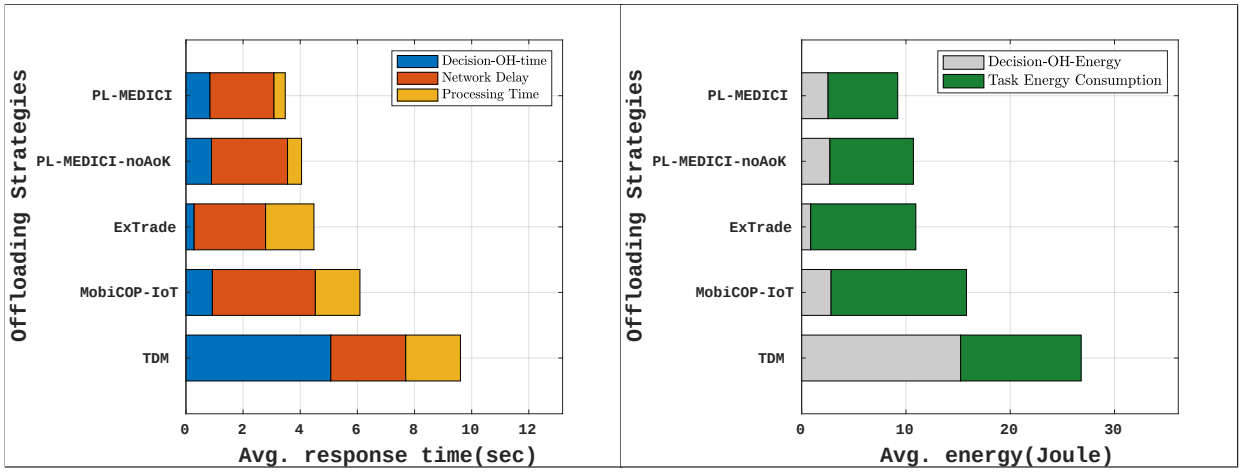


Table 4.5: Response time and energy for App1 (Face detection)

(a) Response time(s)

Delay	TDM	MobiCOP-IoT	ExTrade	MEDICI-noAoK	MEDICI
Decision Overhead-time	5.07	0.92	0.29	0.89	0.84
Network Delay	2.62	3.61	2.50	2.66	2.24
Processing Time	1.91	1.56	1.69	0.49	0.39

(b) Energy consumption(J)

Delay	TDM	MobiCOP-IoT	ExTrade	MEDICI-noAoK	MEDICI
Decision Overhead-energy	15.24	2.82	0.87	2.71	2.54
Task energy consumption	11.56	12.98	10.08	8.01	6.68

Across all cases, the dynamic probeless decision support offered by PL-MEDICI has consistently better overall results for all applications in terms of response time and energy consumption, improving all aspects, meaning the processing time and the network time, while providing low overheads. We can also see that taking into consideration the AoK improves PL-MEDICI's performance. For App 1 the response time is reduced by 16.4% and the energy by 16.3%, for App2 the response time is reduced by 23% and the energy by 21.8%, and for App 3 the response time is reduced by 6.5% and the energy by 5.4% if we include in the decision the AoK metric.

More analytically, comparing to the other mechanisms, in figures 4.5b for the face de-

Figure 4.6: Response time and energy for App2 (VBHRM)

(a) Response time(s) (b) Energy consumption(J)

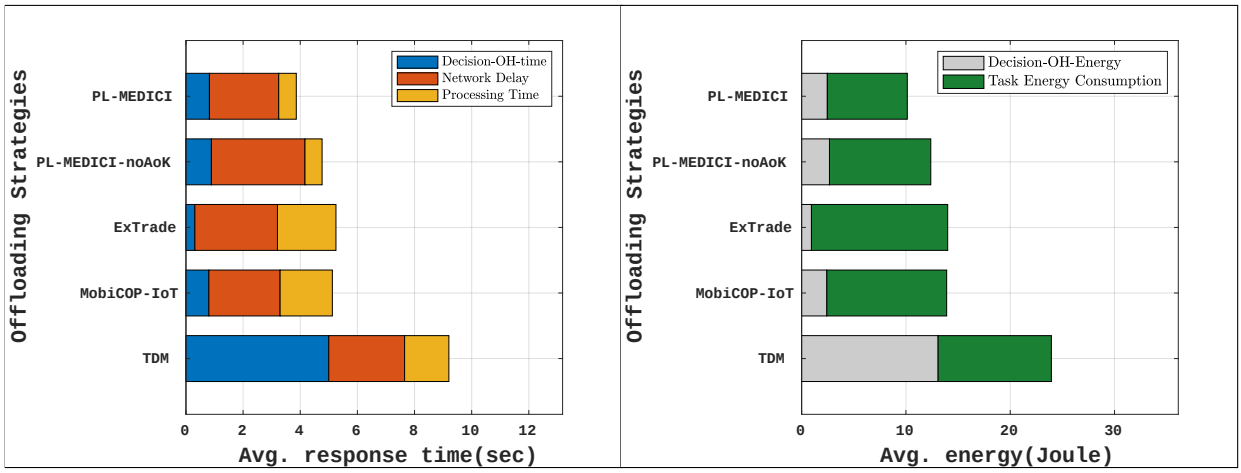


Table 4.6: Response time and energy for App2 (VBHRM)

(a) Response time(s)

Delay	TDM	MobiCOP-IoT	ExTrade	MEDICI-noAoK	MEDICI
Decision OH-time	5.00	0.80	0.31	0.89	0.82
Network Delay	2.65	2.49	2.89	3.28	2.43
Processing Time	1.55	1.83	2.05	0.60	0.61

(b) Energy consumption(J)

Delay	TDM	MobiCOP-IoT	ExTrade	MEDICI-noAoK	MEDICI
Decision OH-energy	13.08	2.41	0.94	2.67	2.46
Task energy consumption	10.86	11.49	13.06	9.71	7.67

Table 4.7: Response time and energy for App3 (Radix Sort)

(a) Response time(s)

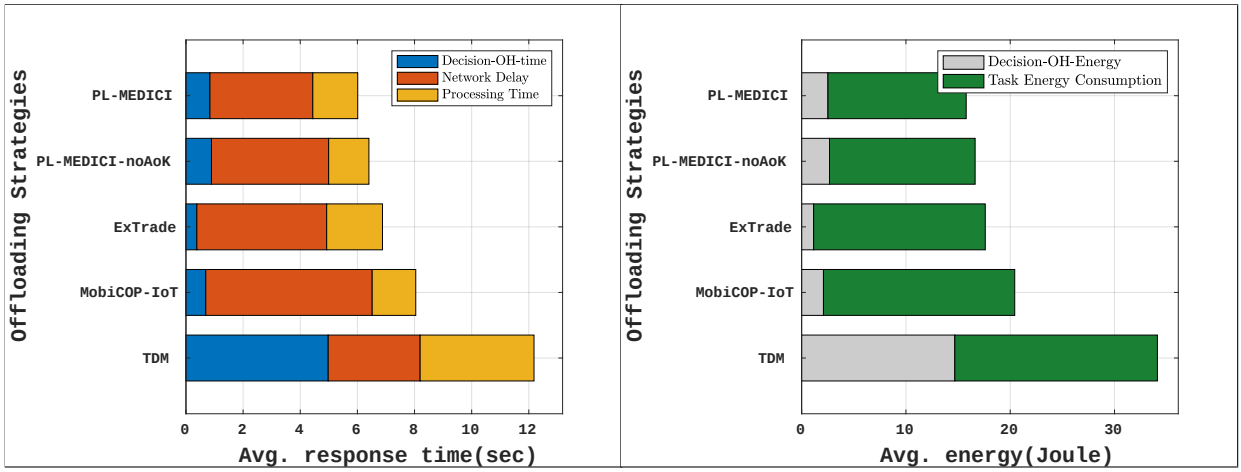
Delay	TDM	MobiCOP-IoT	ExTrade	MEDICI-noAoK	MEDICI
Decision OH-time	4.98	0.69	0.38	0.89	0.84
Network Delay	3.22	5.82	4.54	4.10	3.60
Processing Time	3.99	1.53	1.95	1.41	1.57

(b) Energy consumption(J)

Delay	TDM	MobiCOP-IoT	ExTrade	MEDICI-noAoK	MEDICI
Decision OH-energy	14.69	2.09	1.15	2.67	2.52
Task energy consumption	19.43	18.34	16.46	13.96	13.26

Figure 4.7: Response time and energy for App3 (Radix Sort)

(a) Response time (b) Energy consumption



tection application (App1), which is characterised by large mean task size, and medium input and output sizes, as well as very low β (i.e., strong preference for energy efficiency over response time minimisation), PL-MEDICI achieved a reduction of 27% in energy consumption over the best of the other three offloading mechanisms, which was Extrade. In terms of response time(Figure 4.5a), PL-MEDICI achieved a reduction of 40%.

For the video-based VBHRM application (App2), which is characterised by large input sizes, medium task and output sizes, and has a high value of β (i.e., strong preference for response time), PL-MEDICI achieved a reduction of 20% of response time(Figure 4.6a). In terms of energy (Figure 4.6b), PL-MEDICI achieved a reduction of also 20%.

For the Radix sort application (App3), which is characterised by large input and output sizes, medium task size and has equal preference for energy efficiency and response time minimisation ($\beta = 0.5$), PL-MEDICI achieved a reduction of 17% of response time (Figure 4.7a). In terms of energy (Figure 4.7b), PL-MEDICI achieved a reduction of 23%.

We need to mention here that although PL-MEDICI has low decision overhead, ExTrade's was lower. TDM is the mechanism with the biggest overhead as it probes before each decision to calculate the bandwidth and the speed of the offloading devices.

However, this additional overhead enables PL-MEDICI to reduce the processing and network times by distributing the task more appropriately, as seen in Figures 4.8 - 4.10 and Tables 4.8 - 4.10 which show the average percentage of tasks that were allocated to each device per application. PL-MEDICI appropriately distributes the tasks between the different devices to achieve better performance according to the needs of each application.

Figure 4.8: Percentage of tasks run at each device for App1 (Face detection)

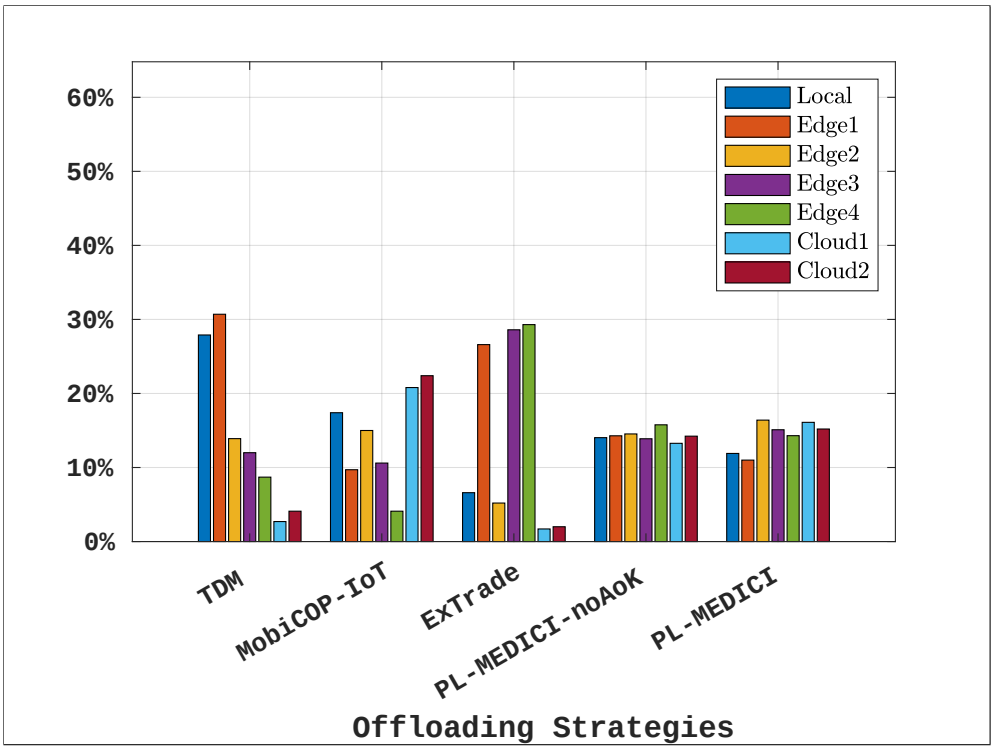


Table 4.8: Percentage (%) of tasks run at each device for App1 (Face detection)

Strategy	Local	Edge1	Edge2	Edge3	Edge4	Cloud1	Cloud2
TDM	28%	31%	14%	12%	9%	3%	4%
MobicopIoT	17%	10%	15%	11%	4%	21%	22%
ExTrade	7%	27%	5%	29%	29%	2%	2%
MEDICI-noAoK	14%	14%	15%	14%	16%	13%	14%
MEDICI	12%	11%	16%	15%	14%	16%	15%

Figure 4.9: Percentage of tasks run at each device for App 2(VBHRM)

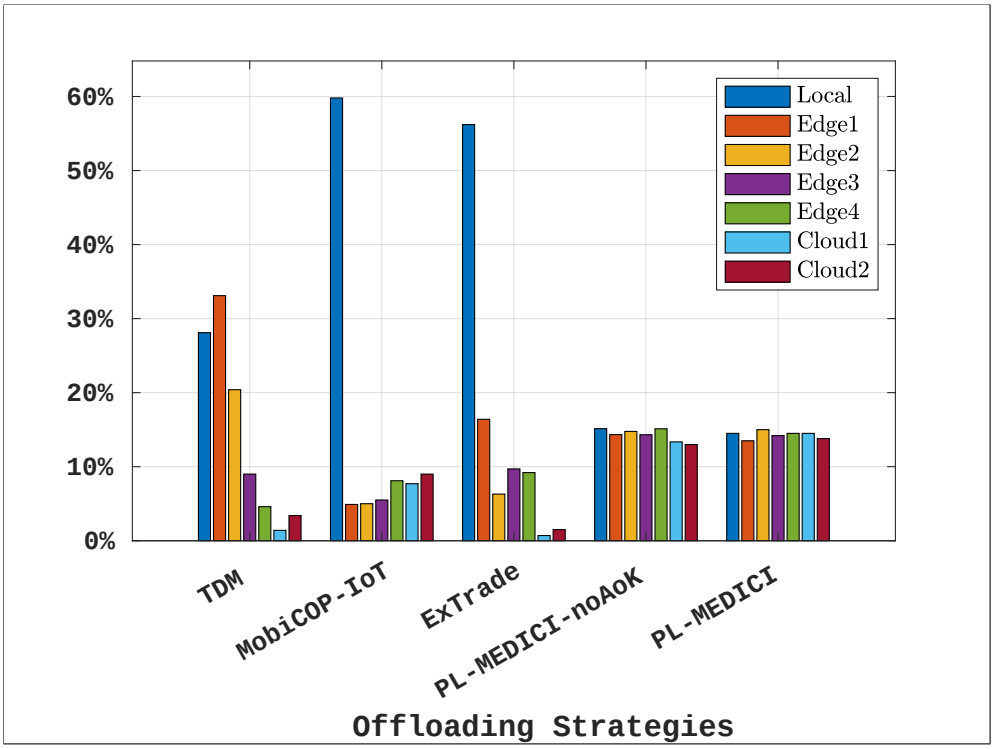


Table 4.9: Percentage(%) of tasks run at each device for App 2(VBHRM)

Strategy	Local	Edge1	Edge2	Edge3	Edge4	Cloud1	Cloud2
TDM	28%	33%	20%	9%	5%	1%	3%
MobicopIoT	60%	5%	5%	6%	8%	8%	9%
ExTrade	56%	16%	6%	10%	9%	1%	2%
MEDICI-noAoK	15%	14%	15%	14%	15%	13%	13%
MEDICI	15%	14%	15%	14%	15%	15%	14%

Figure 4.10: Percentage of tasks run at each device for App3 (Radix Sort)

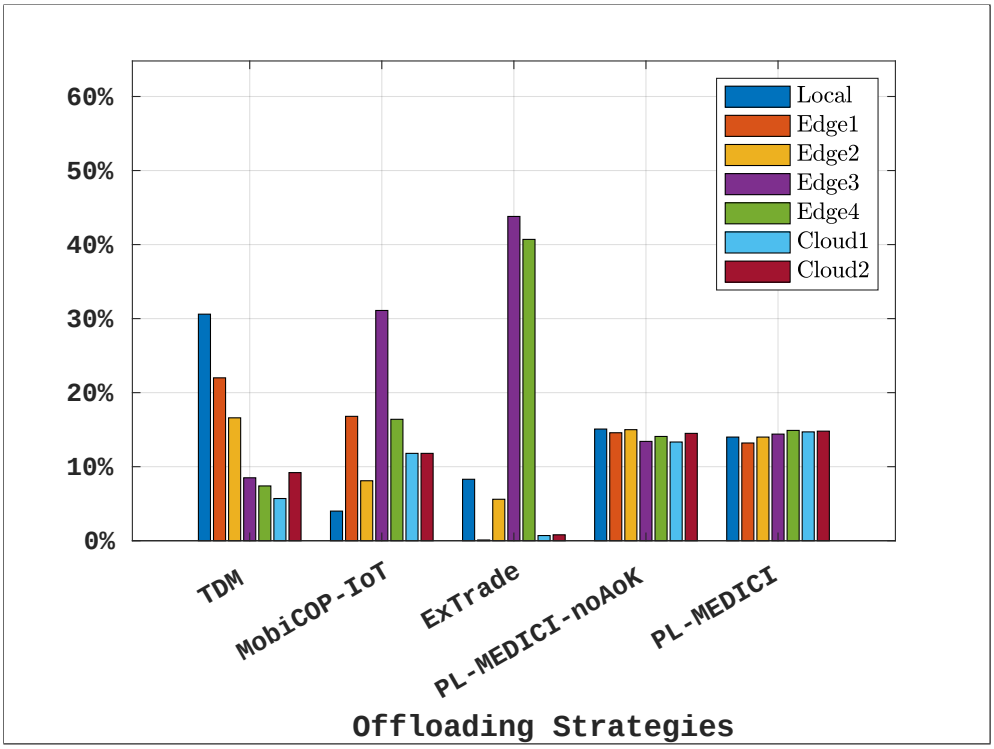


Table 4.10: Percentage(%) of tasks run at each device for App3 (Radix Sort)

Strategy	Local	Edge1	Edge2	Edge3	Edge4	Cloud1	Cloud2
TDM	31%	22%	17%	9%	7%	6%	9%
MobicopIoT	4%	17%	8%	31%	16%	12%	12%
ExTrade	8%	0%	6%	44%	41%	1%	1%
MEDICI-noAoK	15%	15%	15%	13%	14%	13%	15%
MEDICI	14%	13%	14%	14%	15%	15%	15%

4.6 Summary

In this chapter, we have addressed the assumptions that we have made in the previous chapter (Chapter 3) by proposing a dynamic, probeless multi-criteria offloading decision mechanism PL-MEDICI for heterogeneous IoT applications which takes into account multiple parameters such as the processing time of a task, the time it takes to offload its data through the network and the energy consumed by the IoT device. In order to estimate those values our mechanism does not probe the other devices but uses historical data, taking also into consideration the age of the previous knowledge when making its decision.

To evaluate its performance, we have conducted experiments in a real environment, where six raspberry Pis, which act as IoT devices, run three different IoT applications with different characteristics and needs and where the network is emulated to be more realistic and provide dynamic network conditions. We compared our mechanism with three dynamic mechanism that exist in the literature, as well as our mechanism when we do not consider the age of information as a decision criterion. Our experiments showed that our mechanism outperforms all others not only in respect to reducing the overheads, but also by reducing the overall processing and network times and the energy consumption of the IoT devices. Our evaluation also demonstrated the need for

up-to-date information and how choosing an appropriate γ value could further improve the performance of our mechanism. Towards this direction, our future work will involve investigating ways of automatically assigning γ values to different IoT applications.

Chapter 5

Centralised Theoretical Optimal Solution Based on a Priori Information

In this chapter, we formulate, solve and simulate a mixed-integer mathematical program (MILP) problem for benchmarking purposes.

This optimisation problem provides the globally optimal offloading solution when all information is *a priori* known and centrally available. This will allow us to determine the theoretically optimal offloading decisions in order to compare them against our mechanism PL-MEDICI.

First, we state the problem definitions, then the problem formulation and the optimisation solver. And finally, we present the results by comparing the optimal solution given by our centralised MILP optimisation to our mechanism (PL-MEDICI).

5.1 Definition

Let N^I, N^E, N^C, and N^{ALL} be the number of IoT, edge, cloud and all devices respectively, N^A be the number of different IoT applications and N^T the number of tasks.

Let also $\mathcal{S}^I$, $\mathcal{S}^E$, $\mathcal{S}^C$, $\mathcal{S}^{ALL}$ denote the set of IoT, edge, cloud and all devices respectively, $\mathcal{S}^A$ the set of IoT applications and $\mathcal{S}^T$ denote the set of all tasks.

Let also $\mathcal{S}^D_{n,a}$ denote the set of devices which can execute task n of application a, and $\mathcal{S}^T_{a,d}$ denote the set of tasks of application a that can be executed by device d. Let also $T^X_{n,a,d}$ denote the total time needed (transmission time + network queuing time) to send task n of application a to device d, $T^P_{n,a,d}$ denote the processing time of task n of application a from device d and $T^R_{n,a,d}$ the corresponding transmission time that device d needs to send back to the source devices the output file from the execution of task n of application a.

Let $T^T_{n,a,d}$ be the total response time for a task n of application a, which is defined as the total time it takes for a task n to be transmitted and processed in device d and the result to be returned back to the IoT device it was generated at.

Let also $E^T_{n,a,d}$ be the total energy consumption for a task n of application a. Let also β be the weight denoting preference in minimising response time over energy for application a, with $\beta \in [0, 1]$. Let also P^X_d, P^{idl}_d, P^R_d and P^{pro}_d denote the average power consumption of the local IoT device d ($d \in S^I$) when transmitting data, being idle (waiting), receiving data or when processing (busy) respectively.

Let also T^G_n denote the generation time of task n and D^G_n denote the device ID at which task n is generated.

5.2 Formulation

$$\mathcal{S}^D = \cup_{a\in A}\mathcal{S}_a^D, \mathcal{S}^D = \mathcal{S}^E \cup \mathcal{S}^I \cup \mathcal{S}^C, \tag{5.1}$$

$$\text{minimize} \sum_{a\in A}\sum_{d\in\mathcal{S}_a^D}\sum_{n\in\mathcal{S}_d^T}\left\{\beta_a \bar{T}_{n,d}^T + (1-\beta_a)\bar{E}_{n,d}^T\right\} \tag{5.2}$$

$$\text{Subject to } \bar{T}_{n,d}^T = \frac{T_{n,d}^T}{\sum_{y\in\mathcal{S}_i^D} T_{n,y}^T}, \tag{5.3}$$

$$\bar{E}_{n,d}^T = \frac{E_{n,d}^T}{\sum_{y\in\mathcal{S}_i^D} E_{n,y}^T}, \tag{5.4}$$

$$T_{n,d}^T = (T_{n,d}^E - T_{n,d}^S) + Y_{n,d}, n \in S^T, d \in S^I, \tag{5.5}$$

$$E_{n,d}^T = ((T_{n,d}^E - T_{n,d}^S) * P_d^{pro})) + (Y_{n,d} * P_d^{idl}), n \in S^T, d \in S^I, \tag{5.6}$$

$$T_{n,d}^T = T_{n,d}^X * X_{n,d} + (T_{n,d}^E - T_{n,d}^S) + Y_{n,d} + (T_{n,d}^R * X_{n,d}), n \in S^T, \\ d \in \{S^E, S^C\}, \tag{5.7}$$

$$E_{n,d}^T = (T_{n,d}^X * X_{n,d} * P_{\widehat{d}}^X) + ((T_{n,d}^E - T_{n,d}^S) * P_{\widehat{d}}^{idl})) + (Y_{n,d} * P_{\widehat{d}}^{idl}) \\ + (T_{n,d}^R * X_{n,d} * P_{\widehat{d}}^R), n \in S^T, d \in \{S^E, S^C\}, \widehat{d} \in S^I, \tag{5.8}$$

$$T_{n,d}^S = T_n^G + T_{n,d}^X + W_{n,d}, n \in S^T, d \in S^{ALL}, \tag{5.9}$$

$$T_{n,d}^E = T_{n,d}^S + T_{n,d}^P X_{n,d}, n \in S^T, d \in S^{ALL}, \tag{5.10}$$

$$T_{n,d}^S \geq T_{\hat{n},d}^E, \hat{n} = \mathcal{S}_{n,d}^P,\ n \in \mathcal{S}^T, d \in \mathcal{S}^{ALL}, \tag{5.11}$$

$$\sum_{d\in\mathcal{S}^D} X_{n,d} = 1, n \in \mathcal{S}^T, \tag{5.12}$$

$$Y_{n,d} \leq W_{n,d}^{\max} X_{n,d},\ n \in \mathcal{S}^T, d \in \mathcal{S}^{ALL}, \tag{5.13}$$

$$Y_{n,d} \leq W_{n,d},\ n \in \mathcal{S}^T, d \in \mathcal{S}^{ALL}, \tag{5.14}$$

$$Y_{n,d} \geq W_{n,d} - W_{n,d}^{\max}(1 - X_{n,d}),\ n \in \mathcal{S}^T, d \in \mathcal{S}^{ALL}, \tag{5.15}$$

$$Y_{n,d} \geq 0,\ W_{n,d} \geq 0, X_{n,d} \in \{0,1\},\ n \in \mathcal{S}^T, d \in \mathcal{S}^{ALL}, \tag{5.16}$$

$$Variables: X_{n,d},\ W_{n,d},\ Y_{n,d},\ T_{n,d}^S,\ T_{n,d}^E,\ T_{n,d}^T,\ n \in \mathcal{S}^T, d \in \mathcal{S}^{ALL}. \tag{5.17}$$

Where equation (5.1) guarantees that each IoT device can only run one application. Equation (5.2) is our objective of minimising the total response time and energy consumption. Equations (5.3) and (5.4) are to normalise the values of the response time and energy respectively. Equations (5.5) and (5.6) are to calculate the response time and energy consumption respectively in case of local execution, and equations (5.7) and (5.8) in case of edge or cloud offloading.

Equations (5.9) and (5.10) are to measure the task's start and end times. Constrain (5.11) is to ensure at a certain device d the start of a task n that succeeds another task $\hat{n} = n - 1$ has to wait for that task to finish, note that $\hat{n} = S^P_{n,d}$ where $S^P_{n,d}$ is a set of task $n's$ predecessor. Equation (5.12) is to guarantee executing all tasks by assigning each task to one device. While the constrains (5.13 - 5.16) are to calculate the waiting time that a task has to wait in order to be processed by a certain device.

5.3 Optimisation solver

The problem (5.2) is a mixed-integer linear programming problem that is difficult to solve using traditional heuristics and evolutionary algorithms such as Genetic algorithm and Particle Swarm, which cannot solve the problem directly (Hussein & Mousa 2020, Canali & Lancellotti 2019). Therefore, here we use Branch and Bound algorithm to solve the problem. It is a widely used method in solving integer and mixed-integer optimisation problems. The Branch and Bound algorithm is actually an enumeration of candidates solutions in the search space. It splits the original problem into branches of sub-problems before enumerating. The candidate solutions of a branch are checked against upper or lower estimated bounds of the optimal solution. The branch is discounted if it cannot produce a better solution than the best one found so far by the algorithm (He et al. 2014).

To do so, we developed a third party model in MATLAB that employs Gurobi optimiser 9.1 (Gurobi Optimization 2021) to solve the problem using Branch and Bound algorithm in order to obtain the optimal solutions using the same settings described in Table 4.3 for β and γ.

5.4 Comparison with the optimal

In this section, we compare the results from our proposed probless mechanism in Chapter 4 to the optimal solution given by our MILP optimisation using MATLAB, where the input values for the optimisation solver were obtained from the estimated values of our real testbed experiments in the previous chapter(See Section 4.4).

As we can see in figures 5.1, 5.2, PL-MEDICI is close to the optimal solution, with some applications being closer to the optimal values than others. For example, the response time and energy values of App1 (Face detection are 30.8% and 36.9% higher than the optimal values respectively). Similarly, for App3 (Radix sort), it is 20.8% for the response time and 30.2% for energy.

App1 and App2 are computational demanding applications and have been set to have relatively high γ values (0.6 and 0.4 respectively), meaning accuracy in estimation is more important, and thus the decision could choose devices that will allow the update of the knowledge (fresher network and execution times) of the system. This has led to PL-MEDICI, for App1 and App3, to favour some offloading devices that have not been chosen recently, instead of choosing devices that would minimise, possibly outdated, response times and energy consumption values. And since the optimisation does not consider in its decision the freshness (AoK), it resulted in choosing different offloading devices. We believe that choosing γ in a more optimal way could further improve PI-MEDICI to be closer to the optimal allocations.

The values of App2 (VBHRM), are only 9.7% and 4.8% higher than the optimal response time and energy consumption values respectively. As App2 is a time-critical application, it has been assigned a low γ, meaning the decision is mainly based on minimising response time and thus it is very close to the optimal solution.

This deviation from the optimal choices can also be seen in Figures 5.3,5.4, 5.5, which show the average percentage of tasks that were allocated to each device per application for PL-MEDICI and for the optimisation solution, where App2 is closer to the optimal distribution of tasks than the other two applications. However, the optimisation appropriately minimises cloud usage slightly when energy is important, and the task is computationally demanding. But It also makes more

usage of the cloud when the application is not computationally demanding, time-sensitive, or characterised by a large amount of data as input.

So, we can observe that choosing an appropriate γ value could further improve the performance of our mechanism. Towards this direction, our future work will involve investigating ways of determining those weight values in a more automated and optimal way that could influence the decision-making outcome and could also further improve the performance of our mechanism.

Figure 5.1: Total Response Time per Application

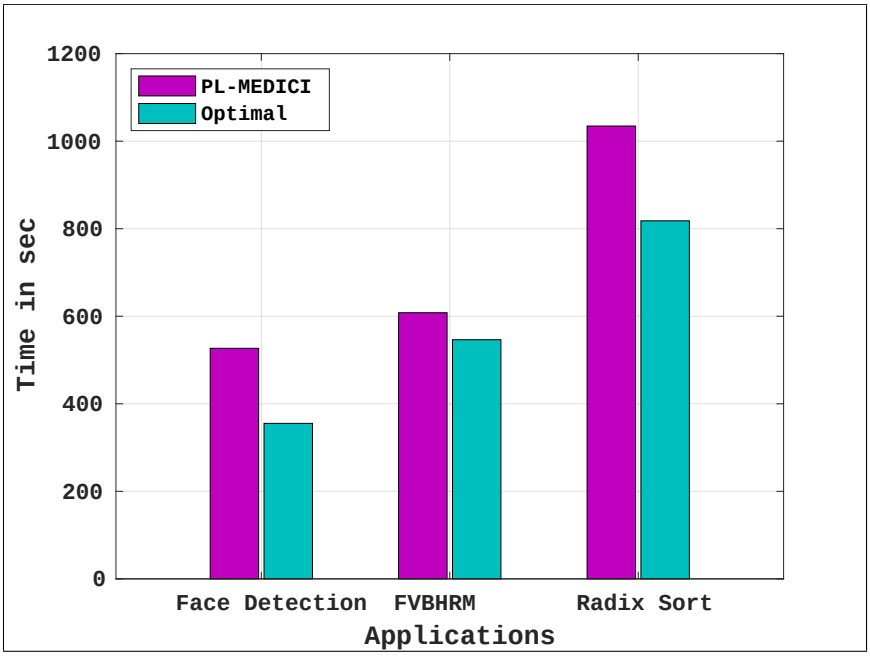


Figure 5.2: Total Energy Consumed per Application

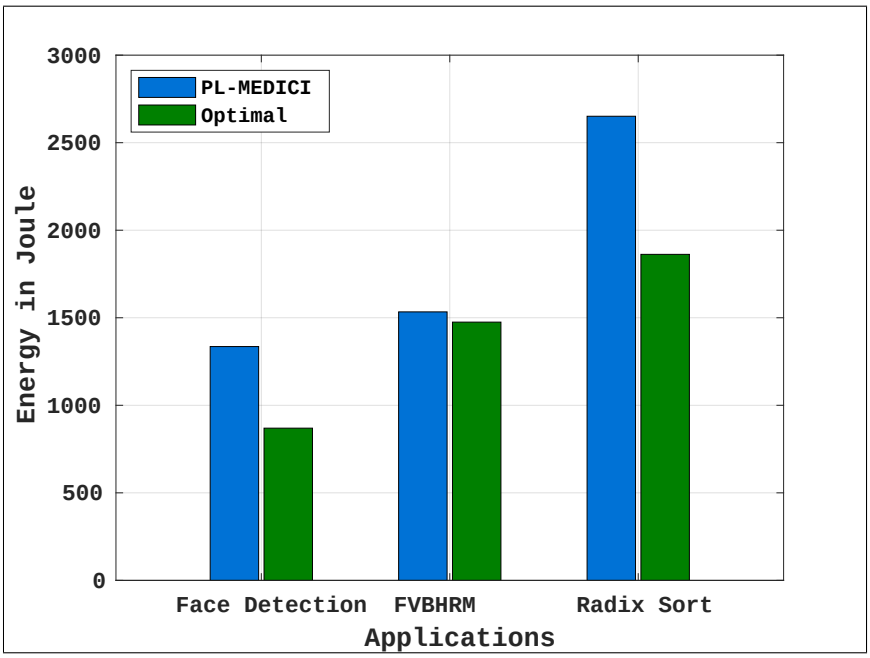


Figure 5.3: Percentage of tasks run at each device for App1 (Face detection)

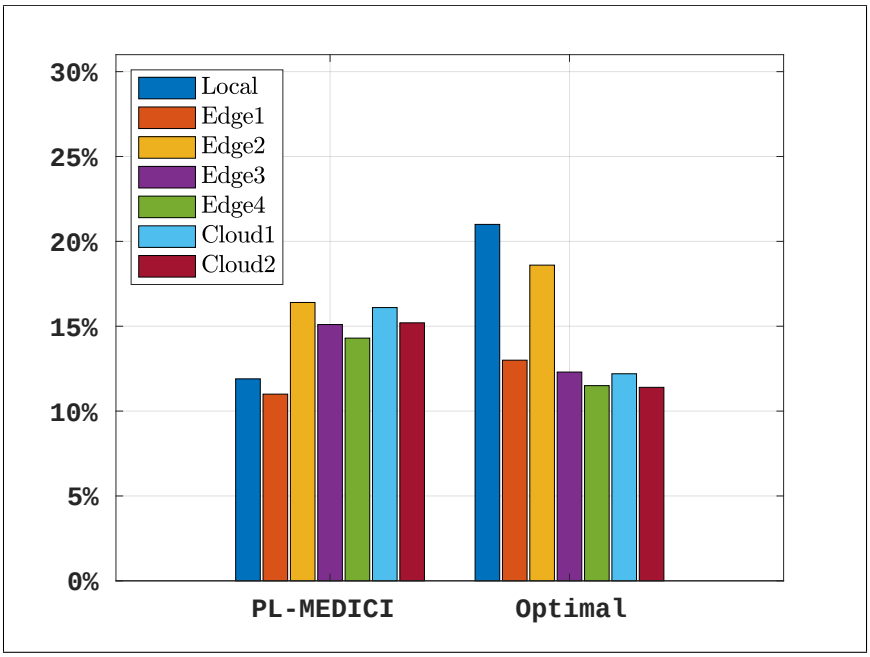


Figure 5.4: Percentage of tasks run at each device for App2 (VBHRM)

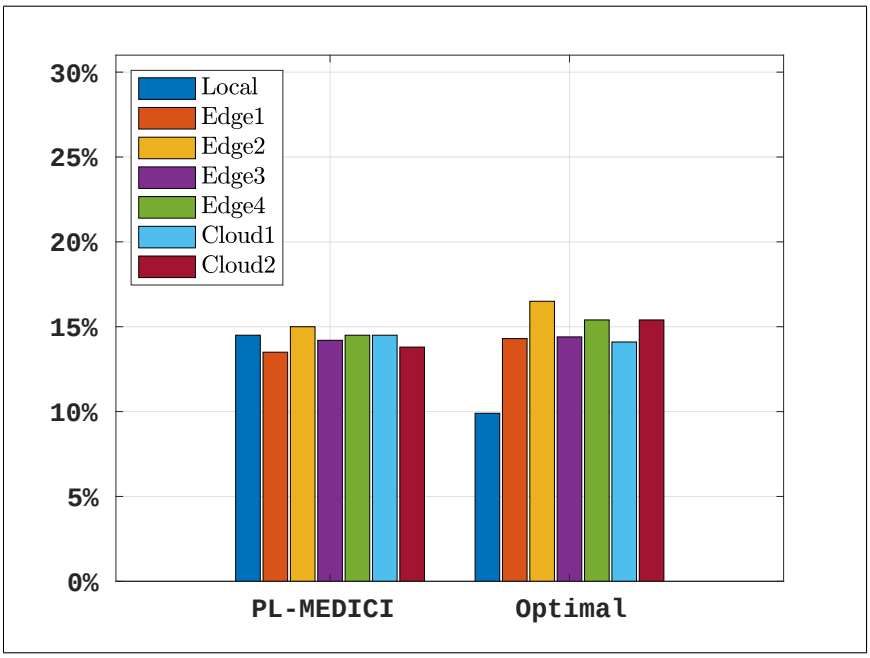
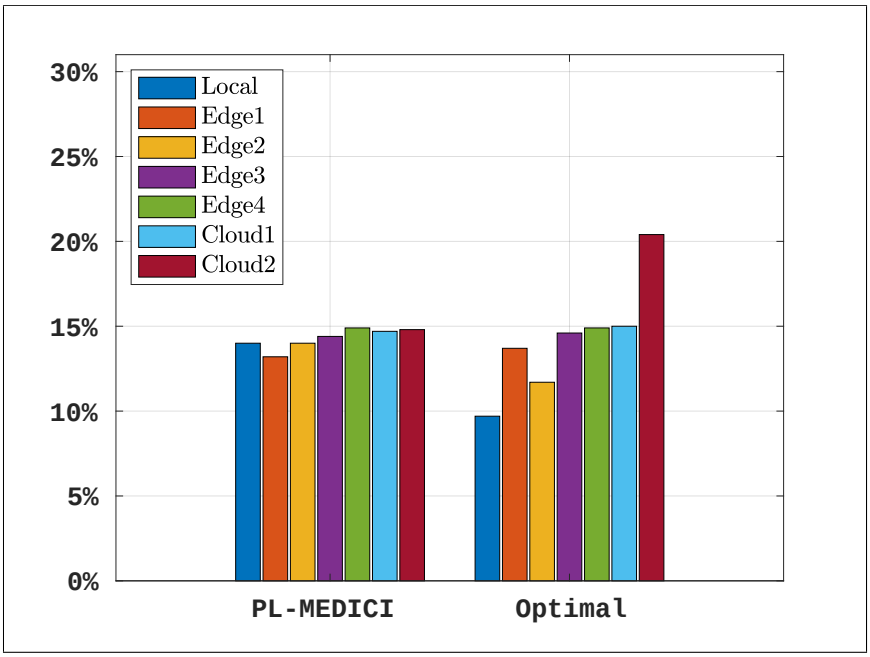


Figure 5.5: Percentage of tasks run at each device for App3 (Radix Sort)



5.5 Summary

In this chapter, we have presented our centralised mixed integer linear programming optimisation formulation that provides us with the theoretically optimal offloading decisions. This has allowed us to compare our solution against the theoretical optimum, given the same estimated input. The results showed that our offloading mechanism can be close to the obtained optimal solution in terms of both response time and energy consumption for some application than others due to the choice of the β and γ values. Therefore, we strongly believe that finding an optimal way of appointing those parameters will further improve our mechanism.

Chapter 6

Conclusion and Future work

6.1 Summary of the problem

Computation offloading is one of the primary technological enablers of IoT and edge computing, especially since modern IoT applications (e.g. smart surveillance, autonomous driving, AI-driven IoT, etc.) are becoming more computationally and energy demanding and operate within volatile network environments. Thus it is important for IoT devices to be able to efficiently and autonomously decide where it is more beneficial to process their tasks, by dynamically utilising the resources available. This, however, is challenging as different IoT devices might have different needs and the conditions under which they operate are constantly changing.

6.2 Summary of our contributions

1. **Comprehensive IoT offloading taxonomy**

We have surveyed a large variety of existing publications on IoT resource offloading

(both academic and industrial) and produced a comprehensive taxonomy (Chapter 2) covering not only computation offloading but also storage offloading, techniques such as partitioning, filtering, cashing and scheduling. Furthermore, we have looked into offloading from the industry perspective, by reviewing industrial solutions relevant to IoT offloading.

2. **Multi-criteria offloading decision support mechanism (MEDICI)**

In Chapter 3, we have proposed our initial multi-criteria offloading decision mechanism (MEDICI) for heterogeneous IoT devices based on modelling the response time and energy consumption of IoT edge environments. Depending on the energy-consciousness of an IoT user, the device can choose to minimise its own energy (selfish mode) or the total energy of all the devices involved (altruistic mode).

We have evaluated our mechanism by extending the popular EdgeCloudSim simulator. We 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). We also demonstrated that even when IoT devices decide in a selfish way to minimise their own metrics, the system as a whole still benefits and the overall goals are close to the altruistic mode.

3. **Probeless dynamic and decentralised offloading decision support mechanism (PL-MEDICI)**

In Chapter 4, we enhanced our MEDICI mechanism to be able to operate in real-world environments by estimating the response and network times based on historical data rather than predefined values or probing. Our proposed PL-MEDICI is the first probeless offloading mechanism, which estimates in a near real-time and distributed way the response times and energy consumption of all devices involved and dynamically decides where an IoT task should be processed.

In order to estimate those values PL-MEDICI does not probe the other devices but utilises lightweight statistical techniques that use historical data and takes also into consideration the age of the previous knowledge when making its decision.

We have evaluated PL-MEDICI's performance, by conducting experiments in a real environment, where six raspberry Pis, acting as IoT devices, run three different IoT applications with different characteristics and needs and where the network is emulated to be more realistic and provide dynamic network conditions. We compared our mechanism with three popular dynamic mechanisms that exist in the literature. Our experiments showed that our mechanism outperforms all others not only in respect to reducing the decision overheads, but also by reducing the overall processing and network times and the energy consumption of the IoT devices.

4. **Centralised optimal solution based on *a priori* information**

In Chapter 5, we have presented our mixed integer linear programming optimisation formulation that provides the theoretical optimal centralised offloading decisions. This has allowed us to compare our solution against the theoretical optimum given the same estimated input. The results showed that our offloading mechanism PL-MEDICI is close to the obtained optimal solution in terms of response time and energy consumption.

6.3 Open issues and future work

Despite the very good performance that our approaches demonstrate, there is still room for improvement. More specifically:

1. **Further improving PL-MEDICI through parameter tuning**

PL-MEDICI mechanism is a weighted multi-criteria offloading mechanism that allows each IoT application to consider the importance of each decision metric independently,

according to their needs. For example, IoT applications that are more time-critical (e.g. a healthcare IoT applications) give more weight to the response time metric rather than the energy consumption, while for more computationally demanding applications where energy consumption is important (e.g.face detection), the energy consumption metric has more weight in the offloading decision. This preference is denoted by the parameter β of our decision function (equation 4.12).

Another parameter that depends on the type of application is the parameter γ, which denotes how important it is to have up-to-date knowledge for each specific IoT application (e.g. if accuracy of estimation is very important AoK should be given more weight).

In our experiments, β and γ values were empirically chosen and remained the same throughout the experiments, by running experiments for each application (as widely done in the literature (Lin et al. 2013, Khoda et al. 2016*b*, Jaddoa et al. 2020, Chen et al. 2015) and finding the best values that correspond to the best goal function G of equation 4.12.

However, determining those weight values in a more automated and optimal way could influence the decision-making outcome and could further improve the performance of our mechanism.

Some promising methods that could potentially be used to chose the values of those weights in a more dynamic way could be the Entropy method (Deng et al. 2000, Al-Aomar 2010, Jahan et al. 2012). It is used to weight certain criteria in a given problem based on a certain amount of information. Mathematically, it represents the uncertainty of the criterion in the form of a discrete probability distribution in the Multi-Attribute Decision-Making(MADM) problem.

Additionally, other semi-dynamic methods that calculate the weights for the criteria based on predefined preferred values could be used, such as in (Wu et al. 2013), (Zhou et al. 2015), (Ravi & Peddoju 2015) and (Singla & Kaushal 2015) which use the Analytic Hierarchy Process (AHP) that employs a pairwise comparison to express relative strength or intensity of the decision criteria.

2. **Further improving our mechanism through better estimation of processing or network times**

The choice of polynomial regression was based on the behaviour of the selected applications and the need for a lightweight prediction mechanism. Application of different lightweight machine learning implementations, which might be more appropriate for larger-scale IoT environments could further improve our predictions without considerably greater overheads.

Also, the smoothing technique used to estimate the network times was chosen for its simplicity and the fact that it was lightweight. A more accurate way for estimating network delays could include more network parameters such as packet loss, propagation delays, packet duplication or packet corruption and assess the impact added to the delay of a task's overall response time. This however could be more computationally demanding, adding to the overhead of the estimation.

3. **Data overload**

Our current probeless offloading mechanism highly depends on the stored historical data of previous executions of tasks. This data is stored at the IoT device after the completion of each task. Over time, this may lead to a large size of data size for the IoT device itself, which hasn't been considered in this thesis. An easy solution would be to simply remove the oldest values and only keep values of a specific age, however, this could disadvantage devices that are not frequently chosen, as there might not be enough data to make accurate estimations. An optimal data eviction policy could potentially help with optimally storing the right amount of data Gupta et al. (2020).

4. **Expand our mechanism to be applied to other IoT applications**

In this thesis we concentrated on IoT applications whose performance was measured through response time and energy consumption and thus used those as criteria in the

decision. This though could be extended to include other types of IoT tasks. For example, having shown its usefulness in reducing energy cost and time for cyber intrusion detection, MEDICI has been adopted by the EU research project C4IIoT[1] ("Cybersecurity 4.0 for protecting the Industrial Internet of Things") to make offloading decisions for AI-based anomaly detection applications running on industrial IoT devices. For this, the metrics of MEDICI were enhanced to include, apart from the response time, also the accuracy/confidence of the anomaly detection. In this way MEDICI decides on whether an anomaly detection task should run at an edge or a cloud device so that the overall anomaly detection times are improved without compromising the accuracy of detection. Of course, the estimation of such parameters could prove challenging.

[1] https://www.c4iiot.eu

6.4 Final Remark

IoT devices with smarter and more computationally demanding tasks are expected to be the norm in the near future, helping individuals in every aspect of their everyday lives, from face recognition to protecting themselves against cyber attacks. Technologies such as computational offloading that enable this to happen in a sustainable way are therefore a clear direction of the current research that is related to enhancing IoT performance. The aim of this work was to contribute towards this direction.

References

Abbate, S., Avvenuti, M., Cola, G., Corsini, P., Light, J. & Vecchio, A. (2011), Recognition of false alarms in fall detection systems, *in* 'Consumer Communications and Networking Conference (CCNC), 2011 IEEE', IEEE, pp. 23–28.

Abebe, E. & Ryan, C. (2012), 'Adaptive application offloading using distributed abstract class graphs in mobile environments', *Journal of Systems and Software* **85**(12), 2755–2769.

Abro, A., Deng, Z., Memon, K. A., Laghari, A. A., Mohammadani, K. H. et al. (2019), 'A dynamic application-partitioning algorithm with improved offloading mechanism for fog cloud networks', *Future Internet* **11**(7), 141.

Ahn, S., Gorlatova, M. & Chiang, M. (2017), Leveraging fog and cloud computing for efficient computational offloading, *in* 'URTC', IEEE, pp. 1–4.

Al-Aomar, R. (2010), 'A combined ahp-entropy method for deriving subjective and objective criteria weights', *Int. J Ind. Eng. Theory Appl. Pract* **17**, 12–24.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015), 'Internet of things: A survey on enabling technologies, protocols, and applications', *IEEE Communications Surveys & Tutorials* **17**(4), 2347–2376.

Alam, M. G. R., Hassan, M. M., Uddin, M. Z., Almogren, A. & Fortino, G. (2019), 'Auto-

nomic computation offloading in mobile edge for iot applications', *Future Generation Computer Systems* **90**, 149–157.

Alelaiwi, A. (2019), 'An efficient method of computation offloading in an edge cloud platform', *Journal of Parallel and Distributed Computing* **127**, 58–64.

Ali, K. & Lhoták, O. (2012), Application-only call graph construction, *in* 'European Conference on Object-Oriented Programming', Springer, pp. 688–712.

Amazon (2018), 'Amazon Web Services (AWS) - Cloud Computing Services', `https://aws.amazon.com/`.

Ananthanarayanan, G., Bahl, P., Bodík, P., Chintalapudi, K., Philipose, M., Ravindranath, L. & Sinha, S. (2017), 'Real-time video analytics: The killer app for edge computing', *Computer* **50**(10), 58–67.

AWS (2019), 'Aws iot greengrass - amazon web services', `https://aws.amazon.com/greengrass/`. (Accessed on 10/21/2019).

Baktir, C., Sonmez, C., Ersoy, C., Ozgovde, A. & Varghese, B. (2018), 'Addressing the challenges in federating edge resources', *arXiv preprint arXiv:1803.05255* .

Balan, R. K., Gergle, D., Satyanarayanan, M. & Herbsleb, J. (2007), Simplifying cyber foraging for mobile devices, *in* 'Proceedings of the 5th International Conference on Mobile Systems, Applications and Services', ACM, pp. 272–285.

Beloglazov, A., Buyya, R., Lee, Y. C. & Zomaya, A. (2011), A taxonomy and survey of energy-efficient data centers and cloud computing systems, *in* 'Advances in computers', Vol. 82, Elsevier, pp. 47–111.

Benedetto, J. I., González, L. A., Sanabria, P., Neyem, A. & Navon, J. (2019*a*), 'Towards a practical framework for code offloading in the internet of things', *Future Generation Computer Systems* **92**, 424–437.

Benedetto, J. I., González, L. A., Sanabria, P., Neyem, A. & Navon, J. (2019*b*), 'Towards a practical framework for code offloading in the internet of things', *Future Generation Computer Systems* **92**, 424–437.

Bienert, J. (2016), An Introduction to ParStream, Technical report, Cisco.

Bittencourt, L. F., Diaz-Montes, J., Buyya, R., Rana, O. F. & Parashar, M. (2017), 'Mobility-aware application scheduling in fog computing', *IEEE Cloud Computing* **4**(2), 26–35.

Bonomi, F., Milito, R., Zhu, J. & Addepalli, S. (2012), Fog computing and its role in the internet of things, *in* 'Proceedings of the first edition of the MCC workshop on Mobile cloud computing', ACM, pp. 13–16.

Bors, A. G. & Nasios, N. (2009), 'Kernel bandwidth estimation for nonparametric modeling', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **39**(6), 1543–1555.

Botta, A., De Donato, W., Persico, V. & Pescapé, A. (2016), 'Integration of cloud computing and internet of things: a survey', *Future Generation Computer Systems* **56**, 684–700.

Bourke, A., O'brien, J. & Lyons, G. (2007), 'Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm', *Gait & posture* **26**(2), 194–199.

Canali, C. & Lancellotti, R. (2019), 'Gasp: genetic algorithms for service placement in fog computing systems', *Algorithms* **12**(10), 201.

Cao, Y., Chen, S., Hou, P. & Brown, D. (2015), Fast: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation, *in* 'International Conference on Networking, Architecture and Storage (NAS)', IEEE, pp. 2–11.

Castillo, J. C., Carneiro, D., Serrano-Cuerda, J., Novais, P., Fernández-Caballero, A. & Neves, J. (2014), 'A multi-modal approach for activity classification and fall detection', *International Journal of Systems Science* **45**(4), 810–824.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. & Gruber, R. E. (2008), 'Bigtable: A distributed storage system for structured data', *ACM Transactions on Computer Systems (TOCS)* **26**(2), 4.

Chen, S., Zheng, Y., Wang, K. & Lu, W. (2019), Delay guaranteed energy-efficient computation offloading for industrial iot in fog computing, *in* 'ICC 2019-2019 IEEE International Conference on Communications (ICC)', IEEE, pp. 1–6.

Chen, X., Jiao, L., Li, W. & Fu, X. (2015), 'Efficient multi-user computation offloading for mobile-edge cloud computing', *IEEE/ACM Transactions on Networking* **24**(5), 2795–2808.

Chen, X., Shi, Q., Yang, L. & Xu, J. (2018), 'Thriftyedge: Resource-efficient edge computing for intelligent iot applications', *IEEE Network* **32**(1), 61–65.

Chen, Y., Zhang, N., Zhang, Y. & Chen, X. (2018), 'Dynamic computation offloading in edge computing for internet of things', *IEEE Internet of Things Journal* **6**(3), 4242–4251.

Chowdhery, A., Levorato, M., Burago, I. & Baidya, S. (2018), Urban iot edge analytics, *in* 'Fog computing in the internet of things', Springer, pp. 101–120.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M. & Patti, A. (2011), Clonecloud: elastic execution between mobile device and cloud, *in* 'Proceedings of the sixth conference on Computer systems', ACM, pp. 301–314.

Cisco (2018), 'Cisco Annual Internet Report (2018–2023) White Paper', `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/`

annual-internet-report/white-paper-c11-741490.html. Last accessed 15 April 2021.

Cisco (2019), 'Cisco IoT System', https://www.cisco.com/c/m/en_us/solutions/internet-of-things/iot-system.html.

Craciunescu, R., Mihovska, A., Mihaylov, M., Kyriazakos, S., Prasad, R. & Halunga, S. (2015), Implementation of fog computing for reliable e-health applications, *in* '49th Asilomar Conference on Signals, Systems and Computers', IEEE, pp. 459–463.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R. & Bahl, P. (2010), Maui: making smartphones last longer with code offload, *in* 'Proceedings of the 8th international conference on Mobile systems, applications, and services', ACM, pp. 49–62.

DaCosta, F. (2013), *Rethinking the Internet of Things: a scalable approach to connecting everything*, Apress.

Deng, H., Yeh, C.-H. & Willis, R. J. (2000), 'Inter-company comparison using modified topsis with objective weights', *Computers & Operations Research* **27**(10), 963–973.

Devices, E. E. (2009), 'Watts up pro'.

Dimakis, A. G., Ramchandran, K., Wu, Y. & Suh, C. (2011), 'A survey on network codes for distributed storage', *Proceedings of the IEEE* **99**(3), 476–489.

Du, J., Zhao, L., Feng, J. & Chu, X. (2018), 'Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee', *IEEE Transactions on Communications* **66**(4), 1594–1608.

Dubey, H., Yang, J., Constant, N., Amiri, A. M., Yang, Q. & Makodiya, K. (2015), Fog data: Enhancing telehealth big data through fog computing, *in* 'Proceedings of the ASE BigData & SocialInformatics 2015', ACM, p. 14.

Edge & Fog Processing Module (2018), Cisco Kinetic, Technical report, Cisco.

Elazhary, H. (2017), 'Context-aware mobile application task offloading to the cloud', *Context* **8**(5).

Elbamby, M. S., Bennis, M. & Saad, W. (2017), Proactive edge computing in latency-constrained fog networks, *in* '2017 European Conference on Networks and Communications (EuCNC)', IEEE, pp. 1–6.

ETSI, G. et al. (2018), 'Multi-access edge computing (mec); phase 2: Use cases and requirements', *ETSI Standards Search* .

Fan, C.-T., Chang, Y.-S., Wang, W.-J. & Yuan, S.-M. (2012), Execution time prediction using rough set theory in hybrid cloud, *in* '2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing', IEEE, pp. 729–734.

Finnegan, M. (2013), 'Boeing 787s to create half a terabyte of data per flight, says virgin atlantic', *Computerworld UK* **6**.

Fox, G. C., Kamburugamuve, S. & Hartman, R. D. (2012), Architecture and measured characteristics of a cloud based internet of things, *in* 'International Conference on Collaboration Technologies and Systems (CTS)', IEEE, pp. 6–12.

Frankston, B. (2016), 'Mobile-edge computing versus the internet?: Looking beyond the literal meaning of mec', *IEEE Consumer Electronics Magazine* **5**(4), 75–76.

Gelenbe, E., Sakellari, G. & D'arienzo, M. (2008), 'Admission of QoS aware users in a smart network', *ACM Trans. on Autonomous and Adaptive Systems* **3**(1), 4.

Google Cloud (2018), 'Edge TPU – Run Inference at the Edge', `https://cloud.google.com/edge-tpu/`.

Goyal, T., Singh, A. & Agrawal, A. (2012), 'Cloudsim: simulator for cloud computing infrastructure and modeling', *Procedia Engineering* **38**(4), 3566–3572.

Gupta, D., Rani, S., Ahmed, S. H. & Hussain, R. (2020), 'Caching policies in ndn-iot architecture.'.

Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K. & Buyya, R. (2017), 'ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments', *Software: Practice and Experience* **47**(9), 1275–1296.

Gurobi Optimization, L. (2021), 'Gurobi optimizer reference manual'.
URL: *http://www.gurobi.com*

Härdle, W. (1990), *Applied nonparametric regression*, number 19, Cambridge university press.

Hassan, M. A., Bhattarai, K., Wei, Q. & Chen, S. (2014), 'Pomac: Properly offloading mobile applications to clouds', *Energy (J)* **25**, 50.

Hassan, M. A., Wei, Q. & Chen, S. (2015), Elicit: Efficiently identify computation-intensive tasks in mobile applications for offloading, *in* 'Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on', IEEE, pp. 12–22.

He, H., Daume III, H. & Eisner, J. M. (2014), 'Learning to search in branch and bound algorithms', *Advances in neural information processing systems* **27**, 3293–3301.

He, Q., Zhang, C., Ma, X. & Liu, J. (2017), 'Fog-based transcoding for crowdsourced video livecast', *IEEE Communications Magazine* **55**(4), 28–33.

Hossain, M. D., Huynh, L. N., Sultana, T., Nguyen, T. D., Park, J. H., Hong, C. S. & Huh, E.-N. (2020), Collaborative task offloading for overloaded mobile edge computing

in small-cell networks, *in* '2020 International Conference on Information Networking (ICOIN)', IEEE, pp. 717–722.

HPE, Saguna and AWS (2018), A Platform for Computing at the Mobile Edge: Joint solution with HPE, Saguna, AWS, Technical report, HPE, Saguna, and AWS.

Huawei Technologies Co., Ltd. (2015), NB-IOT – Enabling New Business Opportunities, Technical report, Huawei.

Huawei Technologies Co., Ltd. (2017), 'Huawei AR Series Agile Gateways Brochures', `https://www.huawei.com/minisite/iot/img/hw_ar_series_agile_gateways_brochure_en.pdf`.

Huawei Technologies Co., Ltd. (2018), 'IoT, Driving Verticals to Digitization', `https://www.huawei.com/minisite/iot/en/`.

Hubert, B. (2002), 'The wonder shaper'.
URL: *http://lartc.org/wondershaper/.*

Hussein, M. K. & Mousa, M. H. (2020), 'Efficient task offloading for iot-based applications in fog computing using ant colony optimization', *IEEE Access* .

IBM Cloud (2018), 'Watson IoT Platform Edge Overview', `https://console.bluemix.net/docs/services/IoT/edge/WIoTP_edge.html`.

Ibrahim, M. F., Jamal, M., Yahya, S. & Taib, M. N. (2012), 'Available bandwidth estimation in network-aware applications for wireless campus e-learning system', *Journal of Computer Networks and Communications* **2012**.

Igarashi, Y., Hiltunen, M., Joshi, K. & Schlichting, R. (2015), An extensible home automation architecture based on cloud offloading, *in* '18th International Conference on Network-Based Information Systems (NBiS)', IEEE, pp. 187–194.

Intel (2018), 'Internet of Things – Developer Journey', https://software.intel.com/en-us/iot/journey/digging-deeper.

Intharawijitr, K., Iida, K. & Koga, H. (2016), Analysis of fog model considering computing and communication latency in 5g cellular networks, *in* 'Pervasive Computing and Communication Workshops (PerCom Workshops), 2016 IEEE International Conference on', IEEE, pp. 1–4.

Jaddoa, A., Sakellari, G., Panaousis, E., Loukas, G. & Sarigiannidis, P. G. (2020), 'Dynamic decision support for resource offloading in heterogeneous internet of things environments', *Simulation Modelling Practice and Theory* **101**, 102019.

Jahan, A., Mustapha, F., Sapuan, S., Ismail, M. Y. & Bahraminasab, M. (2012), 'A framework for weighting of criteria in ranking stage of material selection process', *The International Journal of Advanced Manufacturing Technology* **58**(1-4), 411–420.

Jalali, F., Lynar, T., Smith, O. J., Kolluri, R. R., Hardgrove, C. V., Waywood, N. & Suits, F. (2019*a*), Dynamic edge fabric environment: Seamless and automatic switching among resources at the edge of iot network and cloud, *in* '2019 IEEE International Conference on Edge Computing (EDGE)', IEEE, pp. 77–86.

Jalali, F., Lynar, T., Smith, O. J., Kolluri, R. R., Hardgrove, C. V., Waywood, N. & Suits, F. (2019*b*), Dynamic edge fabric environment: Seamless and automatic switching among resources at the edge of iot network and cloud, *in* '2019 IEEE International Conference on Edge Computing (EDGE)', IEEE, pp. 77–86.

Jalali, F., Lynar, T., Smith, O. J., Kolluri, R. R., Hardgrove, C. V., Waywood, N. & Suits, F. (2019*c*), Dynamic edge fabric environment: Seamless and automatic switching among resources at the edge of iot network and cloud, *in* '2019 IEEE International Conference on Edge Computing (EDGE)', IEEE, pp. 77–86.

Jalali, F., Smith, O. J., Lynar, T. & Suits, F. (2017), Cognitive iot gateways: Automatic task sharing and switching between cloud and edge/fog computing, *in* 'Proceedings of the SIGCOMM Posters and Demos', ACM, pp. 121–123.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013), *An introduction to statistical learning*, Vol. 112, Springer.

Jiang, Z., Zhou, S., Guo, X. & Niu, Z. (2018), 'Task replication for deadline-constrained vehicular cloud computing: Optimal policy, performance analysis, and implications on road traffic', *IEEE Internet of Things Journal* **5**(1), 93–107.

Kapsalis, A., Kasnesis, P., Venieris, I. S., Kaklamani, D. I. & Patrikakis, C. Z. (2017), 'A cooperative fog approach for effective workload balancing', *IEEE Cloud Computing* **4**(2), 36–45.

Karatza, H. (2021), Large scale real-time distributed systems–resource allocation and scheduling issues, *in* '7th Conference on the Engineering of Computer Based Systems', pp. 1–2.

Kemp, R. (2014), 'Programming frameworks for distributed smartphone computing'.

Kemp, R., Palmer, N., Kielmann, T. & Bal, H. (2011), Energy efficient information monitoring applications on smartphones through communication offloading, *in* 'International Conference on Mobile Computing, Applications, and Services', Springer, pp. 60–79.

Ketykó, I., Kecskés, L., Nemes, C. & Farkas, L. (2016), Multi-user computation offloading as multiple knapsack problem for 5g mobile edge computing, *in* 'European Conference on Networks and Communications (EuCNC)', IEEE, pp. 225–229.

Khoda, M. E., Razzaque, M. A., Almogren, A., Hassan, M. M., Alamri, A. & Alelaiwi, A. (2016*a*), 'Efficient computation offloading decision in mobile cloud computing over 5g network', *Mobile Networks and Applications* **21**(5), 777–792.

Khoda, M. E., Razzaque, M. A., Almogren, A., Hassan, M. M., Alamri, A. & Alelaiwi, A. (2016*b*), 'Efficient computation offloading decision in mobile cloud computing over 5g network', *Mobile Networks and Applications* **21**(5), 777–792.

Ko, H., Lee, J., Jang, S., Kim, J. & Pack, S. (2019), 'Energy efficient cooperative computation algorithm in energy harvesting internet of things', *Energies* **12**(21), 4050.

Kosta, S., Aucinas, A., Hui, P., Mortier, R. & Zhang, X. (2012), Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, *in* '2012 Proceedings IEEE Infocom', IEEE, pp. 945–953.

Krishnaswamy, S., Zaslavsky, A., Loke, S. W. et al. (2002), Predicting run times of applications using rough sets, *in* 'Ninth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2002)', pp. 455–462.

Kristo, A., Vaidya, K., Çetintemel, U., Misra, S. & Kraska, T. (2020), The case for a learned sorting algorithm, *in* 'Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data', pp. 1001–1016.

Kumar, K. & Lu, Y.-H. (2010), 'Cloud computing for mobile users: Can offloading computation save energy?', *Computer* **43**(4), 51–56.

Lee, J. & Lee, J. (2018), 'Hierarchical mobile edge computing architecture based on context awareness', *Applied Sciences* **8**(7), 1160.

Lee, K., Murray, D., Hughes, D. & Joosen, W. (2010), Extending sensor networks into the cloud using amazon web services, *in* '2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications', IEEE, pp. 1–7.

Li, H., Ota, K. & Dong, M. (2018), 'Learning iot in edge: Deep learning for the internet of things with edge computing', *IEEE Network* **32**(1), 96–101.

Li, X., Ding, R., Liu, X., Yan, W., Xu, J., Gao, H. & Zheng, X. (2018), Comec: Computation offloading for video-based heart rate detection app in mobile edge computing, *in* '2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)', IEEE, pp. 1038–1039.

Li, X., Wan, J., Dai, H.-N., Imran, M., Xia, M. & Celesti, A. (2019), 'A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing', *IEEE Transactions on Industrial Informatics* **15**(7), 4225–4234.

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H. & Zhao, W. (2017), 'A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications', *IEEE Internet of Things Journal* **4**(5), 1125–1142.

Lin, Y.-D., Chu, E. T.-H., Lai, Y.-C. & Huang, T.-J. (2013), 'Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds', *IEEE Systems Journal* **9**(2), 393–405.

Lin, Y.-D., Chu, E. T.-H., Lai, Y.-C. & Huang, T.-J. (2015*a*), 'Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds', *IEEE Systems Journal* **9**(2), 393–405.

Lin, Y.-D., Chu, E. T.-H., Lai, Y.-C. & Huang, T.-J. (2015*b*), 'Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds', *IEEE Systems Journal* **9**(2), 393–405.

Liu, L., Chang, Z., Guo, X., Mao, S. & Ristaniemi, T. (2017), 'Multiobjective optimization for computation offloading in fog computing', *IEEE Internet of Things Journal* **5**(1), 283–294.

Liu, X., Yang, Q., Luo, J., Ding, B. & Zhang, S. (2018), 'An energy-aware offloading framework for edge-augmented mobile rfid systems', *IEEE Internet of Things Journal* **6**(3), 3994–4004.

Long, C., Cao, Y., Jiang, T. & Zhang, Q. (2018), 'Edge computing framework for cooperative video processing in multimedia iot systems', *IEEE Transactions on Multimedia* **20**(5), 1126–1139.

Loukas, G., Vuong, T., Heartfield, R., Sakellari, G., Yoon, Y. & Gan, D. (2018), 'Cloud-based cyber-physical intrusion detection for vehicles using deep learning', *Access* **6**, 3491–3508.

Loukas, G., Yoon, Y., Sakellari, G., Vuong, T. & Heartfield, R. (2017), 'Computation offloading of a vehicle's continuous intrusion detection workload for energy efficiency and performance', *SIMPAT* **73**, 83–94.

Lyu, X., Tian, H., Jiang, L., Vinel, A., Maharjan, S., Gjessing, S. & Zhang, Y. (2018), 'Selective offloading in mobile edge computing for the green internet of things', *IEEE Network* **32**(1), 54–60.

Ma, X., Lin, C., Zhang, H. & Liu, J. (2018), 'Energy-aware computation offloading of iot sensors in cloudlet-based mobile edge computing', *Sensors* **18**(6), 1945.

Mach, P. & Becvar, Z. (2017), 'Mobile edge computing: A survey on architecture and computation offloading', *IEEE Communications Surveys & Tutorials* **19**(3), 1628–1656.

Mahini, H., Rahmani, A. M. & Mousavirad, S. M. (2021), 'An evolutionary game approach to iot task offloading in fog-cloud computing', *The Journal of Supercomputing* **77**(6), 5398–5425.

Mahmud, R. & Buyya, R. (2019), 'Modelling and simulation of fog and edge comput-

ing environments using ifogsim toolkit', *Fog and edge computing: Principles and paradigms* pp. 1–35.

Mahmud, R., Kotagiri, R. & Buyya, R. (2018), Fog computing: A taxonomy, survey and future directions, *in* 'Internet of Everything', Springer, pp. 103–130.

Mahmud, R., Ramamohanarao, K. & Buyya, R. (2018), 'Latency-aware application module management for fog computing environments', *ACM Transactions on Internet Technology (TOIT)* **19**(1), 1–21.

Malawski, M., Juve, G., Deelman, E. & Nabrzyski, J. (2015), 'Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds', *Future Generation Computer Systems* **48**, 1–18.

Mehmeti, F. & Spyropoulos, T. (2014), Is it worth to be patient? analysis and optimization of delayed mobile data offloading, *in* 'INFOCOM', IEEE, pp. 2364–2372.

Meurisch, C., Gedeon, J., Nguyen, T. A. B., Kaup, F. & Muhlhauser, M. (2017), Decision support for computational offloading by probing unknown services, *in* '2017 26th International Conference on Computer Communication and Networks (ICCCN)', IEEE, pp. 1–9.

Microsoft Azure (2018), 'Azure IoT Edge', `https://azure.microsoft.com/en-gb/services/iot-edge/`.

Min, M., Xu, D., Xiao, L., Tang, Y. & Wu, D. (2017), 'Learning-based computation offloading for iot devices with energy harvesting', *arXiv preprint arXiv:1712.08768* .

Minerva, R., Biru, A. & Rotondi, D. (2015), 'Towards a definition of the internet of things (iot)', *IEEE Internet Initiative* **1**(1), 1–86.

Misra, S., Wolfinger, B. E., Achuthananda, M., Chakraborty, T., Das, S. N. & Das, S. (2019), 'Auction-based optimal task offloading in mobile cloud computing', *IEEE Systems Journal* **13**(3), 2978–2985.

Nazmudeen, M. S. H., Wan, A. T. & Buhari, S. M. (2016), Improved throughput for power line communication (plc) for smart meters using fog computing based data aggregation approach, *in* 'IEEE International Smart Cities Conference (ISC2)', IEEE, pp. 1–4.

Ning, Z., Dong, P., Kong, X. & Xia, F. (2018), 'A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things', *IEEE Internet of Things Journal* **6**(3), 4804–4814.

Niu, J., Song, W. & Atiquzzaman, M. (2014), 'Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications', *Journal of Network and Computer Applications* **37**, 334–347.

OpenCV (2019), `https://opencv.org/`. (Accessed on 10/23/2019).

OpenFog Consortium Architecture Working Group (2017), OpenFog Reference Architecture for Fog Computing, Technical report, OpenFog Consortium.

Ostertagová, E. (2012), 'Modelling using polynomial regression', *Procedia Engineering* **48**, 500–506.

Oueis, J., Strinati, E. C. & Barbarossa, S. (2015), The fog balancing: Load distribution for small cell cloud computing, *in* 'Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st', IEEE, pp. 1–6.

Park, S., Kwon, D., Kim, J., Lee, Y. K. & Cho, S. (2020), 'Adaptive real-time offloading decision-making for mobile edges: Deep reinforcement learning framework and simulation results', *Applied Sciences* **10**(5), 1663.

Patterson, D. A. (2004), 'Latency lags bandwith', *Communications of the ACM* **47**(10), 71–75.

Pelegris, P., Banitsas, K., Orbach, T. & Marias, K. (2010), A novel method to detect heart beat rate using a mobile phone, *in* '2010 Annual International Conference of the IEEE Engineering in Medicine and Biology', IEEE, pp. 5488–5491.

Pham, X.-Q. & Huh, E.-N. (2016), Towards task scheduling in a cloud-fog computing system, *in* 'Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific', IEEE, pp. 1–4.

Pyzyk, K. (2018), 'Anyvision, nvidia to develop surveillance cameras with facial recognition tech'.

Ra, M.-R., Priyantha, B., Kansal, A. & Liu, J. (2012), Improving energy efficiency of personal sensing applications with heterogeneous multi-processors, *in* 'Proceedings of the 2012 ACM Conference on Ubiquitous Computing', ACM, pp. 1–10.

Rahman, M. A., Hossain, M. S., Loukas, G., Hassanain, E., Rahman, S. S., Alhamid, M. F. & Guizani, M. (2018), 'Blockchain-based mobile edge computing framework for secure therapy applications', *IEEE Access* .

Rahmani, A. M., Liljeberg, P., Preden, J.-S. & Jantsch, A. (2017), *Fog computing in the internet of things: Intelligence at the edge*, Springer.

Rao, B. P., Saluia, P., Sharma, N., Mittal, A. & Sharma, S. V. (2012), Cloud computing for internet of things & sensing based applications, *in* 'Sixth International Conference on Sensing Technology (ICST)', IEEE, pp. 374–380.

Ravi, A. & Peddoju, S. K. (2015), 'Handoff strategy for improving energy efficiency and cloud service availability for mobile devices', *Wireless Personal Communications* **81**(1), 101–132.

Roberto De La Mora (2018), 'Cisco IOx: Making Fog Real for IoT', https://blogs.cisco.com/digital/cisco-iox-making-fog-real-for-iot. Last accessed 22 September 2021.

Saguna (2018), 'Multi Access Edge Computing Pioneer', https://www.saguna.net/.

Salcedo, D., D Guerrero, C. & Martinez, R. (2018), 'Available bandwidth estimation tools metrics, approaches and performance'.

Samie, F., Tsoutsouras, V., Bauer, L., Xydis, S., Soudris, D. & Henkel, J. (2016), Computation offloading and resource allocation for low-power iot edge devices, *in* 'WF-IoT', IEEE, pp. 7–12.

SAP Interface Integration Certification (2017), SAP Interface Scenarios: IOT Gateway Edge Device 1.0, Technical report, SAP.

Scoca, V., Aral, A., Brandic, I., De Nicola, R. & Uriarte, R. B. (2018), Scheduling latency-sensitive applications in edge computing., *in* 'CLOSER', pp. 158–168.

Shah-Mansouri, H. & Wong, V. W. (2018), 'Hierarchical fog-cloud computing for iot systems: A computation offloading game', *IEEE Internet of Things Journal* .

Sheng, J., Hu, J., Teng, X., Wang, B. & Pan, X. (2019), 'Computation offloading strategy in mobile edge computing', *Information* **10**(6), 191.

Shi, W., Cao, J., Zhang, Q., Li, Y. & Xu, L. (2016), 'Edge computing: Vision and challenges', *IEEE Internet of Things Journal* **3**(5), 637–646.

Shi, W. & Dustdar, S. (2016), 'The promise of edge computing', *Computer* **49**(5), 78–81.

Singla, C. & Kaushal, S. (2015), Cloud path selection using fuzzy analytic hierarchy process for offloading in mobile cloud computing, *in* '2015 2nd International Confer-

ence on Recent Advances in Engineering & Computational Sciences (RAECS)', IEEE, pp. 1–5.

Sinha, K. & Kulkarni, M. (2011), Techniques for fine-grained, multi-site computation offloading, *in* 'Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on', IEEE, pp. 184–194.

Sonmez, C., Ozgovde, A. & Ersoy, C. (2017), Edgecloudsim: An environment for performance evaluation of edge computing systems, *in* 'Second International Conference on Fog and Mobile Edge Computing (FMEC)', IEEE, pp. 39–44.

Stavrinides, G. L. & Karatza, H. D. (2017), Periodic scheduling of mixed workload in distributed systems, *in* '2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)', IEEE, pp. 94–99.

Stavrinides, G. L. & Karatza, H. D. (2019), 'A hybrid approach to scheduling real-time iot workflows in fog and cloud environments', *Multimedia Tools and Applications* **78**(17), 24639–24655.

Stavrinides, G. L. & Karatza, H. D. (2020), Multi-criteria scheduling of complex workloads on distributed resources, *in* '2020 International Conference on Computer, Information and Telecommunication Systems (CITS)', IEEE, pp. 1–5.

Suciu, G., Vulpe, A., Halunga, S., Fratu, O., Todoran, G. & Suciu, V. (2013), Smart cities built on resilient cloud computing and secure internet of things, *in* '19th International Conference on Control Systems and Computer Science (CSCS)', IEEE, pp. 513–518.

Sultana, S. (2017), 'An experimental survey of fog computing and iot: Escalate the cloud to where the things are', *IJSRST* **3**(8), 444–450.

Tan, H., Han, Z., Li, X.-Y. & Lau, F. C. (2017), Online job dispatching and scheduling in

edge-clouds, *in* 'IEEE INFOCOM 2017-IEEE Conference on Computer Communications', IEEE, pp. 1–9.

Taneja, M. & Davy, A. (2017), Resource aware placement of iot application modules in fog-cloud computing paradigm, *in* '2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)', IEEE, pp. 1222–1228.

Terzopoulos, G. & Karatza, H. D. (2016), 'Power-aware bag-of-tasks scheduling on heterogeneous platforms', *Cluster Computing* **19**(2), 615–631.

The Intel IoT Platform (2015), Architecture Specification White Paper, Technical report, Intel.

The Intel White Paper (2015), Maximizing IoT Edge Processing, Technical report, Intel.

Tychalas, D. & Karatza, H. (2020), 'A scheduling algorithm for a fog computing system with bag-of-tasks jobs: Simulation and performance evaluation', *Simulation Modelling Practice and Theory* **98**, 101982.

UK-Government, H. (2011), 'The carbon plan: Delivering our low carbon future'.

Vakilinia, S., Vakilinia, I. & Cheriet, M. (2017), Green process offloading in smart home, *in* 'IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)', IEEE, pp. 1–5.

Varghese, B., Wang, N., Nikolopoulos, D. S. & Buyya, R. (2017), 'Feasibility of fog computing', *arXiv preprint arXiv:1701.05451* .

Violeta, M., Mitja, L. & Matjaž, G. (2014), 'Combining domain knowledge and machine learning for robust fall detection', *Expert Systems* **31**(2), 163–175.

Wang, B., Wang, C., Huang, W., Song, Y. & Qin, X. (2020), 'A survey and taxonomy on task offloading for edge-cloud computing', *IEEE Access* **8**, 186080–186101.

Wang, T., Liang, Y., Zhang, Y., Zheng, X., Arif, M., Wang, J. & Jin, Q. (2020), 'An intelligent dynamic offloading from cloud to edge for smart iot systems with big data', *IEEE Transactions on Network Science and Engineering* **7**(4), 2598–2607.

Wortmann, F. & Flüchter, K. (2015), 'Internet of things', *Business & Information Systems Engineering* **57**(3), 221–224.

Wu, H. (2018), 'Multi-objective decision-making for mobile cloud offloading: A survey', *IEEE Access* .

Wu, H., Knottenbelt, W. & Wolter, K. (2015), Analysis of the energy-response time tradeoff for mobile cloud offloading using combined metrics, *in* 'ITC', IEEE, pp. 134–142.

Wu, H., Knottenbelt, W., Wolter, K. & Sun, Y. (2016), An optimal offloading partitioning algorithm in mobile cloud computing, *in* 'International Conference on Quantitative Evaluation of Systems', Springer, pp. 311–328.

Wu, H., Sun, Y. & Wolter, K. (2018), 'Energy-efficient decision making for mobile cloud offloading', *IEEE Transactions on Cloud Computing* .

Wu, H., Wang, Q. & Wolter, K. (2013), 'Optimal cloud-path selection in mobile cloud offloading systems based on qos criteria', *International Journal of Grid and High Performance Computing (IJGHPC)* **5**(4), 30–47.

Xu, D., Li, Y., Chen, X., Li, J., Hui, P., Chen, S. & Crowcroft, J. (2018), 'A survey of opportunistic offloading', *IEEE Communications Surveys & Tutorials* .

Yang, L., Cao, J., Yuan, Y., Li, T., Han, A. & Chan, A. (2013), 'A framework for partitioning and execution of data stream applications in mobile cloud computing', *ACM SIGMETRICS Performance Evaluation Review* **40**(4), 23–32.

Yoon, Y., Sakellari, G., Anthony, R. J. & Filippoupolitis, A. (2016), Energy-efficiency evaluation of computation offloading in personal computing, *in* 'International Symposium on Computer and Information Sciences', Springer, pp. 163–171.

Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J. & Yang, X. (2018), 'A survey on the edge computing for the internet of things', *IEEE Access* **6**, 6900–6919.

Yu, Y.-P., Kwan, B.-H., Lim, C.-L., Wong, S.-L. & Raveendran, P. (2013), Video-based heart rate measurement using short-time fourier transform, *in* '2013 International Symposium on Intelligent Signal Processing and Communication Systems', IEEE, pp. 704–707.

Zagha, M. & Blelloch, G. E. (1991), Radix sort for vector multiprocessors, *in* 'Conference on High Performance Networking and Computing: Proceedings of the 1991 ACM/IEEE conference on Supercomputing', Vol. 18, Citeseer, pp. 712–721.

Zao, J. K., Gan, T.-T., You, C.-K., Chung, C.-E., Wang, Y.-T., Rodríguez Méndez, S. J., Mullen, T., Yu, C., Kothe, C., Hsiao, C.-T. et al. (2014), 'Pervasive brain monitoring and data sharing based on multi-tier distributed computing and linked data technology', *Frontiers in human neuroscience* **8**, 370.

Zhang, M., Luo, H. & Zhang, H. (2015), 'A survey of caching mechanisms in information-centric networking', *IEEE Communications Surveys & Tutorials* **17**(3), 1473–1499.

Zhang, Q., Lin, M., Yang, L. T., Chen, Z., Khan, S. U. & Li, P. (2018), 'A double deep q-learning model for energy-efficient edge scheduling', *IEEE Transactions on Services Computing* .

Zhang, Y., Sun, W. & Inoguchi, Y. (2008), 'Predict task running time in grid environments based on cpu load predictions', *Future Generation Computer Systems* **24**(6), 489–497.

Zhao, M. & Zhou, K. (2019), 'Selective offloading by exploiting arima-bp for energy optimization in mobile edge computing networks', *Algorithms* **12**(2), 48.

Zhao, T., Zhou, S., Guo, X., Zhao, Y. & Niu, Z. (2016), Pricing policy and computational resource provisioning for delay-aware mobile edge computing, *in* '2016 IEEE/CIC International Conference on Communications in China (ICCC)', IEEE, pp. 1–6.

Zhong, J., Zhang, W., Yates, R. D., Garnaev, A. & Zhang, Y. (2019), Age-aware scheduling for asynchronous arriving jobs in edge applications, *in* 'IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)', IEEE, pp. 674–679.

Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N. & Buyya, R. (2015), A context sensitive offloading scheme for mobile cloud computing service, *in* '2015 IEEE 8th International Conference on Cloud Computing', IEEE, pp. 869–876.